Uttar Pradesh Rajarshi Tandon Open University

Master in Computer Applications

MCS-112

Java Programming

Uttar Pradesh Rajarshi Tandon
Open University

Master in Computer Applications

MCS-112

Java Programming

BLOCK

1

OBJECT ORIENTED METHODOLOGY AND JAVA

UNIT-1

Object Oriented Programming

UNIT-2

Java Language Basics

UNIT-3

Expressions Statements and Arrays

Course Design Committee

Prof. Ashutosh Gupta — **Chairman**
Director (In-charge)
School of Computer and Information Science, UPRTOU Allahabad

Prof. Suneeta Agarwal — **Member**
Department of CSE
MNNIT Allahabad, Prayagraj

Dr. Upendra Nath Tripathi — **Member**
Associate Professor, Department of Computer Science
Deen Dayal Upadhyaya Gorakhpur University, Gorakhpur

Dr. Ashish Khare — **Member**
Associate Professor, Department of Computer Science
University of Allahabad, Prayagraj

Dr. Marisha — **Member**
Assistant Professor (Computer Science),
School of Science, UPRTOU Allahabad

Mr. Manoj Kumar Balwant — **Member**
Assistant Professor (computer science),
School of Sciences, UPRTOU Allahabad

Course Preparation Committee

Er. Pooja Yadav — **Author (Unit 1 to 14)**
Asst Professor, Dept. of Computer Science & Information Technology, MJPRU, Bareilly

Prof. Abhay Saxena — **Editor**
Professor, DSVV, Shantikunj, Haridwar (UK)-249411

Prof. Ashutosh Gupta — **Director (In-charge)**
School of Computer and Information Science, UPRTOU Allahabad

Dr. Marisha, — **Coordinator**
Asstt. Professor (Computer Science), School of Science, UPRTOU, Prayagraj

Mr. Manoj Kumar Balwant — **Coordinator**
Asstt. Professor (Computer Science), School of Science, UPRTOU, Prayagraj

ISBN :

Printed and Published by Dr. Arun Kumar Gupta Registrar, Uttar Pradesh Rajarshi Tandon Open University, 2020.
Printed By : Chandrakala Universal Pvt. 42/7 Jawahar Lal Neharu Road, Prayagraj.

UNIT-1 OBJECT ORIENTED PROGRAMMING

Structure :

1.1 OBJECTIVE

In this unit, our objective is to review the OOPs concept, what those concepts in regard of java are and how many features of OOPs is being followed by java. We will also compare the procedure oriented and object oriented approaches.

1.2 INTRODUCTION

Automation is about solving the problems via digital devices. To solve the real world problem we must follow any predefined methodology or approach. There have been several approaches since the inception and as the time moves on refinement is always done. We start with the solution of simple problems to complex and dynamic problems. These days we are also working on automation of software engineering.

There are some predefined approaches for the implementation of problems. In this unit we will discuss the various programming paradigms which define the way to solve some problems. We will also discuss the object oriented approach with class and object.

1.3 PARADIGMS OF PROGRAMMING LANGUAGES

Paradigm can be defined as a method to solve any problem or to do some task. So programming paradigm can be defined as an approach to solve problem using any programming language. It is a method to solve a problem with tools and techniques via following any approach. There are several programming languages that are known to us but all of them need to follow some strategy at implementation. This methodology/strategy is known as paradigms.

There are several programming paradigms:

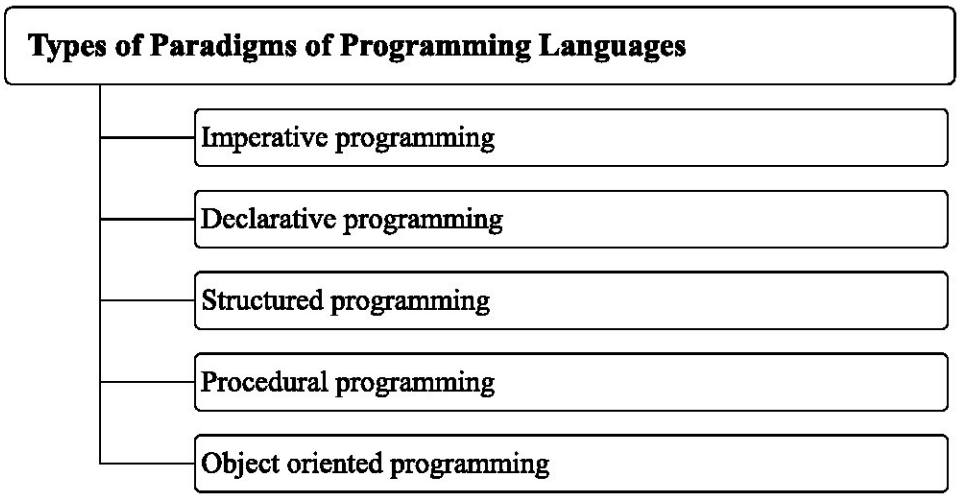


Figure 1.1

1.3.1 IMPERATIVE PROGRAMMING

It is the Programming with an explicit sequence of commands that updates the state. Imperative programming is the style of programming in which there is a sequence of statements that change the state of the program. Imperative programming states how to do something. It is very simple to implement. Some examples of imperative programming languages are: Algol and Pascal etc.

1.3.2 DECLARATIVE PROGRAMMING

Declarative programming mainly focuses on the logic of the problem solution. It does not contain if-else, loop or any control flow statement. It focuses on what should be the result rather than how to obtain the result. SQL, XQuery is the best example of this category.

1.3.3 STRUCTURED PROGRAMMING

Structured programming is a type of programming where control flow is defined by loops, conditionals, and subroutines, rather than via goto statement. This is the programming with clean, goto-free, nested control structures. Some examples of this category are: Algol 60, PL/I, Pascal, C, Ada 83, Modula etc.

1.3.4 PROCEDURAL PROGRAMMING

This programming paradigm mainly focuses on procedure. There is no as such difference in between procedural and imperative approach. We may define procedures in this approach. So it additionally has the ability to reuse the code. It was boon at that time when it was in use because of its reusability. Sometimes it is also called Imperative programming with procedure calls. C is one of the example of this type. It follows top down approach.

1.3.5 OBJECT ORIENTED PROGRAMMING

In object-oriented, everything is represented as an object. It follows the bottom up approach. It is used to solve real world and complex problems. This approach is not suitable for small and simple problems. For example, if we think of chair, table, student, stock, account, teacher, manager etc., everything is an object. Every object has some characteristics and methods. Multiple objects of same characteristics belong to one class. There may be multiple classes and multiple objects in one program. We have to be very alert while defining those classes and objects. Java, C#, C++ are some of the examples of this category.

Check Your Progress

1. Compare object oriented and procedure oriented programming paradigms.
2. Make a list of 5 programming languages of each programming paradigm.
3. State True / False
   a. Pascal follows Object Oriented programming paradigm.
   b. Ruby follows Object Oriented programming paradigm.
   c. COBOL supports imperative programming paradigm.

1.4 EVOLUTION OF OBJECT ORIENTED METHODOLOGY

The Object Oriented Methodology is most commonly used approach now a days. OOM is being used for designing large and complex problems. Before OOM many programming approaches existed which had many limitations.

These programming approaches have been passing through rapid revolutionary phases just like computer hardware. Initially for designing small and simple programs, at that time machine language was used. There after came the Assembly Language which was used for designing larger programs as compared

to previous languages. Both machine language and Assembly languages are machine dependent. Then Procedural Programming Approach came which enabled us to write larger and hundred lines of code. Then a new programming approach called Structured Programming Approach was evolved for designing medium sized programs. In 1980, as the size of the program was increasing, a new methodology Object Oriented Methodology came into existence.

Figure 1.2

There is brief to some object oriented programming language:

1.4.1 SIMULA

By the 1960s, programmers realized that programming systems needed to be broken up into small, manageable pieces. The introduction of Simula-67 brought with it the first true programming object, classes, and a form of inheritance; therefore, Simula is an important milestone in any discussion on O-O programming languages. The language was designed by Dahl, Myhrhaug, and Nygaard at the Norwegian Computing Centre at Oslo, Norway. The initial version of the language, Simula-1, was introduced in 1966. The programming modules defined by Simula were not based on procedures, but on actual physical objects. Simula had a novel way of presenting the object, so that each object has its own behaviour and data.

1.4.2 SMALLTALK

Many consider that the first truly O-O language was Smalltalk, developed at the Learning Research Group at Xerox's Palo Alto Research Centre in the early 1970s. In Smalltalk, everything is really an object that enforces the O-O paradigm. It is virtually impossible to write a program in Smalltalk that is not O-O. This is not the case for other languages that support objects, such as C++ and Visual Basic (and Java, for that matter).

1.4.3 C++

Although Smalltalk gave O-O development a certain amount of legitimacy in the marketplace, it took C++ to bring O-O development what it really needed: widespread acceptance in the marketplace. For this reason, C++ may well be the most important O-O language. Not until C++ was released and the industry started supported it with a vengeance did O-O development become mainstream. Before the initial standardization in 1998, C++ was developed by Danish computer scientist Bjarne Stroustrup at Bell Labs. C++ was originally implemented in 1982 under the name C with Classes.

1.4.4 JAVA

Java's Originates from consumer electronics. In 1991, Sun Microsystems began to investigate how it might exploit this growing market. Java was conceived by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan at Sun Microsystems, Inc. in 1991. It took 18 months to develop the first working version. This language was initially called "Oak," but was renamed "Java". Between the initial implementation of Oak in the fall of 1992 and the public announcement of Java in the spring of 1995, many more people contributed to the design and evolution of the language. We will discuss more about java in coming units.

1.4.5 C# .NET

C# was designed by Anders Hejlsberg. Microsoft responded to the popularity of Java by producing a version of Java called Visual J++. However, Microsoft decided on a more comprehensive response. Using many of the ground breaking that Java implemented, Microsoft developed a language called C# that became the foundation for the companies .NET platform. As with Java, C# relied heavily on the success and failures of earlier languages. The .NET development environment includes many of the really good characteristics of several other platforms. C#, like Java, uses C/C++ syntax. C# incorporates many of concepts introduced by the initial Java released (which I will list later in the article). The C# platform also builds upon many of the powerful features of the VB6 and Visual C++ environments. Here is a table showing the comparison between java and .net.

JAVA vs. .NET		
S.No	JAVA	.NET
1	It is technology of SUN.	It is technology of Microsoft.
2	It is language and technology both.	It is technology only.
3	It does not support multi language support.	It supports multi languages in one project development.
4	SUN does not provides its own IDE.	Microsoft provides its IDE (Visual Studio).
5	It may be developed on approximately on all platforms.	It may not be developed on all platforms.
6	It is approximately platform independent.	It is not completely platform independent.

Table 1.1

Check Your Progress

1. Compare object oriented and procedure oriented programming paradigms.
2. Give some differences between java and .net.
3. Make a list of founder of top object oriented programming languages.

1.5 BASIC CONCEPTS OF OO APPROACH

Earlier programming was restricted to procedure oriented programming but it was sufficient to solve the all real world problems. Because many problems were dynamic in nature so we need a different approach. In this approach, the major requirement is to focus more on data as compared to procedure because procedure will be different in different scenarios. That's why object oriented approach was developed.

In this approach we use objects which may be any instance. For Example any chair, table, student, school, hospital etc. may be an object. There may be multiple objects of one category. But each object of one category has some common fields for example if we have multiple students, all of them will have name, roll number, age etc. And their values may be different. So an object can be considered a "thing" that can perform a set of activities. The set of activities that the object performs defines the object's behaviour.

Object-oriented programming attempts to provide a model for programming based on objects. Object-oriented programming integrates code and data using the concept of an "object". In object oriented approach data and functions are not freely moved as they were moving in procedure oriented programming. We need this kind of restriction so that we may run a complicated and lengthy problem. Besides movement, data and functions may also not be directly accessible anywhere. Addition to this, one concept may also be reused at other places to save the time.

Here are some features of object-oriented programming approach: *Class, Object, abstraction, encapsulation, data hiding, polymorphism* and *inheritance.*

Class : A class is a structure of making a user-defined data type which holds the entire set of data of an object. So, a class is basically a blueprint or a template of a set of objects which shares some common properties and behaviours. We can also say that a class is a collection of similar type of objects. Class is the collection of data members and methods.

Object : Objects are the basic identifiable runtime entities in object-oriented programming. Objects may represent a real-world entity like a person, a car, a place, a house etc. For instance, we can say 'car' is an object which has some special characteristics like the 'number of gears', 'colour' etc. and it also holds

some functions like 'braking', 'accelerating' and so on. So, an object represents an entity that can store data and has its interface through functions.

Abstraction : Abstraction is the feature to hide the programming code to outside world and reveals its required information. Abstraction saves us form dealing with complicated and lengthy internal details. We just need to know the essential part for implementation.

For example, if we want to implement the banking system, then customer only needs to know about deposit and withdrawal forms, rest internal details are of no use for him. Internal processing will be there but the customer only need to know the interfacing with forms. In the same way, in programming if we are implementing any problem, then we only need to know how to implement the pre solved things. In simpler way, we can say abstraction hides the complicated details and reveals the essential information only.

Encapsulation : Encapsulation is the feature to wrap up the data members and methods into a single unit. Wrapping the members in a single unit restrict it to be accessed directly outside that unit. That unit is known as class. Encapsulation restricts the free flow of the data across the program. It also ensures that outside data to the class may also not be accessed inside the class. So we may also say that encapsulation creates a layer between inside class members and outside class members. Whenever we create the object of that class, that object may access only those members. It means, data of one object will be different from data of other object. One object cannot update the data of other object directly. If we need to update the data of one object, we must have one method for that in the class.

Data Hiding : Data hiding restricts the data member accessing via object / inheritance. It is used to implement the data security in class. There are three visibility modes for data accessing- public, private and protected. Public mode is used to access the members via object and inheritance. Private mode is used to restrict the member access outside the class. Protected mode is used to restrict the member accessing only via inheritance. So private visibility mode hides the data to be accessed via object. Without object / inheritance any member cannot be accessed outside the class. Members may be accessed inside the class directly.

Inheritance : Inheritance is the ability to reuse the members of one class into other class without redefining them. The class which is to be accessed is called super class or base class. The class in which members will be accessed is known as sub class or derived class. In C++, we may inherit more than one class into any class.

Polymorphism : Polymorphism is the feature which enables any object to act differently in different scenarios. For example, there is a boy, at home- he is son, at school- he is student, at sports ground- he is sportsman but he is the same boy. He will act differently at different scenarios. Polymorphism is again very important feature of object oriented approach as we are more focused on real world problems and real world problems are dynamic in nature.

Check Your Progress

1. Explain the basic concept of OOPs.
2. Differentiate class and object.
3. Explain the connection between abstraction, hiding and encapsulation.
4. State True / False
   a. Java is procedure oriented programming language.
   b. C++ is object oriented programming language.

1.6 COMPARISON OF OBJECT ORIENTED AND PROCEDURE - ORIENTED APPROACHES

Procedure-oriented Programming (POP) and Object-oriented programming (OOP) both are the programming approaches, which uses high-level language for programming. A Program can be written in both the languages, but if the task is highly complex, OOP operates well as compared to POP. In POP, the 'data security' is at risk as data freely moves in the program, as well as, 'code reusability' is not achieved which makes the programming lengthy, and hard to understand. Large programs lead to more bugs, and it increases the time of debugging. All these flaws lead to a new approach namely "object-oriented programming". In object-oriented programming's main concern is given on 'data security'; it binds the data closely to the functions which operate on it. It also resolves the problem of 'code reusability', as if a class is created, its multiple instances (objects) can be created which reuses the members and member functions defined by a class.

There are some differences between both of them as listed below:

PROCEDURE ORIENTED VS. OBJECT ORIENTED PROGRAMMING LANGUAGE		
S. No	**Procedure Oriented**	**Object Oriented**
Divided Into	In POP, program is divided into small parts called **functions**.	In OOP, program is divided into parts called **objects**.
Importance	Importance is given to functions as well as **sequence** of actions to be done.	Importance is given to the data.
Approach	POP follows **Top Down approach**.	OOP follows **Bottom Up approach**.

Access Specifiers	POP does not have any access specifier.	OOP has access specifiers named Public, Private, Protected etc.
Data Movement	Data can move freely from function to function in the system.	Objects can move and communicate with each other through member functions.
Expansion	To add new data and function in POP is not so easy.	OOP provides an easy way to add new data and function.
Data Access	Global data is used for sharing.	Data cannot move easily from function to function.
Data Hiding	POP does not have any proper way for hiding data so it is **less secure**.	OOP provides Data Hiding so provides **more security**.
Overloading	In POP, Overloading is not possible.	In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.
Examples	Basic, VB, FORTRAN, Pascal	C++, JAVA, VB.NET, C#.NET

Table 1.2

Check Your Progress

1. Give some examples of procedure oriented programming languages with their founder and foundation year.
2. Give a list of some object oriented programming languages with their founder and foundation year.

1.7 BENEFITS OF OOPS

Some of the **advantages** of object-oriented programming include:

1. **Improved software-development productivity:**

   When we are in the phase of software development and our project is large enough for our team, in that case, we may use this approach. This approach provides us the facility to divide our code into classes. We may assign them into teams and multiple teams may work on the project simultaneously without the interruption of other team. This will make the complexity of the project as negligible.

2. **Improved software maintainability:**

   If we have already divided the project into teams, then maintenance of the project is also easy and efficient because we just have to maintain the specific segment rather than the entire project.

3. **Faster development:**

   Division of the project into teams also reduces the time required for development besides the complexity.

4. **Lower cost of development:**

   Hiring the teams for specific segment will reduce the project cost, as we don't need to hire all the project members for one project all the time. That team may give their services to other projects as well, and that's how cost is reduced to an extent in our company.

5. **Higher-quality software:**

   Definitely quality is going to be one of the most important factor in the time to come. If we need to give quality, we have to hire expert professional. And with this approach, we may hire the expert project wise, if we have limited project. Or if we have multiple projects, we may use that team into multiple projects. That's how we may produce the high-quality software.

6. **Increased reusability via inheritance:**

   OOPs support inheritance, it gives us the reusability. Inheritance is one of the most important feature of OOPs. We may reuse one class or package in one program or it may also be used in any other program. This feature is also known as WORA- Write Once Run Anytime.

7. **Best suitable for complex problems:**

   OOPs was required because procedure oriented approach was not suitable for complex problems. There were many issues like free-flow of data, security, reusability and many more. So professionals design OOPs. It is best suitable for complex problem due to abstraction, inheritance, data hiding, polymorphism and most importantly Class & Objects.

8. **Flexibility through polymorphism:**

   Polymorphism gives us the compile time and runtime flexibility. Compile time flexibility is provided via overloading and runtime flexibility is provided by overriding.

9. **Modularity for easy trouble shooting:**

   If we develop any project, errors are surely to come. OOPs give us the modularity. Modularity provides us the easy and fast trouble shooting

because we need not to go through all the modules; we only need to focus on the module which is generating problem.

Check Your Progress

1. Explain the benefits of OOPs.
2. Differentiate compile time and runtime polymorphism.

1.8 APPLICATIONS OF OOPS

There are several applications of OOPs. In fact OOPs is used in almost all complicated system development. It is also used in approximate all real world problem solutions.

For example User interface design such as windows, menu, Real Time Systems, Simulation and Modelling, Object oriented databases, AI and Expert System, Neural Networks and parallel programming, Decision support and office automation systems etc.

Some of them are listed as :

1. **Client-Server Systems**

   Object-oriented Client-Server Systems provide the IT infrastructure, creating object-oriented Client-Server Internet (OCSI) applications. Here, infrastructure refers to operating systems, networks, and hardware.

2. **Object-Oriented Databases**

   They are also called Object Database Management Systems (ODBMS). These databases store objects instead of data, such as real numbers and integers. Objects consist of the following:

   Attributes: Attributes are data that defines the traits of an object. This data can be as simple as integers and real numbers. It can also be a reference to a complex object.

   Methods: They define the behavior and are also called functions or procedures.

3. **Object Oriented Databases**

   These databases try to maintain a direct correspondence between the real-world and database objects in order to let the object retain their identity and integrity. They can then be identified and operated upon.

4. **Real-Time System Design**

Real time systems inherit complexities that makes difficult to build them. Object-oriented techniques make it easier to handle those complexities. These techniques present ways of dealing with these complexities by providing an integrated framework which includes schedulability analysis and behavioural specifications.

5. Simulation and Modelling System

It's difficult to model complex systems due to the varying specification of variables. These are prevalent in medicine and in other areas of natural science, such as ecology, zoology, and agronomic systems. Simulating complex systems requires modelling and understanding interactions explicitly. Object-oriented Programming provides an alternative approach for simplifying these complex modelling systems.

6. Hypertext and Hypermedia

OOP also helps in laying out a framework for Hypertext. Basically, hypertext is similar to regular text as it can be stored, searched, and edited easily. The only difference is that hypertext is text with pointers to other text as well.

Hypermedia, on the other hand, is a superset of hypertext. Documents having hypermedia, not only contain links to other pieces of text and information, but also to numerous other forms of media, ranging from images to sound.

7. Neural Networking and Parallel Programming

It addresses the problem of prediction and approximation of complex time-varying systems. Firstly, the entire time-varying process is split into several time intervals or slots. Then, neural networks are developed in a particular time interval to disperse the load of various networks. OOP simplifies the entire process by simplifying the approximation and prediction ability of networks.

8. Office Automation Systems

These include formal as well as informal electronic systems primarily concerned with information sharing and communication to and from people inside as well as outside the organization. Some examples are: Email, Word processing, Web calendars, Desktop publishing etc.

9. CIM/CAD/CAM Systems

OOP can also be used in manufacturing and design applications as it allows people to reduce the effort involved. For instance, it can be used while designing blueprints, flowcharts, etc. OOP makes it possible for the designers and engineers to produce these flowcharts and blueprints accurately.

10. AI Expert Systems

These are computer applications which are developed to solve complex problems pertaining to a specific domain, which is at a level far beyond the reach of a human brain.

Check Your Progress

1. List the application of OOPs.
2. State True / False:
   a. AI system is best designed via procedural oriented approach.
   b. Networking based applications are developed via object oriented approach.

1.9 CLASSES AND OBJECTS

Class

Class is the collection of data members and methods. Methods are also known as member functions. Data members are used to store the values and methods are used to be called. Data members are generally defined as private so that they may be hidden from outside the class and methods are generally made public so that object may access them.

- Class is also known as template data.
- Class is the user defined data type.
- Class is the logical concept of data because it does not consume memory at runtime.
- Class acts as a blue print for any problem as it has to be defined only once.
- Class may also be considered as the collection of similar type of objects.
- In Class members are encapsulated so members of class cannot be used directly outside the class.
- In Class members are encapsulated so outside data cannot be accessed inside the class directly.
- Class members may have different modifiers.
- Class members may have different visibility modes (access specifier).
- Classes may be nested or local.
- There may be multiple classes in one program.

Object :

Object is declared to store the values and execute the methods defined inside the class. The may access the public members of their class. They cannot access private or protected members of the class. There may be multiple objects of one class depending on their need in the program. All objects will consume memory separately. Updating data members in one object will not update values in other objects. We need to update each object separately as their allocated memory is different.

- An object is the instance of class.
- We can create multiple objects of one class.
- An object consumes memory at run time.

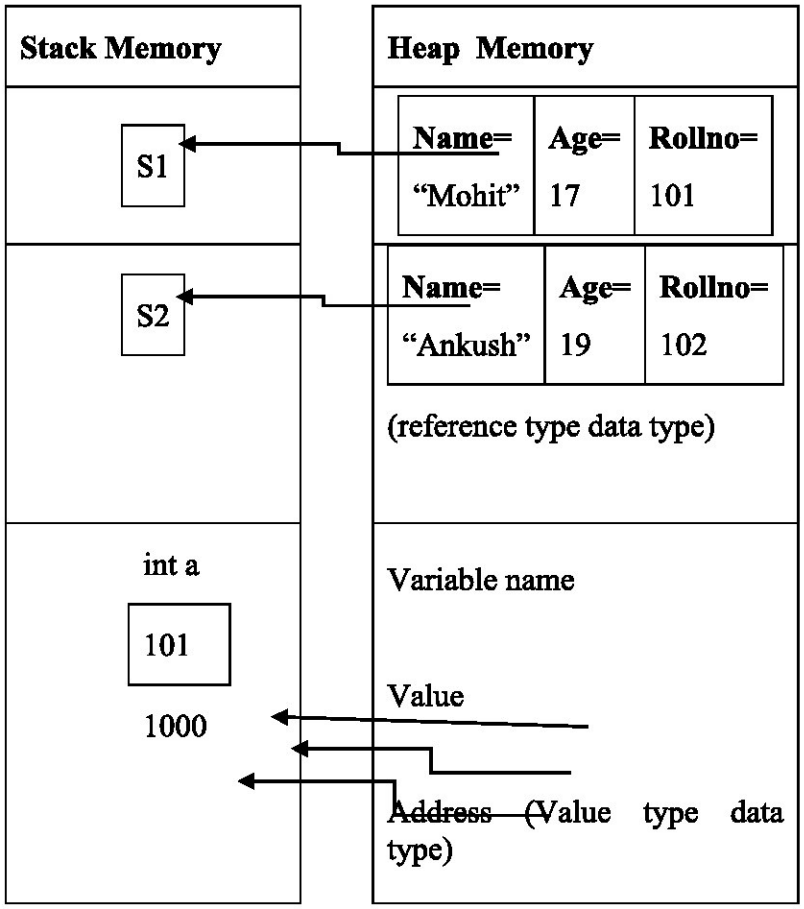


Figure 1.3

- Object consumes memory separately for each object.
- They are also known as physical concept of data.
- Objects can access public members of the class outside the class.
- Objects are called by reference in methods calling as they are created from class and class is the reference type data type.
- Reference of the object is stored in stack memory.

- Memory for values of object is reserved in heap memory when we allocate the memory with new keyword.
- Memory for object is automatically de-allocated by garbage collector when memory is of no-use in program.

Any object may call any public method with dot operator, which is declared in its class definition.

Check Your Progress

1. What is class? Explain its characteristics.
2. Explain the object in OOPs with its characteristics.

1.10 ABSTRACTION AND ENCAPSULATION

Abstraction :

Abstraction is the feature to hide the programming code to outside world and reveals its required information. Abstraction saves us form dealing with complicated and lengthy internal details. We just need to know the essential part for implementation.

For example, if we want to implement the banking system, then customer only needs to know about deposit and withdrawal forms, rest internal details are of no use for him. Internal processing will be there but the customer only need to know the interfacing with forms. In the same way, in programming if we are implementing any problem, then we only need to know how to implement the pre solved things.

In simpler way, we can say abstraction hides the complicated details and reveals the essential in formation only.

Encapsulation :

Encapsulation is the feature to wrap up the data members and methods into a single unit. Wrapping the members in a single unit restrict it to be accessed directly outside that unit. That unit is known as class. Encapsulation restricts the free flow of the data across the program. It also ensures that outside data to the class may also not be accessed inside the class. So we may also say that encapsulation creates a layer between inside class members and outside class members. Whenever we create the object of that class, that object may access only those members. It means, data of one object will be different from data of other object. One object cannot update the data of other object directly. If we need to update the data of one object, we must have one method for that in the class.

Check Your Progress

1. What is abstraction?
2. Explain encapsulation.

1.11 INHERITANCE

Inheritance is the ability to reuse the members of one class into other class without redefining them. The class which is to be accessed is called super class or base class. The class in which members will be accessed is known as sub class or derived class. We may inherit more than one class into any class.

1.11.1 TYPES OF INHERITANCE

Inheritance is of five types :

1) Single Inheritance
2) Multilevel Inheritance
3) Multiple Inheritance
4) Hierarchical Inheritance
5) Hybrid Inheritance

Java does not support multiple inheritance via class.

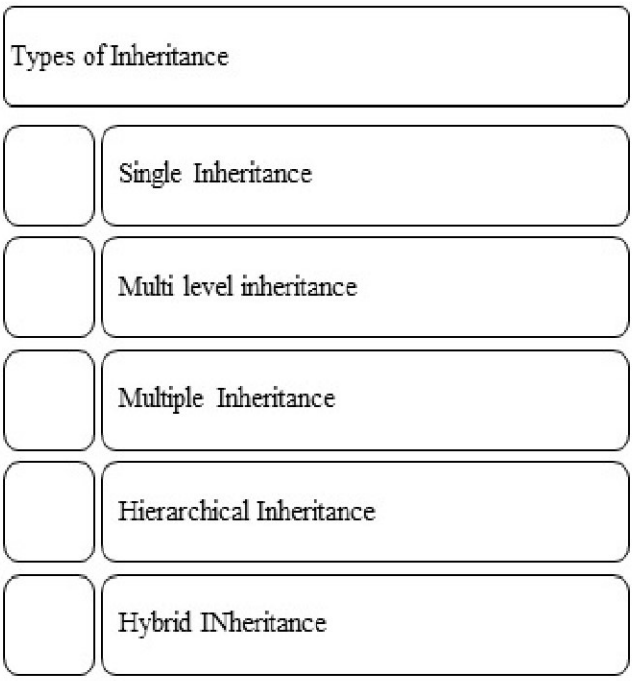


Figure 1.4

Check Your Progress

1. Explain the inheritance.
2. What are the various types of inheritances?

1.12 METHOD OVERRIDING AND POLYMORPHISM

Polymorphism : Polymorphism is the feature which enables any object to act differently in different scenarios. For example, there is a boy, at home- he is son, at school- he is student, at sports ground- he is sportsman but he is the same boy. He will act differently at different scenarios. Polymorphism is again very important feature of object oriented approach as we are more focused on real world problems which are dynamic in nature.

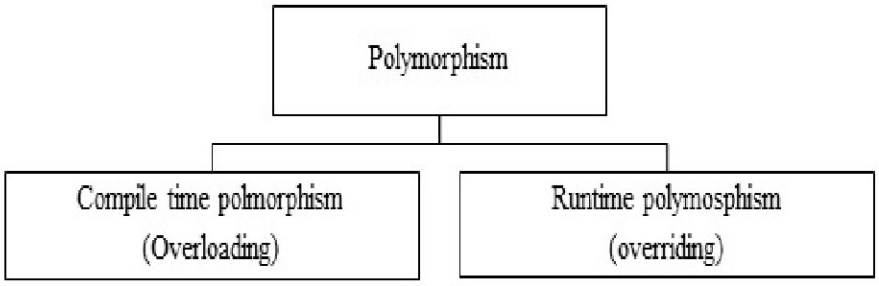


Figure 1.5

Compile time polymorphism is implemented via implemented via overloading. While runtime polymorphism is implemented via overriding.

Check Your Progress

1. Explain the types of inheritances.
2. Describe the polymorphism and its types.

1.13 SUMMARY

OOP's is one of the most popular approach these days for problem solutions. As we have discussed problems are very dynamic, complex and related to real world. So we need such approach. Class, objects, abstraction, encapsulation, inheritance, polymorphism are some of the most important features of OOP's. These work together to solve and complex problem. This approach also gives some flexibility, easiness, easy maintenance and many other benefits which has really made our development easy. Inheritance gives us the reusability feature on the other hand polymorphism gives us the many faces implementation at compile time and as well as runtime.

1.14 EXERCISE

Q. 1. What is OOPS? Explain the features of OOPs.

Q. 2. Compare the procedural oriented and object oriented programming approach.

Q. 3. What is inheritance? Describe its types.

Q. 4. What is Polymorphism? How it may be implemented?

Q. 5. Describe some applications of OOPs.

Q. 6. Explain the class and object along with abstraction.

UNIT-2 JAVA LANGUAGE BASICS

Structure :

2.1 OBJECTIVE

Objective of this is unit is to know the basics of java programming language. Data types in java, variables, all tokens like operators etc. will be discussed in this unit.

2.2 INTRODUCTION TO JAVA

Java is one of the Object oriented programming languages. Before understanding java. Let's first understand the types of applications:

2.2.1 TYPES OF APPLICATIONS

A. Console application

- Fixed Entry & Exit point.
- Command Line Interface (CLI) based.
- May be stand alone or network application.
- Generally, contains main function.
- Generally developed via procedure oriented programming languages.
- Supported languages are C, C++, Java, Basic, Fortran etc.
- Needs to be installed.

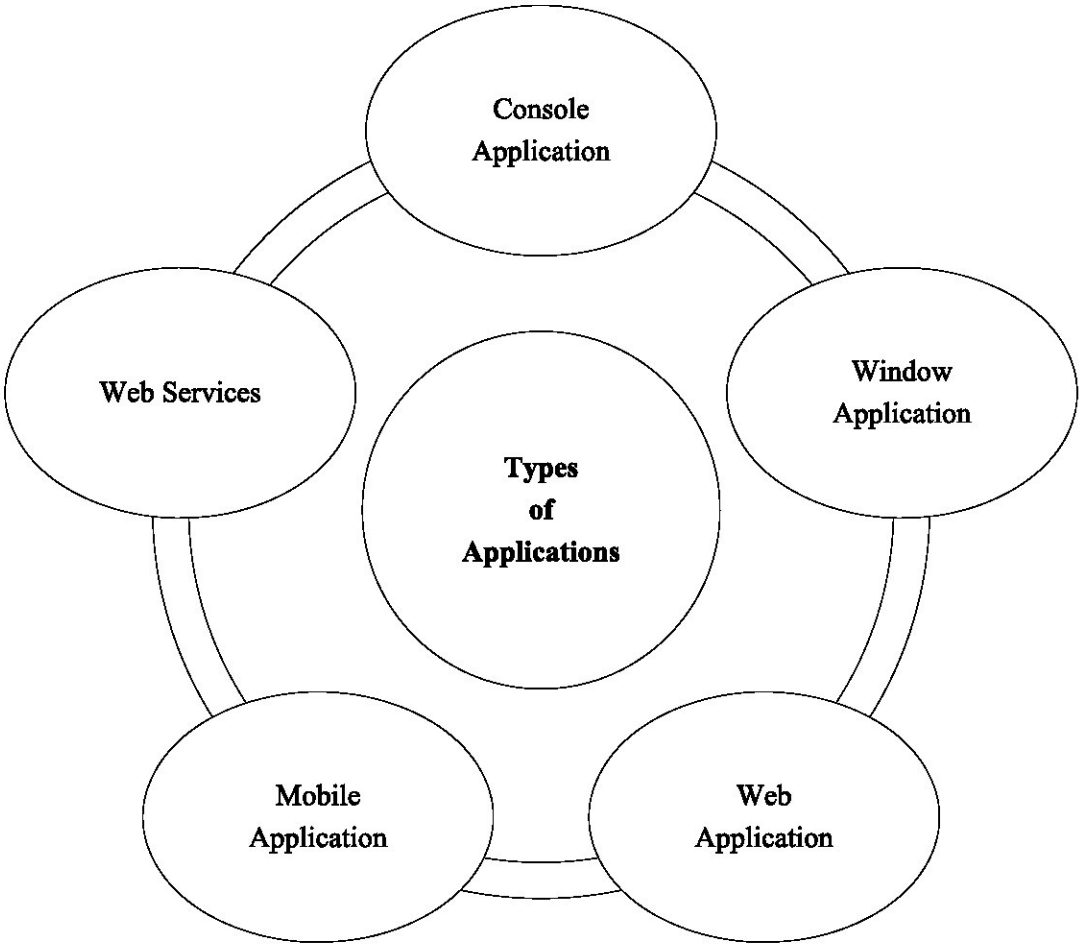


Figure 2.1

B. Window application

- No fixed entry or exit point.
- Graphical User Interface (GUI) based.
- May be standalone or network application.
- Developed via object oriented languages.
- Supported languages are .Net, Java, Visual Basic etc.
- Needs to be installed.

C. Web Application

- Web Server based application.
- Can't be installed, installed in server.
- No high hardware/software configuration required.
- Supported technologies are .Net, Java, PHP, CFML, etc.

D. Mobile Application

- Application for smart devices (devices other than PC).
- Programming constructs are approximately similar as previously used.
- Supported languages are Android, Java, .Net etc.

E. Web Services

Sharable modules via server.

- Sharable among different orgaizations.
- Sharable via different technologies.
- Supported technologies are .Net, Java etc.
- For example SMS gateway, payment gateway.

2.2.2 JAVA

Java was conceived by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan at Sun Microsystems, Inc. in 1991. It took 18 months to develop the first working version. This language was initially called "Oak," but was renamed "Java". Between the initial implementation of Oak in the fall of 1992 and the public announcement of Java in the spring of 1995, many more people contributed to the design and evolution of the language.

The Java is

- A programming language.
- A technology of Sun microsystem.
- A development environment.
- An application environment.
- Similar in syntax to C++; similar in semantics to SmallTalk.
- Used for developing applications, applets & servlets.

2.2.3 MAIN FEATURES OF JAVA

Simple: Java was designed to be easy for the professional programmer to learn and use effectively.

- **Object-Oriented :** The object model in Java is simple and easy to extend, while primitive types, such as integers, are kept as high-performance non objects.

- **Robust :** The multi-platform environment of the Web places extraordinary demands on a program, because the program must execute reliably in a variety of systems. Thus, the ability to create robust programs was given a high priority in the design of Java. To gain reliability, Java restricts you in a few key areas to force you to find your mistakes early in program development.

- **Multithreaded :** Java was designed to meet the real-world requirement of creating interactive, networked programs. To accomplish this, Java supports multithreaded programming, which allows you to write programs that do many things simultaneously.

- **Architecture-Neutral :** A central issue for the Java designers was that of code longevity and portability. One of the main problems facing programmers is that no guarantee exists that if you write a program today, it will run tomorrow—even on the same machine. Operating system upgrades, processor upgrades, and changes in core system resources can all combine to make a program malfunction. The Java designers made several hard decisions in the Java language and the Java Virtual Machine in an attempt to alter this situation. Their goal was "write once; run anywhere, anytime, forever." To a great extent, this goal was accomplished.

- **Interpreted and High Performance :** As described earlier, Java enables the creation of cross-platform programs by compiling into an intermediate representation called Java byte code. This code can be executed on any system that implements the Java Virtual Machine. Most previous attempts at cross-platform solutions have done so at the expense of performance.

- **Distributed :** Java is designed for the distributed environment of the Internet because it handles TCP/IP protocols. In fact, accessing a resource using a URL is not much different from accessing a file. Java also supports Remote Method Invocation (RMI). This feature enables a program to invoke methods across a network.

- **Dynamic :** Java programs carry with them substantial amounts of run-time type information that is used to verify and resolve accesses to objects at run time. This makes it possible to dynamically link code in a safe and expedient manner. This is crucial to the robustness of the Java environment, in which small fragments of byte code may be dynamically updated on a running system.

2.2.4 TYPES OF JAVA CODES

- **Application Code** (Installed on personal computer & executes there)

- **Applet Code** (remotely accessed)

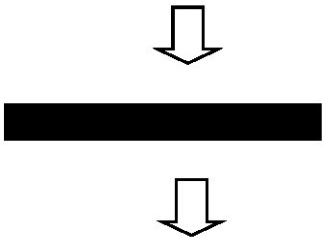
- **Servlet Code** (remotely processed)

2.2.5 JAVA EXECUTION CYCLE

Java Program is executed in two phases. First phase is compilation and second phase is execution phase.

Phase -1 (Compilation)

Source Code (.java file)

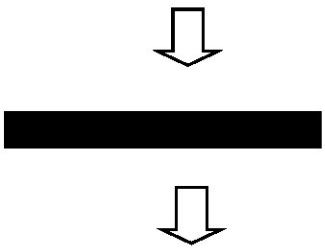


Byte Code (.class file)

Figure 2.2

Phase -2 (Execution)

Byte Code (.class file)



Result

Figure 2.3

Phase -1 (Compilation)

This phase will need source code file of java saved with .java extension. For compilation we need java compiler installed on our machine. That compiler will translate the code into byte code. Byte code is translated code but not machine dependent code like object code.

Phase -2 (Execution)

This phase needs java interpreter. Interpreter read byte code file into main memory and gets its executed by CPU. This process does not require OS for loading process into a new memory partition because that byte code is being loaded with the help of java interpreter.

2.2.6 JAVA AS PLATFORM INDEPENDENT TECHNOLOGY

Platform dependency :

In case of 'C' language compilation, our source code is translated into object code and then it is linked with library and operating system information is attached to it. OS information is essential to attach otherwise this file may not be loaded into

main memory by OS for execution. But as OS information is attached, so this executable file is platform dependent. It may not be executed on other platform. For execution on other program, we need a different executable file. It means we need separate executable files for different platforms. This is called platform dependency. That's why we need separate software setups for different platforms. For example, separate software setup of any game for windows, Linux, android etc.

Given is the 'C' program execution cycle which shows the platform dependency:

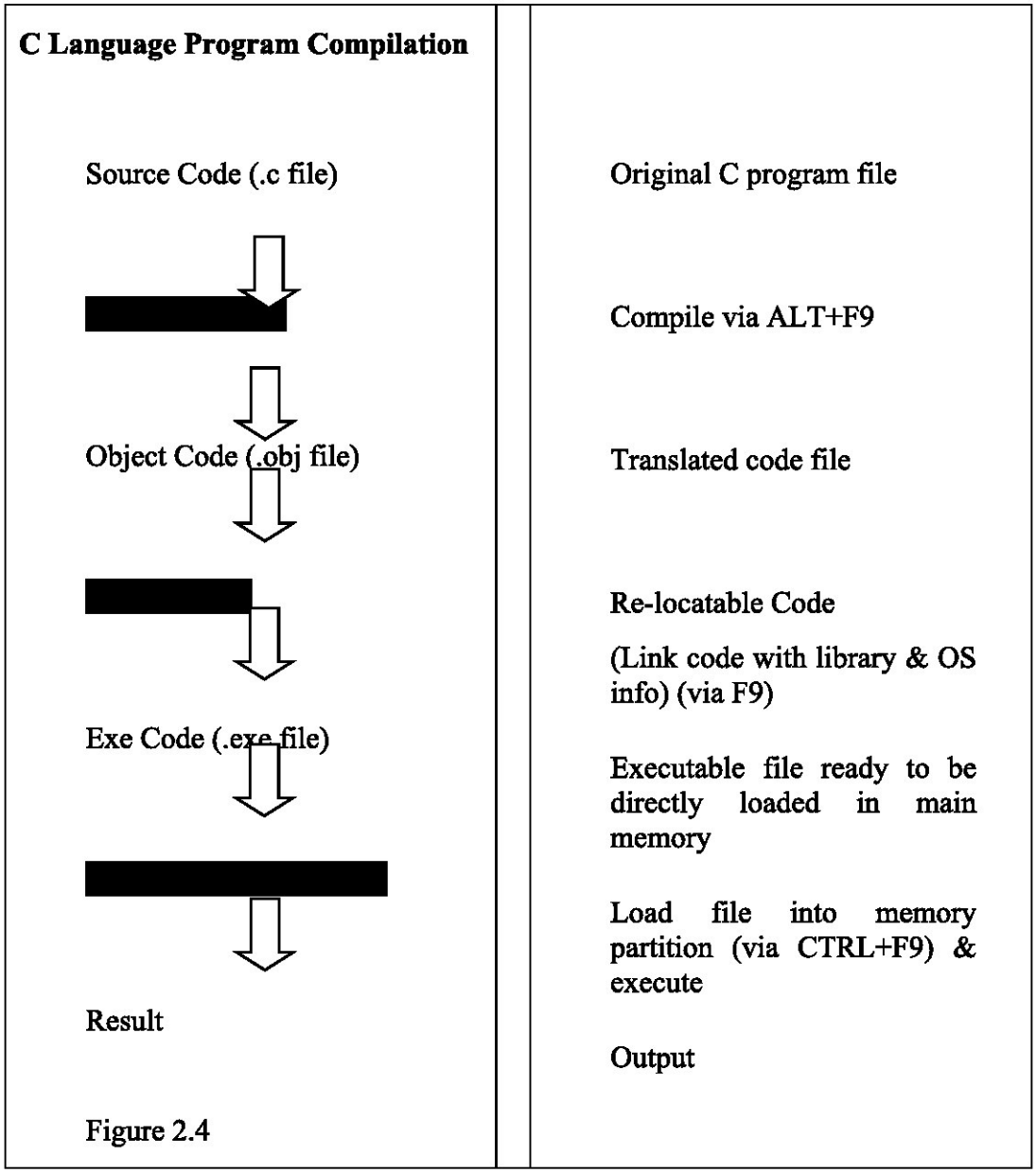


Figure 2.4

Byte code (platform independency code) :

Platform independency is required so that we may execute one file on multiple platforms. This may be achieved with the concept of byte code in java.

Byte code is a form of translated code that is neither binary nor java. It is the special form that may be translated for a special machine called Java Virtual Machine (JVM). JVM is a virtual machine because it is software but works as hardware. It works like a hardware because java compiler generates code to be executed by JVM as in case of 'C' language executable code was generated for CPU to be executed.

But in reality, JVM does not execute code directly, all the execution is done only by CPU but it reads the code into main memory and with the help of JIT (Just In Time), it translates the required code into binary and forward it to CPU for execution. Byte code does not require platform information association with it because that code is already been loaded into main memory and JVM is already in main memory for co-ordination with CPU. Byte code of java is stored in file with .class extension. That class file may be executed on any platform like windows, Linux, mac etc. but JVM must be installed on that platform.

Finally we can conclude that Java Byte code is platform independent but JVM is platform dependent. It is the JVM which makes the java platform independent technology.

Let's again see 2 phase java code compilation cycle:

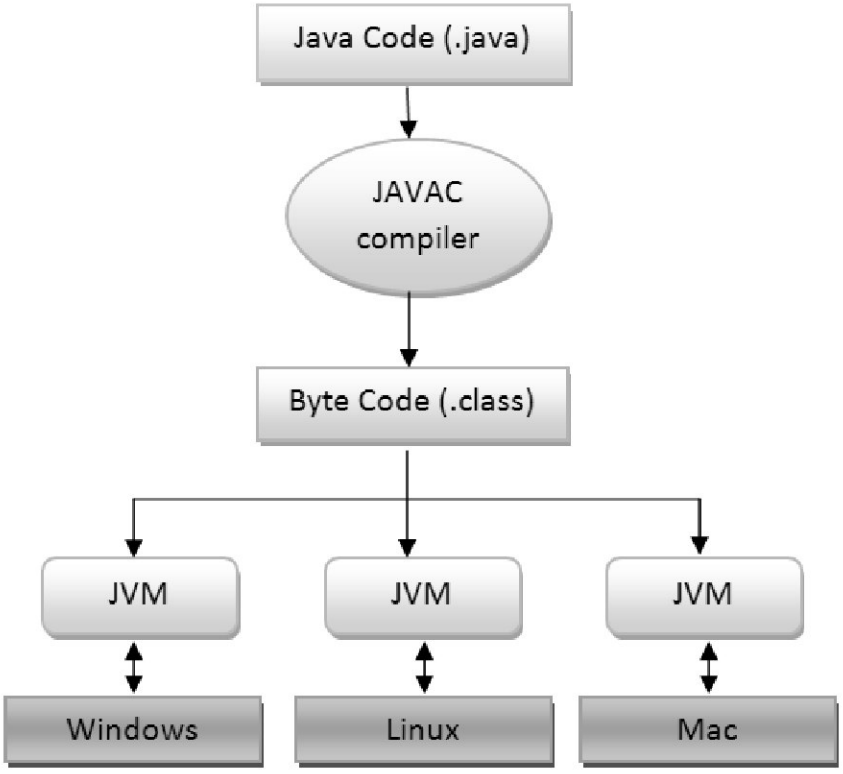


Figure 2.5

JIT Compiler :

The just-in-time compiler comes with the virtual machine. In the Java programming, a just-in-time (JIT) compiler is a program that turns byte code into instructions that can be sent directly to the processor. After writing a Java program, the source language statements are compiled by the Java compiler into

byte code rather than executable code (for an Intel Pentium microprocessor or an IBM System/390 processor). The byte code is platform-independent code that can be sent to any platform and run on that platform. Using the Java just-in-time compiler at the particular system platform compiles the byte code into the particular system code. Once the code has been (re-)compiled by the JIT, it will usually run more quickly in the computer.

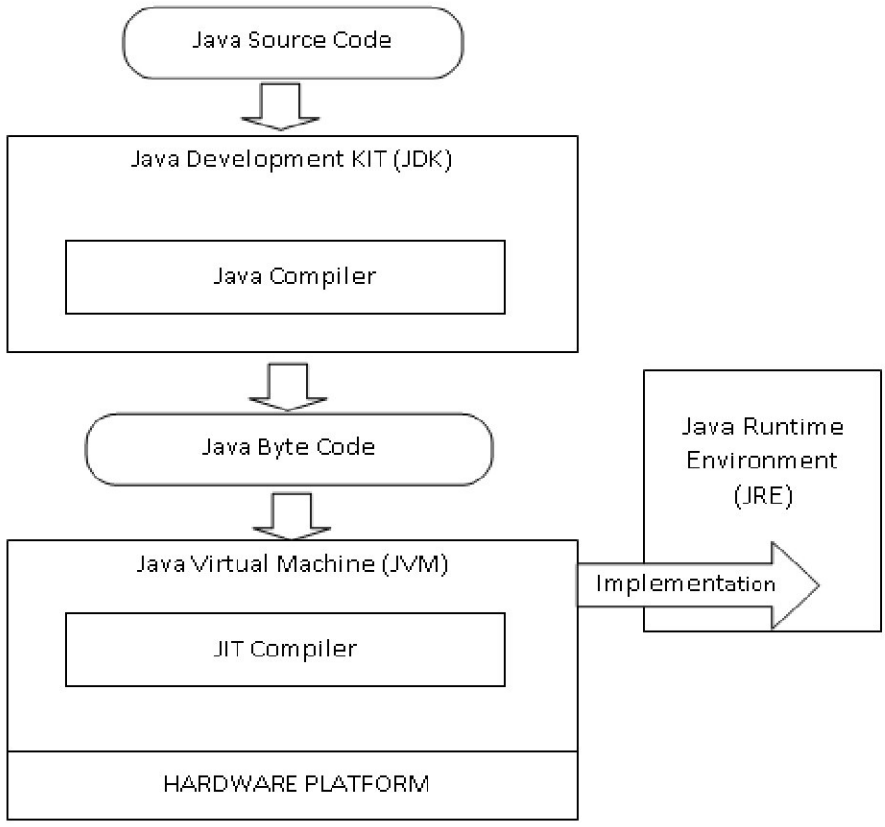


Figure 2.6

2.2.7 CORE PART OF JAVA TECHNOLOGY

- **JVM** is the Java platform component that executes programs.
- **JRE** is the on-disk part of Java that creates the JVM.
- **JDK** allows developers to create Java programs that can be executed and run by the JVM and JRE.
- **JDK:** The JDK is a key platform component for building Java applications. At its heart is the Java compiler. The Java Development Kit (JDK) is a software development environment used for developing Java applications and applets. It includes the Java Runtime Environment (JRE), an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc) and other tools needed in Java development.

Here is the list of java compilers-

COMPILERS OF JAVA		
Version	**Code Name**	**Release**
Beta Java	BETA Version	1995
JDK 1.0	Oak	23-Jan-96
JDK 1.1	Sparkler	19-Feb-97
JDK 1.2	Playground	8-Dec-98
JDK 1.3	Kestrel	8-May-00
JDK 1.4	Merlin	6-Feb-02
JDK 1.5	Tiger	30-Sep-04
JDK 1.6	Mustang	11-Dec-06
JDK 1.7	Dolphin	7-Jul-11
JDK 1.8	Spider	18-Mar-14
JDK 1.9		21-Sep-17
JDK 1.10		20-Mar-2018
JDK 1.11		25-Sep-2018
JDK 1.12		19-Mar-2019
JDK 1.13		10-Sep-2019

Table 2.1

- **JRE:** JRE stands for "Java Runtime Environment". JRE is an installation package which provides environment to only run (not develop) the java program (or application) onto machine. JRE is only used by them who only wants to run the Java Programs. The Java Runtime Environment provides the minimum requirements for executing a Java application. JRE consists of the following components:
- Java Virtual Machine (JVM)
- Deployment technologies

- User interface toolkits
- Integration libraries
- Other base libraries
- Lang and util base libraries

How JRE works:

To understand how the JRE works let us consider a Java source file saved as *.java*. The file is compiled into a set of Byte Code that is stored in a "*.class*" file.

The following actions occur at runtime.

Class Loader: The Class Loader loads all necessary classes needed for the execution of a program. It provides security by separating the namespaces of the local file system from that imported through the network. These files are loaded either from a hard disk, a network or from other sources.

Byte Code Verifier: The JVM puts the code through the Byte Code Verifier that checks the format and checks for an illegal code. Illegal code, for example, is code that violates access rights on objects or violates the implementation of pointers.

The Byte Code verifier ensures that the code adheres to the JVM specification and does not violate system integrity.

Intrepreter: At runtime the Byte Code is loaded, checked and run by the interpreter. The interpreter has the following two functions:

- Execute the Byte Code
- Make appropriate calls to the underlying hardware

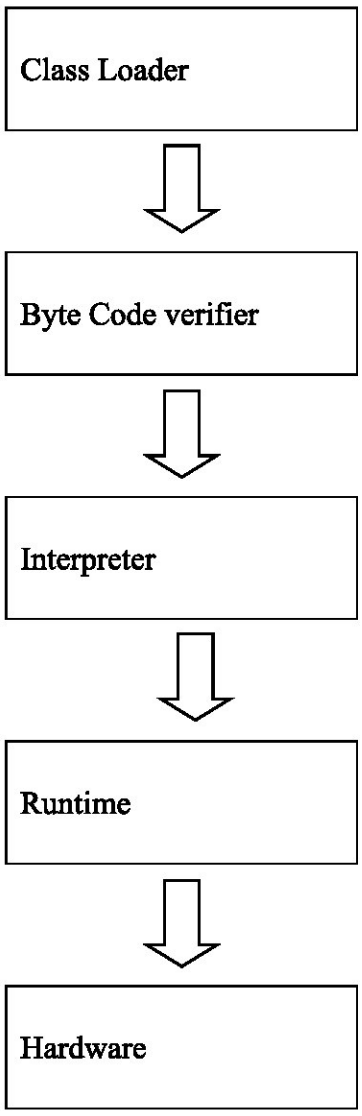


Figure 2.7

- **JVM:** JVM stands for Java Virtual Machine. JVM acts as a run-time engine to run Java applications. JVM is the one that actually calls the main method present in a java code. JVM is a part of JRE. Java applications are called WORA (Write Once Run Anywhere). This means a programmer can develop Java code on one system and can expect it to run on any other

Java enabled system without any adjustment. This is all possible because of JVM. When we compile a .java file, .class files (contains byte-code) with the same class names present in .java file are generated by the Java compiler. This .class file goes into various steps when we run it. These steps together describe the whole JVM.

2.2.8 JAVA IDE

IDE stands for Integrated Development Environment. It allows developer to create, design, analyse, compile, deploy and test etc. their programs. Not all IDE provides all services. We have to have IDE as per our requirement.

There is a list of some of the IDEs for java development-

IDE OF JAVA			
NetBeans	Eclipse	jCreator	BlueJ
IntelliJ	jEdit	jGRASP	JSource
JDeveloper	DrJava	Enide Studio	Android Studio

Table 2.2

Check your progress

1. Make a comparison between JRE and JDK.
2. How java is platform independent?
3. Explain the features of java.
4. Describe the execution cycle of java.

2.3 JAVA TOKENS

A *token* is the smallest unit of information in program. It is the basic building block of program. Any program is made up of tokens only.

We may also understand the token as most important part of any program. Token is the smallest meaningful unit in the program. Tokens are used for different purposes at different places.

For example, if we think of any house, it is made up of different elements like cement, iron, wood, bricks, doors, windows, electric elements, bath elements etc. But builder has to decide where to put which element. In the same way token is the basic element of any program. One program is made up of many tokens. Every token is used for any particular purpose. Some tokens have predefined purposes and some tokens are defined by programmer. As a programmer we have

to decide, where to use which token. We may also define new tokens as per our need. Simply we may say that, anything we write down in program is token.

For example: if, for, while, switch, class, all variables name, all operators are tokens.

There are 5 types of tokens in the program-

1. **Keyword:** A keyword are the reserve words in compiler, which have predefined and fixed meaning.
2. **Identifiers:** identifiers are the names given by programmer or library for any specific purpose. e. g. variable name, object name, class name, method name, array name etc.
3. **Operators:** An Operator is a symbol which takes any action on operand(s). there are 3 types of operators:
   a. *Unary operator*: Takes action on one operand e.g. increment and decrement operator.
   b. *Binary operator*: Takes action on two operands e.g. plus, minus, greater than etc.
   c. *Ternary operator*: Takes action on three operands. There is only 1 ternary operator i.e. ? :

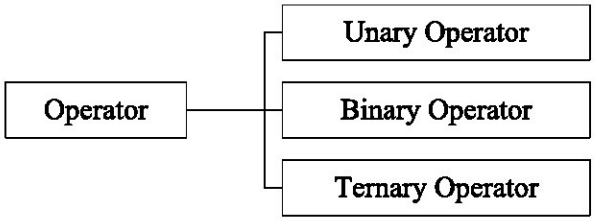


Figure 2.8

4. **Literals:** Literals are the values used in source codes. For example numeric and strings used in source code.
   a. *Numeric literals:* Integer literal (whole numbers & real Literal (numbers with decimal precisions)
   b. *Non numeric literal:* Character literal(enclosed in ‘’), String Literal(enclosed in “”), Boolean literal (true/false)

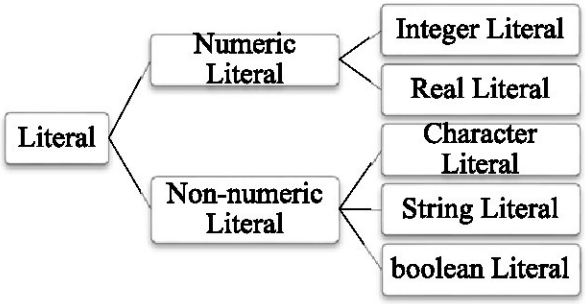


Figure 2.9

5. **Punctuation marks:** Punctuation marks are special symbols used in program. For example space, tab, enter, comma, braces etc.

Check your progress

1. What is token? Explain its types.
2. Explain the naming conventions for identifiers.
3. Make a list of Java Tokens.
4. What is operator? Explain its types.

2.4 PRIMITIVE DATA TYPE

Data type is the collection of values which can be assigned to the variable. Data type is mandatory to be declared before the use of any variable because it determines the type of operations those can be performed on the declared variables. Suppose arithmetic operation can be performed on the numeric values only and string operation can be performed on the char data type only. Every data type has its fixed memory consumption at compile time. The range of values is also there for the numeric values.

Data type can also be treated as the type of the variable. A data type in programming, a classification identifying one of various types of data, as floating-point, or integer, stating the possible values for that type, the operations that can be done on that type, and the way the values of that type are stored.

There are two types of data types:

Primitive Data Type (Basic Data Type)

Primitive data type is the data type which may be used in program directly. They are also known as basic data type. Memory required for such data types is fixed by compiler in advance. Important points for primitive types are-

- Variable contains the value assigned to them. That's why such types as also known as value type data type.
- The size of such variable can't be changed.
- Memory allocation for such variables are also implicit.
- Memory is allocated in stack memory of main memory.
- It supports static memory allocation because Memory allocation is compile time.
- They are available in compiler so they may be used directly in program without any reference.

Primitive Data Types (Value Type data type)		User Defined data types (Reference Type data type)
byte	(1 byte)	class
short	(2 bytes)	Interface
int	(4 bytes)	Array
long	(8 bytes)	
float	(4 bytes)	
double	(8 bytes)	
char	(2 bytes)	
bool	(1 bytes)	

Table 2.3

User Defined Data Type (Reference type data type)

User defined data types are not available in compiler. They have to be first declared or imported before use. Variable name of this type contains the address of the location where the values are stored. Values are stored at different location of heap memory while addresses are stored in stack memory. They are declared in two phases. First phase is declaration and second phase is memory allocation.

Important points for primitive types are-

- Variable contains the address of location which contains value. That's why such types as also known as reference type data type.
- The size of such variable can be changed.
- Memory allocation for such variables are also explicit.
- Memory for address is in stack memory of main memory while for value memory is allocated in heap memory.
- It supports dynamic memory allocation because Memory allocation is run time.
- They are not available in compiler so they must be imported or declared before use.

Check Your Progress

1. What is data type?
2. Compare value type and reference type data types.

2.5 VARIABLES

Variable is the named memory location which stores the value temporarily during program execution. There are 3 types of variables in java-

A. **Instance variables**- These are the members of a class, and created for each object of the class. Instance variables exist as long as the object they belong to is in use at runtime. Values for each instance variable may be different for each object.

B. **Static variables**- These are the members of a class, but not created for any specific object of the class and, therefore, belong only to the class. They are created when the class is loaded at runtime, and exist as long as the class is available at runtime.

C. **Local variables**- They are declared in methods, constructors, and blocks; and created for each execution of the method, constructor, or block. After the execution of the method, constructor, or block completes, local (non-final) variables are no longer accessible.

Scope / Life of Variables		
Instance variables	**Static variables** (class variable)	**Local variables**
They are members of a class, and created for each object of the class separately. The values of these variables at any given time constitute the state of the object. Instance variables exist as long as the object they belong to is in use at runtime.	They are also members of a class, but not created for any specific object of the class and, therefore, belong only to the class. They are created when the class is loaded at runtime, and exist as long as the class is available at runtime.	They are declared in methods, constructors, and blocks; and created for each execution of the method, constructor, or block. After the execution of the method, constructor, or block completes, local variables are no longer accessible.

Table 2.4

Check Your Progress

1. What is variable?
2. Describe the types of variables.
3. What are the rules for naming java variables?

2.6 JAVA OPERATORS

Operators are symbols which take one or more operands or expressions and perform any arithmetic or logical computations. The values on which operator

perform any action are known as operands. E.g. in a+b: + is the operator and a, b are the operands.

There are 3 types of operators.

- **Unary Operator:** Unary operators are those operators which perform action on only one operand. There are some unary operators e.g. sizeof( ), negation operator, increment, decrement operator etc.

- **Binary operator:** Binary operators are those operators which perform action on exactly two operands. Mostly operators are the binary operators e.g. +, -, *, >, = etc.

- **Ternary Operator:** Ternary operators are those operators which perform action on three operands. There is only one ternary operator i.e. ternary operator (? :).

Operators are categorized in the following categories:

- **Arithmetic Operators :** Java provides five basic arithmetic operators. These are summarized in Table.

Operator	Name	Example	
+	Addition	12 + 4.9	// gives 16.9
-	Subtraction	3.98 – 4	// gives -0.02
*	Multiplication	2 * 3.4	// gives 6.8
/	Division	9 / 2.0	// gives 4.5
%	Remainder	13 % 3	// gives 1

Table 2.5

- ***Relational Operators***

Java provides six relational operators for comparing numeric quantities. Relational operators evaluate to 1 (representing the true outcome) or 0 (representing the false outcome).

Operator	Name	Example
==	Equality	5 == 5 // gives 1

!=	Inequality	5 != 5 // gives 0
<	Less Than	5 < 5.5 // gives 1
<=	Less Than or Equal	5 <= 5 // gives 1
>	Greater Than	5 > 5.5 // gives 0
>=	Greater Than or Equal	6.3 >= 5 // gives 1

Table 2.6

- ***Logical Operators***

Java provides three logical operators for combining logical expression. These are summarized in Table. Like the relational operators, logical operators evaluate to 1 or 0.

Operator	Name	Example
!	Logical Negation	!(5 == 5) // gives 0
&&	Logical And	5 < 6 && 6 < 6 // gives 1
||	Logical Or	5 < 6 || 6 < 5 // gives 1

Table 2.7

- ***Bitwise Operators***

Java provides six bitwise operators for manipulating the individual bits in an integer quantity. These are summarized in Table.

Operator	Name	Example
~	Bitwise Negation	~'\011' // gives '\366'
&	Bitwise And	'\011' & '\027' // gives '\001'

|	Bitwise Or	'\011' | '\027' // gives '\037'
^	Bitwise Exclusive Or	'\011' ^ '\027' // gives '\036'
<<	Bitwise Left Shift	'\011' << 2 // gives '\044'
>>	Bitwise Right Shift	'\011' >> 2 // gives '\002'

Table 2.8

- ***Increment/Decrement Operators***

The auto increment (++) and auto decrement (--) operators provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable. The examples assume:

```
int     k = 5;
```

Operator	Name	Example
++	Auto Increment (prefix)	++k + 10 // gives 16
++	Auto Increment (postfix)	k++ + 10 // gives 15
--	Auto Decrement (prefix)	--k + 10 // gives 14
--	Auto Decrement (postfix)	k-- + 10 // gives 15

Table 2.9

- ***Assignment Operator***

The assignment operator is used for assigning a value at some memory location. Its left operand is an lValue, and its right operand is an rValue. rValue is assigned into lValue.

Assignment may also be used in shorthand form like-

Operator	Example	Equivalent To
=	n = 25	n = 25
+=	n += 25	n = n + 25
-=	n -= 25	n = n – 25
*=	n *= 25	n = n * 25

/=	n /= 25	n = n / 25
%=	n %= 25	n = n % 25
&=	n &= 0xF2F2	n = n & 0xF2F2
|=	n |= 0xF2F2	n = n | 0xF2F2
^=	n ^= 0xF2F2	n = n ^ 0xF2F2
<<=	n <<= 4	n = n << 4
>>=	n >>= 4	n = n >> 4

Table 2.10

- ***Conditional Operator***

The conditional operator takes three operands. It has the general form:

operand1 ? operand2 : operand3

First operand1 is evaluated, which is treated as a logical condition. If the result is nonzero then operand2 is evaluated and its value is the final result. Otherwise, operand3 is evaluated and its value is the final result. For example:

```
m = 1; n = 2;
min = (m < n ? m : n);          // min receives 1
```

Note that of the second and the third operands of the conditional operator only one is evaluated. This may be significant when one or both contain side-effects (i.e., their evaluation causes a change to the value of a variable). For example, in

```
min = (m < n ? m++ : n++);
```

m is incremented because m++ is evaluated but n is not incremented because n++ is not evaluated.

Because a conditional operation is itself an expression, it may be used as an operand of another conditional operation, that is, conditional expressions may be nested. For example:

```
int m = 1, n = 2, p =3;
int min = (m < n ? (m < p ? m : p): (n < p ? n : p));
```

Check Your Progress

1. What is java operator?
2. Explain the types of java operators.
3. What are bitwise operators? Explain with example.

2.7 SUMMARY

Java is one of the most popular OO programming language. C++ programmer finds it easy, as per syntax of basic programming constructs are concern. Programming basics which are available in C++ are mostly available here. Tokens are the basic building block of any java program.

Data types in java are categorised in 2 categories on the basis of what they stored. If they stored values, they are known as value type data type; on the other hand if they stored reference to the values, they are known as reference type data type.

Operators in Java are quite similar to C++. They are typed into 3 parts- unary, binary and ternary. They are categorised on the basis of number of operands, they use to take any action. It may be one, two or three.

If we talk about variables, they are used to hold the values temporarily but they are categorised into three parts- local, instance and static variables. Variables may be declared in class or in any method. Inside the class, they may be declared in two reference- class or object. There uses make the type of variable they are.

2.8 EXERCISE

Q. 1. What is data type in Java? Explain with detail.

Q. 2. Explain the operators in java with their types.

Q. 3. What is variable? Describe their types.

Q. 4. Compare the value type and reference type data types in java.

Q. 5. What is java token? Describe types of tokens.

UNIT-3 EXPRESSIONS, STATEMENTS AND ARRAYS

Structure :

3.1 OBJECTIVE

This unit will cover Statements, expression, control statement, types of control statement like selection, iteration and sequential will be discussed. Array will also be covered in this unit to complete the basic implementation of java programming besides the previous heads.

3.2 INTRODUCTION

Java is one of the widely used Object Oriented Programming Language. But it also supports several mathematical calculations and other expressions for evaluation. There was a time when imperative paradigm was used in programming than came the structured paradigm.

Java supports the execution of sequential statements those are executed one after another. Besides that java also support structured programming statements. Structured programming was that programming which supports non-sequential execution of statement but in structured way. They support sequential statements, selection statements, and iterative statements. In this unit we will discuss expression used in java, selection statements and iterative statements.

3.3 EXPRESSION

Expression is the collection of variables, literals, operators or method calls. Expression evaluates to a single value. There may be simple expression or nested expression. Expression may be used to assign a variable or to calculate something.

For example-

```
int a=10;
int b= 10+20;
int c=a+10;
int d=2*a+(b-5);
float interest=p*r*t/100.0;
```

Type Compatibility

Type compatibility is applied when we use two different types of data types in one operation. Type compatibility allows the use of two different types in one operation without any notification and it also allows substituting one value for other without modification. It is very close to implicit type conversion. Type compatibility is applied in two forms: First at the assignment compatibility, second at the expression compatibility.

Example: Unit-3 (Program-01)

/* Program to show the implementation of expression capability and assignment capability */

```
class unit03p01
{
 public static void main(String[] args)
 {
  int  a, b;
  a=10/3;  // expression capability
// b=10.5;  // not possible in java
  b=10;
  System.out.println("A="+a);
  System.out.println("B="+b);
 }
```

```
}
```

Result:

```
C:\JavaP>javac unit03-1.java
C:\JavaP>java unit03-1
A=3
B=10
```

Value of a will be 3 because 10 and 3 are integers and their result will be integer in JAVA. It will be called expression compatibility. Value of b will be 10 because b is also declared as integer, it cannot store the float value. It is called assignment compatibility.

Type Casting

You can force an expression to be of a specific type by using a *cast*. The general form of a cast is

```
(type) expression
```

where *type* is a valid data type. For example, to cause the expression **x/2** to evaluate to type **float**, write

```
(float) x/2
```

Casts are technically operators. As an operator, a cast is unary and has the same precedence as any other unary operator. Casts can be very useful. Caste can be used for caste promotion or demotion both.

Type Conversion : Type conversion is the process in which one data type will be converted into another data type. When constants and variables of different types are mixed in an expression, they are all converted to the same type. The compiler converts all operands up to the type of the largest operand, which is called type promotion e.g.

Operand1 Type	Operand2 Type	Resultant Type
int	int	int
int	float	float
float	int	float
float	double	double
double	int	double
int	char	int

Operator Precedence

Operator Precedence determines which operator will be executed first in the expression or statement. Operator precedence is manipulated only if multiple

operators in single expression are found. The order in which operators are evaluated in an expression is significant and is determined by precedence rules. These rules divide the C operators into a number of precedence levels. Operators in higher levels take precedence over operators in lower levels.

Given is the operator precedence and association table.

Level	Operator	Description	Associativity
16	[] . ()	access array element access object member parentheses	left to right
15	++ --	unary post-increment unary post-decrement	not associative
14	++ -- + - ! ~	unary pre-increment unary pre-decrement unary plus unary minus unary logical NOT unary bitwise NOT	right to left
13	() new	cast object creation	right to left
12	* / %	multiplicative	left to right
11	+ - +	additive string concatenation	left to right
10	<< >> >>>	shift	left to right
9	< <= > >= instanceof	relational	not associative
8	== !=	equality	left to right
7	&	bitwise AND	left to right

6	^	bitwise XOR	left to right
5	|	bitwise OR	left to right
4	&&	logical AND	left to right
3	||	logical OR	left to right
2	?:	ternary	right to left
1	= += -= *= /= %= &= ^= |= <<= >>= >>>=	assignment	right to left

Table 3.1

Operator Association

If multiple operators of equal precedence are found in the single expression, operator association determines in which sequence the operators will be executed. Association can be either left to right or right to left. Mostly operators are associated from left to right.

When two operators of the same priority are found in the expression, precedence is given to the extreme left operator.

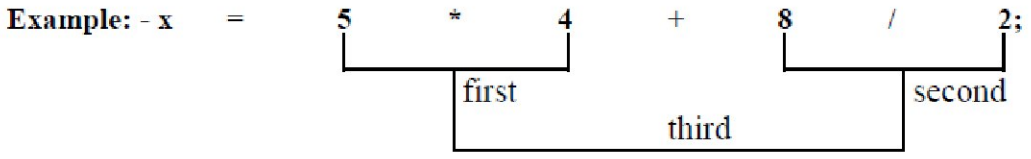


Figure 3.1

Here, 5*4 is solved first. Through * and / have the same priorities. The operator * occurs before /.

Check Your Progress

1. What is operator precedence and operator association? Explain in detail.
2. What is type conversion and
3. What is the result of the what is type casting? Compare them.
4. What is the result of the following code fragment?

```
int x = 5;
int y = 10;
 z = ++x * y--;
System.out.println("1 + 2 = " + 1 + 2);
                System.out.println("1 + 2 = " + (1 + 2));
```

3.4 STATEMENTS

Statements are roughly equivalent to sentences in natural languages. A java statement forms a complete unit of execution. The following types of expressions can be made into a statement by terminating the expression with a semicolon (;). For example-

- Assignment expressions
- Any use of ++ or --
- Method invocations
- Object creation expressions
- Control statement (if, for, while etc.)

A block is a group of zero or more statements between balanced braces and can be used anywhere a single statement is allowed. There may be zero or multiple statements in one block. Blocks may be nested upto any level.

Check Your Progress

1. What is statement? Compare statement with expression.
2. Explain the types of statements in java.

3.5 CONTROL STATEMENTS

A running program spends all of its time executing statements. The order in which statements are executed is called flow control (or control structures). This term reflect the fact that the currently executing statement has the control of the CPU, which when completed will be handed over (flow) to another statement. Flow control in a program is typically sequential, from one statement to the next, but may be diverted to other paths by branch statements. Flow control is an important consideration because it determines what is executed during a run and what is not, therefore affecting the overall outcome of the program.

There are 3 categories of control structures in JAVA

- Sequential Statement

- Selection Statement
- Iterative Statement

Sequential Statement

A statement, the smallest independent computational unit, specifies an action to be performed. In most cases, statements are executed in sequence. Sequential statements are those statements which are executed in linear order. These statement never change their flow of execution. Sequential statements may be used for declaration, method call or any calculation purpose.

3.6 SELECTION STATEMENTS

Selection statement applies the decision making and case control instruction in Programming. Decision and Case control instructions allow the computer to take a decision as to which instruction is to be executed next. Selection statement is used in structured programming to branch the execution of flow at run time. The branching will be done on the basis of any criteria given in the problem. This category involves if statement and switch statement. If statement is the branching statement which transfer the flow of control in the program.

Switch is the selective statement which select any case block on the basis of given value or variable. Switch can be used where programmer has multiple options but only one option can be selected by user. In selective statement specific block of code will be executed on the basis of criteria but only once. Suppose programmer needs to check even/odd number, +ve/-ve number, Teenagers, leap year, conditional discount, Grade of student etc.; they need selective statement. JAVA language must be able to perform different sets of actions depending on the circumstances. In sequence control structure, the various steps are executed sequentially, i.e. in the same order in which they appear in the program. In fact to execute the instructions sequentially, we don't have to do anything at all. By default the instructions in a program are executed sequentially. However, in serious programming situations, seldom do we want the instructions to be executed sequentially. Many a times, we want a set of instructions to be executed in one situation, and an entirely different set of instructions to be executed in another situation. This kind of situation is dealt using a decision control instruction.

A decision control instruction can be implemented using:

- The **if** statement
- The **if-else** statement
- The **if-else ladder** statement
- The **nested if-else** statement
- Switch Statement

if statement

It is sometimes desirable to make the execution of a statement dependent upon a condition being satisfied. The if statement provides a way of expressing this. JAVA uses the keyword **if** to implement the decision control instruction. IF the branching statement which directs the flow of execution at run time.

Syntax of if statement is-

```
Syntax:
:
if
(condition)
{
    :
}
:
```

JAVA uses the keyword "*if*" to execute a set of command lines or one command line when the logical condition is true. It has only one option. The set of command lines or command lines are executed only when the logical condition is true. The statement is executed only when the condition is true. In case condition is false the compiler skips the lines within the if block.

The keyword **if** tells the compiler that what follows is a decision control instruction. The condition following the keyword **if** is always enclosed within a pair of parentheses. If the condition, whatever it is, is true, then the statement is executed. If the condition is not true then the statement is not executed; instead the program skips past it. But how do we express the condition itself? And how do we evaluate its truth or falsity? As a general rule, we express a condition using 'relational' operators. The relational operators allow us to compare two values to see whether they are equal to each other, unequal, or whether one is greater than the other.

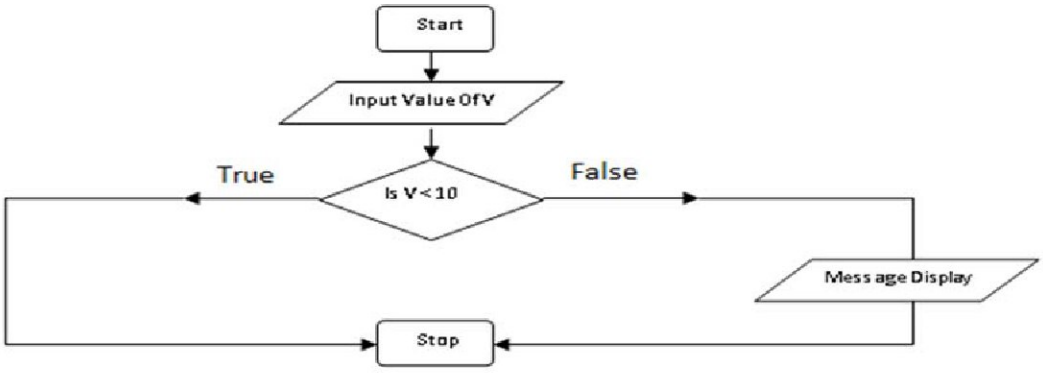


Figure 3.2

Example: Unit-3 (Program-02)

/* Write a program in java to convert the number into even number with its next even number if it is not even number.*/

```
class unit03p02
{
 public static void main(String[] args)
 {
        int a;
        a=9;
        if(a%2==1)    // check weather number is odd
        {
                a++;     // convert number to next even number
        }
        System.out.println("Even Number="+a);
 }
}
```

Result:

Even Number is= 10

Observe that here the two statements to be executed on satisfaction of the condition have been enclosed within a pair of braces. If a pair of braces is not used then the compiler assumes that the programmer wants only the immediately next statement after the **if** to be executed on satisfaction of the condition. In other words we can say that the default scope of the **if** statement is the immediately next statement after it.

The *if-else* Statement

The **if** statement by itself will execute a single statement, or a group of statements, when the expression following **if** evaluates to true. It does nothing when the expression evaluates to false. Programmer can use the if-else statement to specify what to execute if the criteria is true and what to do when criteria is false. It means else block is optional.

```
Syntax:

:

if
(condition)

{

    :

}

else

{

    :

}

:
```

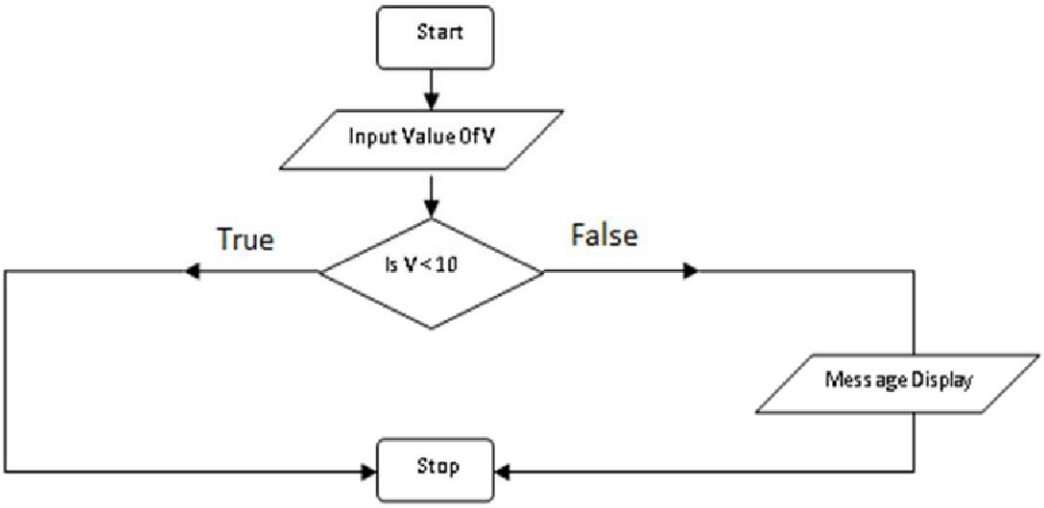


Figure 3.3

Example: Unit-3 (Program-03)

/* Write a program in java to find the greater of two numbers.*/

```
class unit03p03

{
```

```
        public static void main(String[] args)
        {
                int   a, b, max;
                a=10;
                b=12;
                if(a>b)
                {
                        max=a;
                }
                else
                {
                        max=b;
                }
                System.out.println("Greater number is="+max);
        }
}
```

Result:

Greater Number is= 12

Nested *if-else*

If statement can be used to execute anything conditionally. So we can also use if statement within if statement. It is perfectly all right if we write an entire if-else construct within either the body of the if statement or the body of an else statement. This is called 'nesting' of ifs or nested if-else. Nested if can be implemented to any level; but after second level it becomes very typical to debug and understand.

Syntax:

```
if
(condition)
{
```

```
            if( )
            {
            }
        }
        else
        {
            if( )
            {
            }
            else
            {
            }
        }
```

Example: Unit-3 (Program-04)

/* Comparison of three numbers */

```
class unit03p04
{
 public static void main(String[] args)
 {
        int   a, b, c, max;
        a=10;
        b=12;
        c=20;
        if(a>b)
        {
            if(a>c)
            {
                max=a;
            }
            else
```

```
                {
                    max=c;
                }
            }
            else
            {
                if(b>c)
                {
                    max=b;
                }
                else
                {
                    max=c;
                }
            }
            System.out.println("Greatest number is="+max);
    }
}
```

Result:

Greatest Number is= 20

if-else ladder

In this kind of statements numbers of logical conditions are checked for executing various statements. Here, if any logical condition is true the compiler executes the block followed by if condition otherwise it skip and executes else block.

If statement can be used to compare multiple conditions one after another and execute different block for different conditions. This can be done with if-else ladder. This category of if involves a range of conditions and their separate block regarding their execution.

Syntax:

```
if
(condition1)
{
    :
}
```

```
        else if (condition2)
        {
            :
        }
        else if (condition3)
        {
            :
        }
        else
        {
            :
        }
        :
```

Example: Unit-3 (Program-05)

/* Comparison of three numbers */

```
class unit03p05
{
 public static void main(String[] args)
 {
        int  a, b, c, max;
        a=10;
        b=12;
        c=20;
        if(a>=b && a>=c)
        {
            max=a;
        }
        else if(b>=c)
        {
            max=b;
```

```
                }
                else
                {
                        max=c;
                }
                System.out.println("Greatest number is="+max);
        }
}
```

Result:

Greatest Number is= 20

Logical Operator

Logical operators are used to perform action on conditional operands. There are three types of logical operators:

1. *Logical AND operator* (&&): operators are used to combine two conditions and return the 1 or 0. 1 represents true and 0 represents false. This is the binary operator. It returns true only if both the conditions are true otherwise it returns false.

Condition 1	Condition2	Result (condition1 && condition2)
True	True	True
True	False	False
False	True	False
False	False	False

Table 3.2

2. *Logical OR operator* (||)*:* Logical or operator is also used to combine two conditions and return 1 or 0. This is also the binary operator. This operator returns true if any or both the conditions are true. If both the conditions are false only then it returns false.

Condition 1	Condition2	Result (condition1 || condition2)
True	True	True
True	False	True

False	True	True
False	False	False

Table 3.3

3. *Logical NOT operator* (!): Logical not operator is used to invert the condition result. This is the unary operator. If the condition is true it returns false and if the condition is false it return true.

Condition	Result (!condition)
True	False
False	True

Table 3.4

Switch Statement

The switch statement is a multi-way branch statement. If we have multiple choices and we want to choose any of the choice, we may use switch statement. But we may only compare any specific value with equality, we may not compare with any other relational operator like greater than or less than. Switch is the selective statement.

During making match against given value, if match is found in case statement, it execute the statement there onwards till break statement is found. If match is not found, it moves to next case value. At last if no case is matched, it executes default group. Default is optional in switch statement. If we want to compare one value against some values, we may use switch statement or we may use else-if ladder. But we cannot replace else-if ladder with switch statement as in switch, we cannot perform all comparison operations except equality with one value.

The Switch Case vs. Nested if

S.No.	Switch	if
1.	The switch () can only test for equality i.e. only constant values are applicable	The if can evaluate relational or logical expressions.
2.	No two case statements have identical constants in the same switch	Same conditions may be replaced for number of times.
3.	Character constants are automatically converted to integers.	Character constants are automatically converted to integers.

4.	In switch case statement nested if can be used.	In nested if statement switch () case can be used.
5.	Switch statement's program can be created with if-else ladder statement.	All if programs cannot be developed with switch programs.

Table 3.5

Syntax :

```
:
switch(variable)
{
case value1:
:
break;
case value2:
:
break;
case value3:
:
break;
default :
:
}
:
```

Example: Unit-3 (Program-06)

/* Comparison arithmetic operator and perform operation*/

```
class unit03p06
{
 public static void main(String[] args)
 {
        int  a, b, c;
        char op;
```

```
        a=10;
        b=12;
        op='+';
        switch(op)                          /*  switch statement  */
        {
            case '+':                       /*  case value checking */
                c=a+b;
                System.out.println("Result="+c);
                break;
            case '-':                       /*  case value checking */
                c=a-b;
                System.out.println("Result="+c);
                break;
            case '/':                       /*  case value checking */
                c=a/b;
                System.out.println("Result="+c);
                break;
            case '*':                       /*  case value checking */
                c=a*b;
                System.out.println("Result="+c);
                break;
            case '%':                       /*  case value checking */
                c=a%b;
                System.out.println("Result="+c);
                break;
            default:
                System.out.println("Wrong operator");
        }
    }
}
```

Result:

Result=22

In this example the value of op will be checked against all the case values. First of all + will be checked. If it if matched then its blocked will be executed otherwise next case value will be checked. The case matching will be checked till first match. If no case value is matched then default block will be executed. We must use break statement before starting new case value otherwise all the statements after matching will be executed. Case value is ended with colon (:) not with semicolon.

Check Your Progress

1. Write a program in java to convert capital case letter into small case letter.
2. Write a program in java to check for even or odd number.
3. Write a program in java to check weather user can cast his vote or not.
4. Write a program to check whether any symbol is arithmetic operator or not with the help of switch statement.
5. Write a program to check whether year is a leap year or not.
6. Write a program to input any alphabet and check whether it is vowel or consonant.
7. Write a program to input any alphabet and check whether it is vowel or consonant.
8. Write a program to check whether rectangle is square or not.

3.7 ITERATIVE STATEMENTS

Iterative is the non-sequential statement in programming. It is used to implement the reusability concept in code. It is also known looping statement. Looping execute the single or group of statement for multiple times. The number times is to be decided by programmer. It may be fixed number of times or variable number of times. In some cases it may also be infinite loop. But important thing is that single or group will be executed repeatedly. So we can also say that iterative statement involves the repetitive statement. If we write some statements inside the iterative block that will be executed multiple times continuously till any specific condition. Iterative statement defines a specific block which defines the number of repetition. For example display counting of numbers, table of any number, factorial etc. can be solved with iterative statement.

A loop is defined as a block of statements which are repeatedly executed for certain number of times. The condition may be predetermined or open ended.

Parts of the loop :

1. *Initialization*: Initialization is the first step involved in loop. It specifies the starting value of the counter variable of loop. It is executed only once at the start of loop.

2. *Ending Condition:* Ending condition specifies the terminating condition for loop. Loop is executed till the condition is true. As soon as the condition gets false, loop stops. Condition is checked every time the loop is executed.

3. *Step value:* Step value defines the difference between two terms of the loop counter. If loop starts from smaller starting value to larger value, increment is applied. If loop starts from ending value to smaller value, decrement is applied. Increment or decrement is applied from the second time and then each time.

4. *Loop Body:* Loop body can be any single or a group of statements which executes till condition is true. It contains the actual statements which are required to be executed repeatedly.

Categories of looping

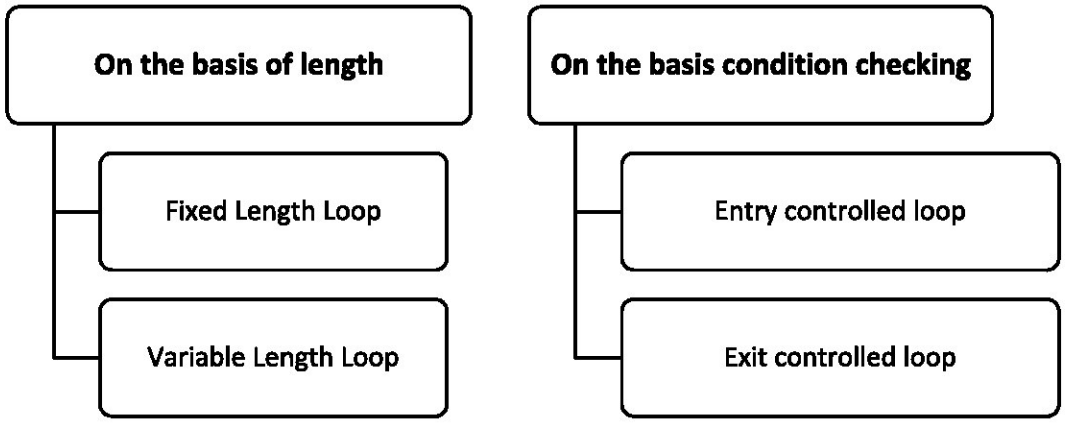


Figure 3.4

Fixed Length Loop :

It is also known as counter controlled loop. Fixed Length loop is used where number of iteration is known at the time of looping. For loop is the type of fixed loop. In this loop initial value of counter, ending condition and step value are known to programmer.

For example:

Requirement	***Initial value***	***Ending condition***	***Step value***
Natural numbers	1	N	+1
Natural number in reverse order	N	1	-1

Even numbers	2	N	+2
Odd Numbers	1	N	+2

Table 3.6

In the above table N is the integer variable, where value is input by user.

Variable length Loop :

Variable length loop is a kind of loop which is executed till specified condition is true. Sometimes programmer doesn't know the number of iterations then variable length loop is used. Its syntax does not include initialization and step value. It is variable length loop because with the help of loop we cannot specify how many times loop will execute. While and do-while is the type of variable length loop. For example to find the LCM, HCF, Binary search, sum of digits etc. we have to use variable length loop.

Fixed Length vs. Variable length Loop

S.No.	Fixed Length Loop	Variable length Loop
1.	In fixed length loop numbers of iterations are known.	In variable length loop numbers of iterations are not known.
2.	In this loop syntax contains initialization, condition and step value.	In this loop syntax contains only condition.
3.	For loop is fixed length loop.	Wshile and do-while loop are the variable length loops.

Table 3.7

Entry controlled Loop:

It is also known as top tested loop. Entry controlled loop is the loop where terminating condition is tested from the start of looping. For and while loops are the top tested loop.

Exit controlled Loop:

It is also known as Bottom tested loop. Exit controlled loop is the loop where terminating condition is tested at the bottom of looping. Do-while loop is the bottom tested loop. Exit controlled loop will execute one time even condition is false on first time. So we can say that when we need loop to be execute at least one time irrespective of the condition, we have to use do-while loop. If the condition gets false on the second time there is no difference between entry controlled and exit controlled loop.

Entry controlled Loop vs. Exit controlled Loop

S.No.	Entry controlled Loop	Exit controlled Loop
1.	Condition is checked at the top of loop.	Condition is checked at the bottom of loop.
2.	It is used to execute till the condition is true.	It is used to execute till the condition is true but at least one time irrespective of the condition.
3.	If the condition gets false on first time then loop will not execute.	If the condition gets false first time, loop will execute one time.
4.	For loop and while are top length loop.	Do-while loop is the variable length loops.

Table 3.8

Types of Loop :

The JAVA language supports three types of loop control statements.

a. The *for* loop

b. The *while* loop

c. The *do-while* loop

d. For each loop (will be discussed in array)

The *for* Loop

For loop is the fixed length and entry controlled loop. Whenever programmer knows the number of iterations for loop is used. The *for* loop allows to execute a set of instructions until a certain condition is satisfied.

Syntax of *for* loop

```
:
for (initialization ; test condition ; increment/decrement)
{
    Statement1;
    Statement2;
}
:
```

Flowchart of *for*-loop

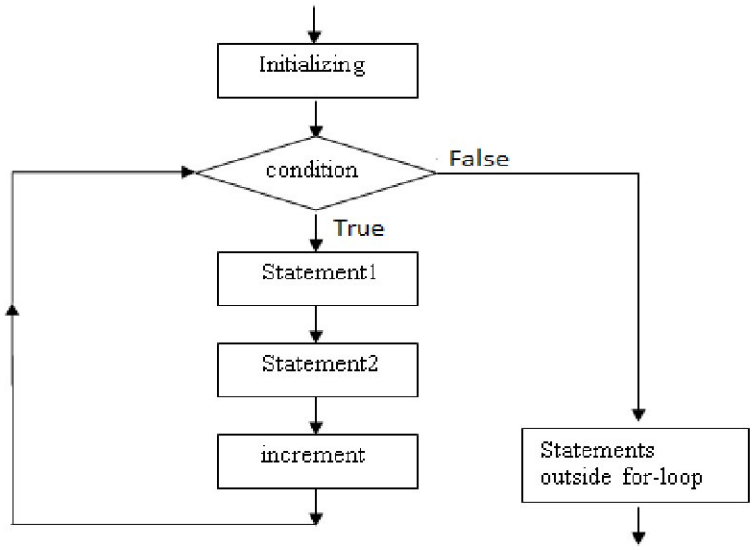


Figure 3.5

The for loop is the simplest and most commonly used loop. The loop consists of three expressions. The first expression is used to initialize the counter, the second is used to specify the terminating condition of loop. Loop will be executed till condition is true. Increment/decrement will specify the step value of counter. You must separate these three major sections by semicolons. The for loop is more powerful when compared to other loops. The number of iterations required to execute the body of the loop is compared by using the formula.

Number of iteration = (Final Value(FV) - Initial Value(IV) + Step Increment(SI)) / (Step Increment(SI))

for Loop Variations

The previous discussion described the most common form of the for loop. However, several variations in for loop are allowed that increase its power, flexibility, and applicability to certain programming situations. One of the most common variations uses the comma operator to allow two or more variables to control the loop.

The Infinite Loop: Although you can use any loop statement to create an infinite loop, **for** is traditionally used for this purpose. Since none of the three expressions that form the **for** loop are required, you can make an endless loop by leaving the conditional expression empty, as here:

```
for( ; ; ) System.out.println("This loop will run forever.");
```

When the conditional expression is absent, it is assumed to be true. You may have an initialization and increment expression, but programmers more commonly use the **for(;;)** construct to signify an infinite loop. Actually, the **for(;;)** construct does not guarantee an infinite loop because a **break** statement, encountered anywhere inside the body of a loop, causes immediate termination.

Program control then resumes at the code following the loop, as shown here:

```
ch = 5;
for( ; ; )
{
        ch++;
        if(ch == 45)
            break;              /* exit the loop */
}
System.out.println(Loop ends now.");
```

for Loops with No Bodies: A statement may be empty. This means that the body of the **for** loop (or any other loop) may also be empty. You can use this fact to simplify the coding of certain algorithms and to create time delay loops

for(i=1; i<=10; i++) ;

As you can see, this loop has no body — and no need for one either.

Time delay loops: The following code shows how to create one by using **for**:

for(t=0; t < SOME_VALUE; t++) ;

The only purpose of this loop is to eat up time.

Syntax	**Output**	**Remarks**
for(a=0; a<=10; a++) System.out.println(a);	Display value from 1 to 10	'a' is increased from 0 to 10. Curly braces are not necessary. Default scope of for loop is one statement after loop.
for(a=10; a>=0; a--) System.out.println(a);	Display value from 10 to 0	'a' is decreased from 10 to 0.
for(; ;) System.out.println(a);	Infinite loop	No terminating statement
for(a=0; a<=20;) System.out.println(a);	Infinite loop	'a' is neither increased nor decreased.

Table 3.9

Example: Unit-3 (Program-07)

/* Print natural numbers up to n */

```
class unit03p07
{
 public static void main(String[] args)
 {
        int  i,n;
        n=10;
        for(i=1;i<=n;i++)
        {
                System.out.println(i);
        }
 }
}
```

Result:

```
1
2
3
4
5
6
7
8
9
0
10
```

The *while* Loop

While is the variable length and entry controlled loop. This loop contains a condition but not initialization and increment/decrement. The condition may be any expression, and true is any nonzero value. The loop iterates while the

condition is true. When the condition becomes false, program control passes to the line of code immediately following the loop.

```
Syntax of while loop
:
:
while (test condition)
{
      :
      :
}
:
```

The test condition may be any expression. The loop statements will be executed till the condition is true i.e. the test condition is evaluated and if the condition is true, then the body of the loop is executed. When the condition becomes false the execution will be out of the loop.

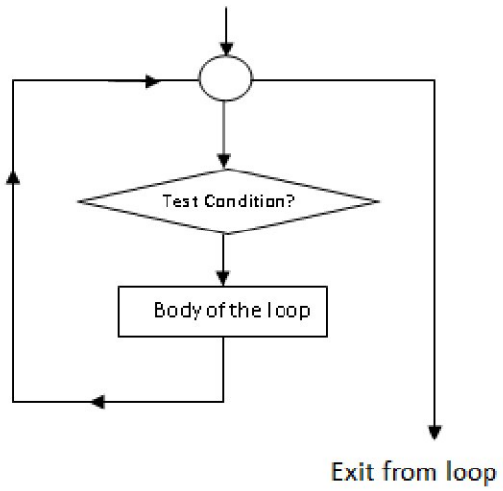


Figure 3.6

Example : Unit-3 (Program-8)

```
/* Print natural numbers up to n */
class unit03p08
{
```

```
        public static void main(String[] args)
        {
                int  i,n;
                n=10;
                i=1;
                while(i<=n)
                {
                        System.out.println(i);
                        i++;
                }
        }
       }
```

Result:

1
2
3
4
5
6
7
8
9
0
10

The *do-while* Loop

Do-while is the variable length and exit controlled loop.

Unlike for and while loops, which test the loop condition at the top of the loop, the do-while loop checks its condition at the bottom of the loop. This means that a do-while loop always executes at least once.

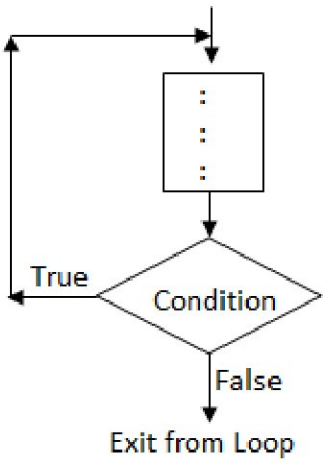


Figure 3.7

The format of *do-while* loop is as follows:

```
Syntax of do-while loop
:
do
{
        :
        :
}while (test condition);
:
```

Do while loop is ended with terminator(;) after condition because it treats the block as a single statement starts from do.

Example: Unit-3 (Program-9)

/* Print natural numbers up to n */

```
class unit03p09
{
 public static void main(String[] args)
 {
        int  i,n;
        n=10;
```

```
                i=1;
                do
                {
                        System.out.println(i);
                        i++;
                }while(i<=n);
        }
    }
```

Result:

1

2

3

4

5

6

7

8

9

0

10

In this program if user input 10 in limit, loop will print natural numbers from 1 to 10. If user input 0 in range, even then it will print 1 because it is the exit controlled loop.

Perhaps the most common use of the **do-while** loop is in a menu selection function. When the user enters a valid response, it is returned as the value of the function. Invalid responses cause a reprompt. Here, the **do-while** loop is a good choice because you will always want a menu function to execute at least once. After the options have been displayed, the program will loop until a valid option is selected.

For Loop vs. While Loop

S.No.	For Loop	While Loop
1.	For loop is the fixed loop.	While loop is the variable loop.

2.	For loop syntax contains initialization, condition and increment/decrement.	While loop syntax contains only condition.
3.	Terminator(;) is used to separate 3 parts.	Only condition is used so separation is not required.
4.	Multiple initialization and increments can be used.	While loop syntax does not include initialization and increment.
5.	for keyword is usd.	while keyword is used.

6.	Syntax-
	for (initialization ; condition ; increment)
	{
	:
	}
		while (test condition)
		{
		:
		}

Table 3.10

While Loop vs. do-while Loop

S.No.	While Loop	Do-while Loop
1.	While loop is the entry controlled loop.	Do-While loop is the exit controlled variable loop.
2.	While loop is executed till condition is true.	Do-while loop is executed till condition is true but at least one time.
3.	While loop is not ended with terminator (;).	While loop is ended with terminator (;).
4.	while keyword is used.	Do and while keywords are used.
5.	Generally used when number of iterations are not known and loop needs to be executed till condition is true.	Generally used in menu based programs where loop needs to be executed at least 1 time even condition is false very first time.

6.	System-
	:
	while (test condition)
	{
	:
	}
		System-
		:
		do
		{
		:
		} while (test condition);

Table 3.11

Check Your Progress

1. Write a program to calculate the sum of first 10 natural number.
2. Write a program to find the factorial value of any number.
3. Write a program to find the power of any number (xn).
4. Write a program to compute the cosine of x. The user should supply x and a positive integer n. We compute the cosine of x using the series and the computation should use all terms in the series up through the term involving x^n

   cos x = 1 - x2/2! + x4/4! - x6/6! .....

5. Print the following pattern:

   1
   1 2
   1 2 3
   1 2 3 4
   1 2 3 4 5

6. Print the following pattern:

   1
   1 2
   1 2 3
   1 2 3 4
   1 2 3 4 5

7. Print the following pattern:

   1
   1 2
   1 2 3
   1 2 3 4
   1 2 3 4 5

8. Print the following pattern:

   1
   1 2 1
   1 2 3 2 1
   1 2 3 4 3 2 1
   1 2 3 4 5 4 3 2 1

9. Print the following pattern:

   *
   * *
   * * *
   * * * *
   * * * * *

3.8 JUMP STATEMENTS

The jumping statements are the control statements which transfer the program execution control to a specific statements.

Java has three types of jumping statements they are break, continue, and return. These statements transfer execution control to another part of the program. Java does not support goto statement like C or C++.

The break statement is used to terminate a case in a switch construct. The break statement is also for termination of a loop, bypassing the normal loop conditional test.

When a break is encountered inside a loop, the loop is terminated and control passes to the statement following the loop body.

Example: Unit-3 (Program-10)

/* Write a program to check for prime number */

```
class unit03p10
{
 public static void main(String[] args)
 {
        int  i,n, flag;
        n=13;
        flag=1;
        for(i=2;i<=n/2;i++)
        {
            if(n%i==0)
            {
                flag=0;
                break;
            }
        }
        if(flag = = 1)
        {
            System.out.println(n+" is prime number");
```

```
                }
                else
                {
                        System.out.println(n+" is not prime number");
                }
        }
}
```

Result:

13 is prime number

Continue statement forces the next iteration of the loop to take place, skipping any code following the continue statement in the loop body. In for loop, the continue statement causes the conditional test and then the re-initialization portion of the loop to be executed. In the while and do. While loops, program control passes to the conditional test. The following program illustrates the usage of the continue statement:

Example: Unit-3 (Program-11)

/* Write a program to print all numbers upto n except numbers divisible by 5.*/

```
class unit03p11
{
 public static void main(String[] args)
 {
        int   i,n, flag;
        n=16;
        for(i=1;i<=n;i++)
        {
                if(i%5==0)
                {
                        continue;
                }
                System.out.println(i);
        }
```

```
        }
}
```

Result:

```
1
2
3
4
6
7
8
9
11
12
13
14
16
```

Labelled Break and Continue

In the case of nested loops to break and continue a particular loop we should go for labelled break and continue statements. The Java labelled loops allows transferring to a particular line or statement.

Example: Unit-3 (Program-12)

/* Write a program to print all numbers upto n except numbers divisible by 5.*/

```
class unit03p12
{
    public static void main(String[] args)
    {
      out:  for(int i=1; i<=100; i++)          // labelled for loop with name out
       {
         System.out.println("outer");
         for(int j=1; j<=100; j++)             // non-labelled for loop
         {
            System.out.println("nested");
```

```
            if(j==2)
            {
                // break; this will exit from inner for loop only
                break out; // this will exit from both for loops because out is
    used
            }
        }
      }
    }
  }
```

Result:

outer

nested

nested

Return statement will return the control of execution from executing method to the calling part of method. Return statement may return at most 1 value, depending on the method declaration. Return statement will be discussed at the time of methods.

Check you Progress

1. What is jump statement? Explain its types.
2. Compare break and continue statement.
3. Print the following pattern:

   1 2 3 4 5 4 3 2 1

   1 2 3 4 4 3 2 1

   1 2 3 3 2 1

   1 2 2 1

   1 1

4. Print the following pattern:

   1 1

   1 2 2 1

   1 2 3 3 2 1

   1 2 3 4 4 3 2 1

   1 2 3 4 5 4 3 2 1

3.9 ARRAYS

An array is a collection of multiple elements of the similar data type that are referred to through a common name. In java array is reference type data type. It means memory will be allocated to array at runtime. Size of the array will be fixed at runtime. Programming basics for execution will be almost similar as in C or C++ but declaration different in Java.

Types of Arrays: There are three types of arrays in java-

- 1 dimensional array
- 2 dimensional array
- Jagged array

Arrays Characteristics

a. Array contains multiple data elements.

b. All the elements are of the similar data type.

c. Elements are stored continuously in the memory internally.

d. Elements are managed with one name only.

e. Elements are accessed with index number. (index number is their position)

f. Index number always starts with 0.

g. Last index number is equal to one less than their size as index number starts from 0.

h. First index number is known as LOWER BOUND and last index number is known as UPPER BOUND. Any other index number outside the bounds cannot be accessed.

i. Array size may be a constant value or input value.

j. Array name contains the address of the first element of the array.

k. Size of the array depends on the number of elements and type of array.

Size of array = number of elements * memory consumed by on element

l. Array element can be directly accessed or changed without disturbing other elements.

Array declaration:

Array declaration determines the size and data type of the array. The size of the array is known as SUBSCRIPT.

Method 1- (Standard Java Notation for one dimensional array)

```
dataType[ ] ArrayName=new dataType[Size];
```

Method 2- (Traditional Notation for one dimensional array)

```
dataType ArrayName[ ]=new dataType[Size];
```

Method 3- (two line notation for one dimensional array)

```
dataType[ ] ArrayName;
ArrayName=new dataType[Size];
```

Method 4- (Dynamic Notation for one dimensional array)

```
dataType ArrayName[ ]={Val 1, Val 2, Val 3, Val 4,… };
```

Method 5-(Standard Java Notation for two dimensional array)

```
dataType[ ][ ] MatrixName= new dataType[RSize][CSize];
```

Method 6- (non-rectangular array or jagged array)

```
int[ ][ ]  MatrixName = new int[RSize][ ];
MatrixName [0] = new int[Col Size 1];
        MatrixName [1] = new int[Col Size 2];
        MatrixName [2] = new int[Col Size 3];
        MatrixName [3] = new int[Col Size 4];
```

Value representation of array in memory:

Internally Index numbers are managed sequentially. For example:

```
int[ ] arr = {11 , 22 , 33 , 44 , 55 , 66 , 77 , 88 , 99 , 100};
```

Given is the details of all array elements of above array arr:

Index number	0	1	2	3	4	5	6	7	8	9
Value stored =>	11	22	33	44	55	66	77	88	99	100
Address =>	1001	1005	1009	1013	1017	1021	1025	1029	1033	1037

Note: 1001 is the hypothetical starting address of array. It may change at runtime.

Total Size= From 1001 to 1040 = 40 bytes (10 elements)

Example: Unit-3 (Program-13)

/* Write a program to calculate the sum of all elements of an integer array*/

```
import java.util.*;              // to use the Scanner class for keyboard from input
class unit03p13
{
        public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);
                int[] arr=new int[10];
                int i,sum;
                for(i=0; i<arr.length; i++)
                {
                        System.out.print("Enter number ");
                        arr[i]=sc.nextInt();
                }
                sum=0;
                for(i=0; i<arr.length; i++)
                {
                        sum=sum+arr[i];
                }
                for(i=0; i<arr.length; i++)
                {
                        System.out.println("Array Element is "+arr[i]);
                }
                System.out.println("Sum of Array Elements are "+sum);
        }
}
```

Result:

Enter number 1

Enter number 2

Enter number 3

Enter number 4

Enter number 5

Enter number 6

Enter number 7

Enter number 8

Enter number 9

Enter number 10

Array Element is 1

Array Element is 2

Array Element is 3

Array Element is 4

Array Element is 5

Array Element is 6

Array Element is 7

Array Element is 8

Array Element is 9

Array Element is 10

Sum of Array Elements are 55

Note: Please note that we have used- import java.util.*;

This statement is used to import util package of java, which contains Scanner class. Scanner class will be used to create the object so that we may take the input via keyboard from user. nextInt() is the method to accept value via keyboard at runtime from user.

Example: Unit-3 (Program-14)

/* Write a program to calculate the sum of all elements of an 2D integer array*/

```
import java.util.*;
class unit03p14
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);
		int[][] mat=new int[3][4];
		int i , j, sum;

		for(i=0; i<mat.length; i++)   // loo[ for input
		{
			for(j=0;j<mat[i].length;j++)
			{
				System.out.print("Enter number ");
				mat[i][j]=sc.nextInt();     // input via keyboard
			}
		}

		sum=0;
		for(i=0; i<mat.length; i++)    // loop for sum calculation
		{
			for(j=0;j<mat[i].length;j++)
			{
				sum=sum+mat[i][j];
			}
		}

		for(i=0; i<mat.length; i++)    // loop to display matrix form
		{
			for(j=0;j<mat[i].length;j++)
```

```
                {
                        System.out.print(mat[i][j]+ "  ");    //display element
with space
                }
                System.out.println();   // to insert new line after row
            }

            System.out.println("Sum of Array Elements are "+sum);
        }
}
```

Result:

```
Enter number 1
Enter number 2
Enter number 3
Enter number 4
Enter number 5
Enter number 6
Enter number 7
Enter number 8
Enter number 9
Enter number 10
Enter number 11
Enter number 12

1  2  3  4
5  6  7  8
9  10  11  12

Sum of Array Elements are 78
```

For Each Loop

In the earlier section, we left for each loop. Now we will discuss it.

For each loop is the extension of for loop. It is used for iteration in collection. When we have a collection like array and we need to use all elements of collection one by one, we may use for each loop. This loop will read all values of collection one after another. But one thing is to be keep in mind that this loop can't update the value of collection.

In this loop we declare one representative of collection that will render all elements one after another sequentially. It can't skip or repeat any element. One additional benefit of using for each loop is that we don't need to use the index number in this case. Use of for each loop totally depends on the programmer. For loop may be used for this purpose as well.

Syntax :

```
:
:
for(datatype var : ArrayName)
{
  :
}
:
:
```

Example: Unit-3 (Program-15)

/* Write a program to calculate the sum of all elements of an integer array via for each loop*/

```
import java.util.*;
class unit03p15
{
        public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);
                int[] arr=new int[10];
                int i,sum;
                for(i=0; i<arr.length; i++)
```

```
            {
                System.out.print("Enter number ");
                arr[i]=sc.nextInt();
            }
            sum=0;
            for(int x : arr)
            {
                sum=sum+x;
            }
            for(int x : arr)
            {
                System.out.println("Array Element is "+x);
            }
            System.out.println("Sum of Array Elements are "+sum);
        }
}
```

Result:

```
Enter number 1
Enter number 2
Enter number 3
Enter number 4
Enter number 5
Enter number 6
Enter number 7
Enter number 8
Enter number 9
Enter number 10

Array Element is 1
Array Element is 2
```

```
Array Element is 3
Array Element is 4
Array Element is 5
Array Element is 6
Array Element is 7
Array Element is 8
Array Element is 9
Array Element is 10
Sum of Array Elements are 55
```

Check Your Progress

1. Write a Java Program to find the largest element from an integer array.
2. Write a Java Program to find the occurrence of any element in an integer array.
3. Write a Java Program to display all even elements of array.
4. Write a Java Program to display the left diagonal element of matrix in matrix format.
5. Write a Java Program to display the right diagonal element of matrix in matrix format.
6. Write a Java Program to display the corner element of matrix in matrix format.
7. Write a Java Program to find the sum of two matrices into third matrix.
8. Write a Java Program to display the transpose of matrix.
9. Write a Java Program to display the upper triangular of matrix.
10. Write a Java Program to calculate and display sum of all rows separately of matrix.

3.10 SUMMARY

Java is one of the most popular programming language. It supports almost all required programming constructs. It supports sequential statements for linear execution. It supports selective statement like if, switch for selective execution. It also supports for, while, do-while for repetitive execution. It introduces for-each loop for accessing individual values form collection like array. Arrays are

dynamic in java. Java does not support pointers like C++ but array does that at runtime in java. It also has one new feature called labelled loop. We may move out of the nested loop directly from any nested level.

Here is the summary of some programming construct in java:

```
if (Condition)
{
   // statements;
}
```

```
if (condition)
{
   // statements;
}
else
{
   // statements;
}
```

```
switch (expression)
{
   case value1:
         //statement;
         break;
   case value2:
         //statement;
         break;
   default:
         //statement;
}
```

```
if (condition)
{
   // statements;
}
else if (condition)
{
   // statements;
}
```

```
do
{
   //Loop body
}while(condition);
```

```
while(conditions)
{
     //Loop body
}
```

```
for(initialization;test condition;increment)
{
     //Loop body
}
```

```
for(int x : ArrName)
{
     //Loop body
}
```

```
for(initialization;test condition;increment)
{
   if(condition)
   {
      //statements
      continue;
   }
}
```

3.11 EXERCISE

Q. 1. What is expression in java? Compare expression with statement.

Q. 2. Explain the types of statements in java.

Q. 3. Compare for loop and for-each loop of java.

Q. 4. Compare else-if ladder with switch statement.

Q. 5. Write a program to input marks of five subjects Physics, Chemistry, Biology, Mathematics and Computer. Calculate percentage and grade according to following:

Percentage >= 90% : Grade A

Percentage >= 80% : Grade B

Percentage >= 70% : Grade C

Percentage >= 60% : Grade D

Percentage >= 40% : Grade E

Percentage < 40% : Grade F

Q. 6. Write a program to input electricity unit charges and calculate total electricity bill according to the given condition:

For first 50 units Rs. 3.50/unit

For next 100 units Rs. 3.75/unit

For next 100 units Rs. 4.20/unit

For unit above 250 Rs. 4.50/unit

An additional surcharge of 20% is added to the bill

Q. 7. Write a program to display the Pascal triangle using 2D array.

Q. 8. Write a Java Program to find the largest and second largest element from an integer array.

Q. 9. Write a Java Program to display the lower triangular of matrix.

Q. 10. Write a Java Program to Sort the Array in an Ascending Order.

Q. 11. Write a Java Program to find the product of two matrices into third matrix.

Q. 12. Print the following pattern:

```
1 2 3 4 5 4 3 2 1
1 2 3 4   4 3 2 1
1 2 3       3 2 1
1 2           2 1
1               1
1 2           2 1
1 2 3       3 2 1
1 2 3 4   4 3 2 1
1 2 3 4 5 4 3 2 1
```

Q.13. Write a program to calculate HCF of Two given number.

Q.14. Write a program to print Fibonacci series of n terms:

0 1 1 2 3 5 8 13 24 .....

Q.15. Write a program to calculate the sum of following series.

$1 + 1/2 + 1/3 + 1/4 + 1/5 + \ldots\ldots\ldots 1/n$

Q.16. Write a program to compute sinx for given x. The user should supply x and a positive integer n. We compute the sine of x using the series and the computation should use all terms in the series up through the term involving x^n

$\sin x = x - x3/3! + x5/5! - x7/7! + x9/9! .....$

Q.17. Print the following pattern:

```
*
* *
* * *
* * * *
* * * * *
* * * *
* * *
* *
*
```

Uttar Pradesh Rajarshi Tandon
Open University

Master in Computer Applications

MCS-112

Java Programming

BLOCK

2

OBJECT ORIENTED CONCEPTS AND EXCEPTIONS HANDLING

UNIT-4

Class and Objects

UNIT-5

Inheritance and Polymorphism

UNIT-6

Packages and interfaces

UNIT-7

Exceptions Handling

Course Design Committee

Prof. Ashutosh Gupta — **Chairman**
Director (In-charge)
School of Computer and Information Science, UPRTOU Allahabad

Prof. Suneeta Agarwal — **Member**
Department of CSE
MNNIT Allahabad, Prayagraj

Dr. Upendra Nath Tripathi — **Member**
Associate Professor, Department of Computer Science
Deen Dayal Upadhyaya Gorakhpur University, Gorakhpur

Dr. Ashish Khare — **Member**
Associate Professor, Department of Computer Science
University of Allahabad, Prayagraj

Dr. Marisha — **Member**
Assistant Professor (Computer Science),
School of Science, UPRTOU Allahabad

Mr. Manoj Kumar Balwant — **Member**
Assistant Professor (computer science),
School of Sciences, UPRTOU Allahabad

Course Preparation Committee

Er. Pooja Yadav — **Author (Unit 1 to 14)**
Asst Professor, Dept. of Computer Science & Information Technology, MJPRU, Bareilly

Prof. Abhay Saxena — **Editor**
Professor, DSVV, Shantikunj, Haridwar (UK)-249411

Prof. Ashutosh Gupta — **Director (In-charge)**
School of Computer and Information Science, UPRTOU Allahabad

Dr. Marisha, — **Coordinator**
Asstt. Professor (Computer Science), School of Science, UPRTOU, Prayagraj

Mr. Manoj Kumar Balwant — **Coordinator**
Asstt. Professor (Computer Science), School of Science, UPRTOU, Prayagraj

ISBN :

Printed and Published by Dr. Arun Kumar Gupta Registrar, Uttar Pradesh Rajarshi Tandon Open University, 2020.
Printed By : Chandrakala Universal Pvt. 42/7 Jawahar Lal Neharu Road, Prayagraj.

UNIT-4 CLASS AND OBJECTS

Structure :

4.1 OBJECTIVE

Java is one of the Object oriented programming languages. Java is truly object oriented language so we must know how to implement classes and object in java. So this unit will cover the implementation of classes and objects in java. Various methods for this implementation will be covered. Along with this, members of the classes will also be focused because without that implementation of classes in incomplete. So this is going to be a very interesting unit for the actual implementation of OOPs.

4.2 INTRODUCTION

Java is one of the Object oriented programming languages. Java is truly object oriented language because, in java we must have to create the class. Without class, we can't create execute java program. While in case of languages like C++, we may execute the program without defining the class. In this unit, we will discuss how to implement the class and object in java at various requirements.

4.3 CLASS FUNDAMENTALS

The building block of java that leads to Object Oriented programming is a Class. It is a user defined data type, which holds its own data members and

member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

Class is the collection of data members and methods.

- A class declaration introduces a new reference type in the java program.
- Class is also known as template data.
- Class is the user defined data type.
- Class is the logical concept of data because it does not consume memory at runtime.
- Class acts as a blue print for any problem as it has to be defined only once.
- Class may also be considered as the collection of similar type of objects.
- In class members are encapsulated so members of class cannot be used directly outside the class.
- In class members are encapsulated so outside data cannot be accessed inside the class directly.
- Class members may have different modifiers.
- Class members may have different visibility modes (access specifier).
- Class may be public or non-public.
- Classes may be nested or local.
- Classes may be anonymous.

Its declaration is as given:

```
class ClassName
{
        Data_Member declarations;   //generally private
        Method definitions;         //generally public
}
```

For example:

```
class Student
{
        private int rollno;
        private string name;
        public void input( )
        {
                ....
```

```
        }
        public void display( )
        {
              …
        }
}
```

Note: Students who have worked in C++ must remember that

- There is no semicolon at the end of class definition.
- Methods are to be defined inside the class only.
- Sequence of member declaration will not affect the class syntax.
- Visibility mode has to be specified for each member separately.
- There is no block declaration like C++.
- Every java program must have at least one class.

Objects :

An object is the instance of class. There are several characteristics of object which will elaborate the object.

- We can create multiple objects of one class.
- Objects consumes memory at run time.
- Object consumes memory separately for each object.
- They are also known as physical concept of data.
- Objects can access public members of the class outside the class.
- Objects are called by reference in methods calling as they are created from class and class is the reference type data type.
- Reference of the object is stored in stack memory.
- Memory for values of object is reserved in heap memory when we allocate the memory with new keyword.
- Memory for object is automatically de-allocated by garbage collector when memory is of no-use in program.
- Any object may call any public method with dot operator, which are declared in its class definition.

Syntax of Object declaration is:

Methods 1:

```
ClassName ObjName=new ClassName();
```

Methods 2:

```
ClassName ObjName;
 ObjName=new ClassName();
```

For example:

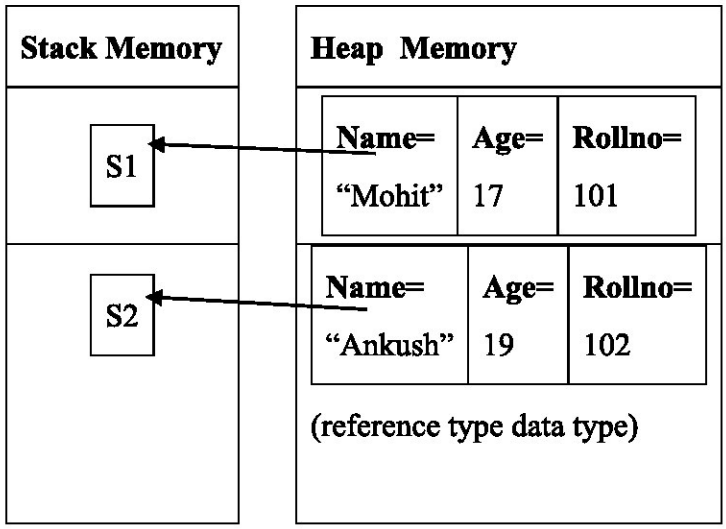
```
Student s1=new Student();
Student s2=new Student();
int a=101;
```

Accessing data members and member functions:

The data members and member functions of class can be accessed using the dot('.') operator with the object. Data member may be a variable, array, object, etc.

For example if the name of object is *s1* and we want to access the member function, we will use this syntax-

```
s1.sname      // not possible outside the class as sname is private
s1.rollno     // not possible outside the class as rollno is private
s1.avg        // not possible outside the class as avg is private
s1.input()    // possible as input() is public
s1.display()  // possible as display() is public
```



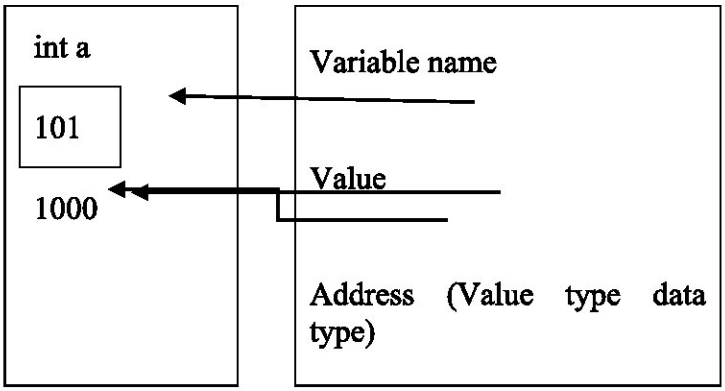


Figure 4.1

Types of Data Members in Class

- **Static data member (class variable)**

  Sometimes we need to store and access the values in one group of objects irrespective of values the object. We need values those are common to all objects. If we change the value of one member of object, it must be reflected to all other objects immediately. In this case we may use static data members. They are also known as class variables.

  They are the members of a class, but not created for any specific object of the class and, therefore, belong only to the class. They consume memory one time for all objects. They are created when the class is loaded at runtime, and exist as long as the class is available at runtime.

- **Non-Static data member (Instance variables)**

  Most of the time we need to store some values for objects separately. That is the normal scenario. In this case we use non-static data members. If we are not declaring any object as static, that will be non-static implicitly.

  They are the members of a class, and created for each object of the class separately. They consume separately for each object. Values may be similar in some objects but memory consumed will be different and changes in one object will not be reflected on other object. Instance variables exist as long as the object they belong to is in use at runtime.

Static & Non-Static Context of data members in Class		
S.No.	**Static data members**	***Non-Static data members***
1	Consumes memory in context of class.	Consumes memory in context of object.
2	Consumes memory only once.	Consumes memory separately for each object.

3	Memory is allocated at the time of class loading.	Memory is allocated at the time of object instantiation.
4	Memory is de-allocated at the end of program.	Memory is de-allocated by garbage collection.
5	Values are common for all objects of that class.	Values are different for all objects of that class.
6	Also known as class variables.	Also known as instance variable.
7	Can access with class reference.	Can access with object reference.

Table 4.1

Access Specifier in Java (Visibility Modes)

Access specifier defines accessibility of any member of the class. Any member of the class can't be accessed outside the class directly due to encapsulation. But we me define any methods to access or update the data members.

There are 3 access specifier in java class

- public
- protected
- private
- friendly (default for package access)

We will discuss only public and private here. Protected will be discussed at the time of inheritance.

Public visibility mode is used to assign the public accessibility to objects. It means public members may be accessed outside the class with the help of objects.

Private visibility mode will restrict the accessing of members within the class only. Outside the class such members can't be used.

All visibility modes are given in following table with their uses-

		Same Packages		Different Packages	
Modifers	**Same Class**	**Sub Class**	**Non-sub Class**	**Sub Class**	**Non-sub Class**

public	Yes	Yes	Yes	Yes	Yes
private	Yes	No	No	No	No
protected	Yes	Yes	Yes	Yes	No
default	Yes	Yes	Yes	No	No

Table 4.2

Example: Unit-4 (Program-01)

/* Write a program in java to create the student class and store one record & display it.*/

```
import java.util.*;
class Student
{
    private int rno;
    private String name;

    public void input( )
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter Student's name ");
        name=sc.next();
        System.out.print("Enter Student's Roll number ");
        rno=sc.nextInt();
    }
    public void display( )
    {
        System.out.println("\nStudent's Name : "+name);
```

```
                System.out.println("Student's Roll number : "+rno);
        }
}

class unit04p01
{
        public static void main(String[] args)
        {
                Student         s=new Student();
                s.input();
                s.display();
        }
}
```

Result:

```
Enter Student's name Ankur
Enter Student's roll no 101

Student's Name : Ankur
Student's Roll number : 101
```

Calculation in class via method:

We can also do calculations in the class. We have access to the member of the object. We just need to access them in any method.

Example: Unit-4 (Program-02)

/* Create a class in java to create the student class which contains, student's name, roll number, marks in 5 subjects and average marks. Create the two objects to store the records and display the records.*/

```
import java.util.*;
class Student
```

```
{
    private int rno;
    private String name;
    private int[] marks;
    private float avgmarks;

    public void input( )
    {
        marks=new int[5];
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter Student's name ");
        name=sc.next();
        System.out.print("Enter Student's Roll number ");
        rno=sc.nextInt();
        for(int i=0;i<marks.length;i++)
        {
            System.out.println("Enter Student's marks in subject "+(i+1));
            marks[i]=sc.nextInt();
        }
    }

    public void display( )
    {
        System.out.println("\nStudent's Name : "+name);
        System.out.println("Student's Roll number : "+rno);
        for(int i=0;i<marks.length;i++)
        {
            System.out.println("Marks in subject "+marks[i]);
        }
        System.out.println("Student's Average Marks : "+avgmarks);
```

```
    }

    public void calculate( )
    {
        int sum=0;
        for(int i=0;i<marks.length;i++)
        {
            sum=sum+marks[i];
        }
        avgmarks=sum/5.0f;
    }
}

class unit04p02
{
    public static void main(String[] args)
    {
        Student     s=new Student();
        s.input();
        s.calculate();
        s.display();
    }
}
```

Result:

```
Enter Student's name Ankur
Enter Student's roll no 101
Enter Student's marks in subject 1 : 34
Enter Student's marks in subject 2 : 45
Enter Student's marks in subject 3 : 37
Enter Student's marks in subject 4 : 41
```

```
Enter Student's marks in subject 5 : 36
Student's Name : Ankur
Student's Roll number : 101
Marks in subject : 34
Marks in subject : 45
Marks in subject : 37
Marks in subject : 41
Marks in subject : 36
Student's Average Marks : 38.6
```

Check Your Progress

1. Create the book class which contains book ID, ISBN, title, author, publication, edition and number of pages. Store and display the record of 2 books.
2. Create the employee class in java which contains employee ID, name, designation, and salary. Create the object to store the records and display it.
3. Create the Inventory class which contains Item ID, quantity and price. Create two records of it and display it.

4.4 INTRODUCING METHODS

Data members are used to store the values temporarily in the objects. But we also need to add some functionality to those objects. For those functionality, we have to define methods inside the class. Those methods are also known as member function. Visibility modes for methods are same as data members. As we know in C++, we may declare the method inside the class and we may define it outside the class. But in java declaration and definition of method isa one thing. We have to define the method inside the class. We can't define the method outside the method in general.

The public data members are also accessed in the same way given however the private data members are not allowed to be accessed directly by the object.

Accessing the methods are already discussed in the previous topic.

Types of methods in Class

- **Static data member (class methods)**

Static methods are those methods which are accessed via class name. for example-

```
System.out.println(“  ”);
```

In this case println is static method as we have not defined any object to access it. It is accessed via class System directly. We have to specify static keyword to declare any method as static like we do for main( ).

In the same way, we may access the methods Math class for mathematical calculations. Most of the methods of Math class are static.

One thing we have to keep in mind that, if we define any method as static, it may access only static members of that class. We can't access non-static members of that class in static method. Static method may access only static members.

We may also note that main( ) is also static in java as we are calling the class containing that main( ) from the prompt by interpreter directly without object creation.

- **Non-Static data member (Instance methods)**

Unlike static members, non-static members are those which are access via object of that class. If we don't specify static keyword before definition, it will be non-static implicitly. It means, if we define nothing in method definition, it will be non-static.

Non-static method may access, non-static members. Static members may be accessed via classname inside the non-static method. Non-static method may access the non-static members directly.

Memory will be allocated for each object separately, so data stored in various object's data members will be different for each object. But we need to access all those values via method calling of their respective object's name.

For user defined objects, we generally define non-static members but at time when we need some calculation work, we may use static methods.

Static & Non-Static Context of methods in Class		
SNo	**Static methods**	***Non-Static Methods***
1.	Can access with class reference.	Can access with object reference.
2.	They cannot access non-static members.	They cannot access static members.

Table 4.3

Example: Unit-4 (Program-03)

/* Create a class in java to 3 static members and 3 non-static members. All should be integer. Input the number and display the sum of those numbers.*/

```
import java.util.*;
class Number
{
    private int a,b,c;
    static private int x,y,z;

    public void inputA( )                    // non-static method
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter 2 numbers ");
        a=sc.nextInt();
        b=sc.nextInt();
    }

    static public void inputB( )             // static method
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter 2 numbers ");
        x=sc.nextInt();
        y=sc.nextInt();
    }

    public void calculateA( )                // non-static method
    {
        c=a+b;
    }
```

```
    static public void calculateB( )                    // static method
    {
        z=x+y;
    }

    public void displayA( )                              // non-static method
    {
        System.out.println("Number="+a+ " "+b );
        System.out.println("Sum="+c);
    }

    static public void displayB( )                  // static method
    {
        System.out.println("Number="+x+ " "+y);
        System.out.println("Sum="+z);
    }
}
class unit04p03
{

    public static void main(String[] args)
    {
        Number n=new Number();
        n.inputA();                          // non-static method call by
object
        Number.inputB();                     // static method call by class
name
        n.calculateA();
        Number.calculateB();
        n.displayA();
        Number.displayB();
```

```
    }
}
```

Result:

```
Enter 2 numbers : 10 20
Enter 2 numbers : 15 25
Number= 10 20
Sum=30
Number= 15 25
Sum=40
```

Accessing private Data Member of object :

Any data member can't be accessed outside the class directly. Private data member can't be access by object of that class.

If we are in the need of accessing any private data member, we may define any public method which return that value. That's how we may access the private data member.

Example: Unit-4 (Program-04)

/* Create a class in java to create the student class which contains, student's name, roll number, and average marks. Create the two objects to store the records and display the record who scores higher average marks.*/

```
import java.util.*;
class Student
{
    private int rno;
    private String name;
    private float avgmarks;

    public void input( )
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter Student's name ");
        name=sc.next();
```

```
            System.out.print("Enter Student's Roll number ");
            rno=sc.nextInt();
            System.out.print("Enter Student's average marks ");
            avgmarks=sc.nextFloat();
    }

    public void display( )
    {
            System.out.println("\nStudent's Name : "+name);
            System.out.println("Student's Roll number : "+rno);
            System.out.println("Student's Average Marks : "+avgmarks);
    }

    public float getAvgMarks( )
    {
            return avgmarks;
    }
}

class unit04p04
{
    public static void main(String[] args)
    {
            Student     s1=new Student();
            Student     s2=new Student();
            s1.input();
            s2.input();
            if(s1.getAvgMarks() > s2.getAvgMarks())
                    s1.display();
            else
```

```
                s2.display();
        }
    }
```

Result:

```
Enter Student's name Ankur
Enter Student's roll no 101
Enter Student's average subject 1 : 84.5

Enter Student's name Ankit
Enter Student's roll no 101
Enter Student's average subject 1 : 81.5

Student's Name : Ankur
Student's Roll number : 101
Student's Average Marks : 84.5
```

Updating / modifying private Data Member of object :

Any data member can't be accessed outside the class directly. Private data member can't be access by object of that class.

If we are in the need of update any private data member, we may define any public method which update that value. That's how we may access the private data member.

Example: Unit-4 (Program-05)

/* Create a class in java to create the student class which contains, student's name, roll number, and average marks. Create the one objects to store the record and update the average marks with 2 marks. Display the record after update.*/

```
import java.util.*;
class Student
{
    private int rno;
    private String name;
    private float avgmarks;
```

```
    public void input( )
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter Student's name ");
        name=sc.next();
        System.out.print("Enter Student's Roll number ");
        rno=sc.nextInt();
        System.out.print("Enter Student's average marks ");
        avgmarks=sc.nextFloat();
    }

    public void display( )
    {
        System.out.println("\nStudent's Name : "+name);
        System.out.println("Student's Roll number : "+rno);
        System.out.println("Student's Average Marks : "+avgmarks);
    }

    public void setAvgMarks(float marks)
    {
        avgmarks=avgmarks+marks; // updating data member with new
value
    }
}

class unit04p05
{
    public static void main(String[] args)
    {
        Student     s1=new Student();
        s1.input();
```

```
            s1.setAvgMarks(2.0f);   // calling method to update  value
            s1.display();
        }
    }
```

Result:

```
Enter Student's name Ankur
Enter Student's roll no 101
Enter Student's average subject 1 : 84.5


Student's Name : Ankur
Student's Roll number : 101
Student's Average Marks : 86.5
```

Array of objects

We may also define the array of objects. Array declaration of objects will be similar as earlier in java. But in java, in array of objects, we have to allocate memory twice- first for array and second for objects.

For array, we have to allocate once. For objects in array, we have to collocate memory separately for each object preferably via loop.

For example-

```
Student[] s=new Student[5];
        for(int i=0;i<s.length;i++)
        {
            S[i]=new Student( );
        }
```

Example: Unit-4 (Program-06)

/* Create a class in java to create the student class which contains, student's name, roll number, and average marks. Create the five objects to store and display the records. */

```
import java.util.*;
```

```
class Student
{
    private int rno;
    private String name;
    private float avgmarks;

    public void input( )
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter Student's name ");
        name=sc.next();
        System.out.print("Enter Student's Roll number ");
        rno=sc.nextInt();
        System.out.print("Enter Student's average marks ");
        avgmarks=sc.nextFloat();
    }
    public void display( )
    {
        System.out.println("\nStudent's Name : "+name);
        System.out.println("Student's Roll number : "+rno);
        System.out.println("Student's Average Marks : "+avgmarks);
    }
    public void setAvgMarks(float marks)
    {
        avgmarks=avgmarks+marks;
    }
}
class unit04p06
{
    public static void main(String[] args)
```

```
        {
                Student[]s=new Student[5];
                for(int i=0;i<s.length;i++)
                {
                        s[i]=new Student();
                }
                for(int i=0;i<s.length;i++)
                {
                        s[i].input();
                }
                for(int i=0;i<s.length;i++)
                {
                        s[i].display();
                }
        }
  }
```

Result :

```
Enter Student's name Ankur
Enter Student's roll no 101
Enter Student's average subject 1 : 84.5

Enter Student's name Ankit
Enter Student's roll no 102
Enter Student's average subject 1 :74.5

Enter Student's name Mohit
Enter Student's roll no 101
Enter Student's average subject 1 : 55.0
Enter Student's name Subit
Enter Student's roll no 101
```

```
Enter Student's average subject 1 : 44.5


Enter Student's name Neeraj
Enter Student's roll no 101
Enter Student's average subject 1 :70.5


Student's Name : Ankur
Student's Roll number : 101
Student's Average Marks : 86.5


Student's Name : Ankit
Student's Roll number : 102
Student's Average Marks : 76.5


Student's Name : Mohit
Student's Roll number : 103
Student's Average Marks : 55.0


Student's Name : Subit
Student's Roll number : 104
Student's Average Marks : 44.5


Student's Name : Neeraj
Student's Roll number : 105
Student's Average Marks : 70.5
```

Constructor :

Constructor is a special method which is used to initialize the object when the object is created by using the new operator. There are some characteristics of constructor:

- The constructor name must be the same as the class name.
- Constructors cannot return a value not even void.

- One constructor is called separately for each object automatically but only once for one object value.
- It may be parameterized, overloaded or static.

Type of constructors in Java:

- ***Implicit Default constructor***: An implicit default constructor, which is used to set the state of its objects with default values.
- ***Default constructor / non-parameterized constructor***: A default constructor is a constructor without any parameters.
- ***Parameterized constructor***: Parameterized constructor is used to pass user defined values in object state.
- ***Copy constructor***: Copy constructor is used to create a clone of existing object with separate allocated memory.

Note: In java we do not need *Destructor* as Garbage collector de-allocates the memory automatically for unused objects.

Check Your Progress

1. Create the book class which contains book ID, ISBN, title, author, publication, edition and number of pages. Store and display the record of 2 books and display the record which contains more number of pages.
2. Create the employee class in java which contains employee ID, name, designation, and salary. Create the object to store the records. Update the salary of employee with 10% and then display the record.
3. Create the Inventory class which contains Item ID, quantity and price. Create two records of it and display the record with has lower cost. Reduce the code of other item by 5% and display both records.
4. Create the employee class in java which contains employee ID, name, designation, and salary. Create the 10 object to store the records and display them.

4.5 THIS KEYWORD

Java does not support pointers. But in java to refer to current object (with which method is called), we may use this keyword before that data member.

This is generally used when parameter name is similar to data member. In this case to differentiate data member with parameter, we may use this keyword.

This is also very useful when we have to return the current object but we don't have the name of that object in method definition.

Example: Unit-4 (Program-07)

```
/* Create a class in java to create the student class which contains, student's name,
```

roll number, and average marks. Update the average marks with new marks and display the record.*/

```
import java.util.*;
class Student
{
    private int rno;
    private String name;
    private float avgmarks;
    public void input( )
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter Student's name ");
        name=sc.next();
        System.out.print("Enter Student's Roll number ");
        rno=sc.nextInt();
        System.out.print("Enter Student's average marks ");
        avgmarks=sc.nextFloat();
    }
    public void display( )
    {
        System.out.println("\nStudent's Name : "+name);
        System.out.println("Student's Roll number : "+rno);
        System.out.println("Student's Average Marks : "+avgmarks);
    }
    public void setAvgMarks(float avgmarks)
    {
        this.avgmarks=avgmarks;
    }
}
class unit04p07
```

```
{
    public static void main(String[] args)
    {
            Student s=new Student();;
            s.input();
            Scanner sc=new Scanner(System.in);
            System.out.print("\nEnter Student's new average marks ");
            float avg=sc.nextFloat();
            s.setAvgMarks(avg);
                        s.display();
    }
}
```

Result:

```
Enter Student's name Ankur
Enter Student's roll no 101
Enter Student's average subject 1 : 84.5
Enter Student's new average subject 1 :74.5
Student's Name : Ankur
Student's Roll number : 101
Student's Average Marks : 74.5
```

Check Your Progress

1. Create the book class which contains book ID, ISBN, title, author, publication, edition and number of pages. Store 10 record of books and display find the book on the basic of author.
2. Create the employee class in java which contains employee ID, name, designation, and salary. Create the 10 object to store the records. Arrange the employee records on the basis of their salary in ascending order and display them.
3. Create the Inventory class which contains Item ID, quantity and price. Create 10 records of it and display all record with average cost of all 10 books.

4. Create the employee class in java which contains employee ID, name, designation, and salary. Create the 10 object to store the records. Increase the salary with 10% of all employees who earn less the 10000. Display the updated records of all employees.

4.6 USING OBJECTS AS PARAMETERS

Java is one of the Object oriented programming languages. So it supports message passing. Message passing is basically meant for transmitting message from one object to another. Object may also be sent as parameter. Object may also be returned from any method. And both operations may be done in one methods as well.

Example: Unit-4 (Program-08) (with non-static method)

/* Create a class in java to create the student class which contains, student's name, roll number, and average marks. Create two objects of student. Compare both objects on the basis of their average marks and display the record with higher average marks.*/

```
import java.util.*;
class Student
{
    private int rno;
    private String name;
    private float avgmarks;

    public void input( )
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter Student's name ");
        name=sc.next();
        System.out.print("Enter Student's Roll number ");
        rno=sc.nextInt();
        System.out.print("Enter Student's average marks ");
        avgmarks=sc.nextFloat();
    }
```

```
    public void display( )
    {
        System.out.println("\nStudent's Name : "+name);
        System.out.println("Student's Roll number : "+rno);
        System.out.println("Student's Average Marks : "+avgmarks);
    }
    public float getAvgMarks()
    {
        return avgmarks;
    }
    public Student compare(Student s1)  // non-static method
    {
        Student max;
        if(s1.getAvgMarks() > this.getAvgMarks())
            max=s1;
        else
            max=this;
        return max;
    }
}
class unit04p08
{
    public static void main(String[] args)
    {
        Student s1=new Student();
        Student s2=new Student();
        Student s3;
        s1.input();
        s2.input();

        s3=s1.compare(s2);
```

```
            s3.display();
        }
    }
```

Result :

```
Enter Student's name Ankur
Enter Student's roll no 101
Enter Student's average subject 1 : 84.5

Student's Name : Subit
Student's Roll number : 104
Student's Average Marks : 44.5

Student's Name : Ankur
Student's Roll number : 101
Student's Average Marks : 84.5
```

Above program is solved via non-static method. Same may be done with static method.

Example: Unit-4 (Program-09) (with static method)

/* Create a class in java to create the student class which contains, student's name, roll number, and average marks. Create two objects of student. Compare both objects on the basis of their average marks and display the record with higher average marks.*/

```
import java.util.*;
class Student
{
    private int rno;
    private String name;
    private float avgmarks;

    public void input( )
```

```
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter Student's name ");
        name=sc.next();
        System.out.print("Enter Student's Roll number ");
        rno=sc.nextInt();
        System.out.print("Enter Student's average marks ");
        avgmarks=sc.nextFloat();
    }

    public void display( )
    {
        System.out.println("\nStudent's Name : "+name);
        System.out.println("Student's Roll number : "+rno);
        System.out.println("Student's Average Marks : "+avgmarks);
    }

    public float getAvgMarks()
    {
        return avgmarks;
    }

    public static Student compare(Student s1, Student s2)
    {
        Student max;
        if(s1.getAvgMarks() > s2.getAvgMarks())
            max=s1;
        else
            max=s2;
        return max;
    }
}

class unit04p09
```

```
{
    public static void main(String[] args)
    {
            Student s1=new Student();
            Student s2=new Student();
            Student s3;
            s1.input();
            s2.input();
            s3=Student.compare(s1,s2);
            s3.display();
    }

}
```

Result :

Enter Student's name Ankur

Enter Student's roll no 101

Enter Student's average subject 1 : 84.5

Student's Name : Subit

Student's Roll number : 104

Student's Average Marks : 44.5

Student's Name : Ankur

Student's Roll number : 101

Student's Average Marks : 84.5

Check Your Progress

1. Create the employee class in java which contains employee ID, name, designation, and salary. Create the two object to store the records. Find and display the record who earns higher salary. Compare it with static method.

2. Create the Inventory class which contains Item ID, quantity and price. Create 2 records of it and display the record where cost is lower. Compare it with non-static method.

4.7 METHOD OVERLOADING

Each method has a signature, which comprises the name of the method, and the types and order of the parameters in the formal parameter list. Several method implementations may have the same name, as long as the method signatures differ. This is called method overloading. Since overloaded methods have the same name, their parameter lists must be different. Method overloading can be used when the same logical operation requires multiple implementations. The return type has no concern with parameter list, return type may or may not be different. For example:

```
pubic int area(int side) {….}  // one integer argument
pubic float area(float radius) {….}     // one float argument
pubic int area(int length, int breadth) {….}    //two integer arguments
```

Example: Unit-4 (Program-010)

/* Create a class in java to calculate the area of circle, rectangle and square. Solve it with method overloading. */

```
import java.util.*;

class Shape
{
    static public void area(float r)
    {
        float area=3.14f*r*r;
        System.out.println("Area of cicle="+area);
    }

    static public void area(int l, int b)
    {
        int area=l*b;
        System.out.println("Area of Rectangle="+area);
    }
    static public void area(int s)
    {
        int area=s*s;
```

```
                System.out.println("Area of Square="+area);
        }
}
class unit04p10
{
        public static void main(String[] args)
        {
                Shape.area(10);
                Shape.area(5,8);
                Shape.area(3.5f);
        }
}
```

Result:

Area=100

Area=40

Area=38.4650004

Calling of the method will totally depends on the number of parameter and type of parameter. In java, method name, parameter list is also known as signature of method.

Check Your Progress

1. What is overloading? Describe its benefits.
2. Write a program in java to calculate the volumes of various shapes with method overloading.

4.8 GARABAGE COLLECTION

Java is one of the Object oriented programming languages. It also supports static and dynamic memory management. Static memory is implicitly allocated and implicitly de-allocated, we need not to do anything extra. It will be defined at compilation time. For such member generally memory is allocated in stack memory.

But at times, we need to allocate the memory for some time and after that we don't need that. This may be done with the help of dynamic memory management. Programmer needs to allocate the memory explicitly. Once the use of that memory is completed, java automatically frees that memory. That memory

is managed in heap memory. This facility is provided by garbage collection in java.

Garbage Collection is the facility in java which free the unused dynamic memory in java.

Check Your Progress

1. What is garbage collection? Compare stack and heap memory.

4.9 THE FINALIZE( ) METHOD

Java is one of the Object oriented programming languages. It supports dynamic memory management. It also supports constructor and destructor. Destructor is used to free the unused memory at runtime. In java, that is done by garbage collection.

But at times, we need to perform some extra operations, that may be done by finalise method() in java. Finalize() acts like a method to de-allocate the help memory at runtime.

Although we generally don't need it as garbage collection is there in java.

Check Your Progress

1. What is dynamic memory in java? Where it is allocated and how it is de-allocated in java.

4.10 SUMMARY

Java is one of the most popular Object oriented programming language. It implements the concept of class and objects. Class definition in java is different with C++. There are various visibility modes (modifiers / access specifiers) in java to restrict the members accessing in java. In Java, we may implement classes and objects in various formats. Members I java, may also be created in regard with objects separately and; in regard with class that will be common to all objects. Java does not support pointers but it supports this keyword to access the reference of current object in method definition. Java also supports garbage collection to manage memory dynamically.

4.11 EXERCISE

Q. 7. Write a program to print the area of two rectangles having sides (4,5) and (5,8) respectively by creating a class named 'Rectangle' with a method named 'Area' which returns the area.

Q. 8. Write a program to print the area of a rectangle by creating a class named 'Area' having two methods. First method named as 'setDim' takes length and breadth of rectangle as parameters and the second method named as 'getArea' returns the area of the rectangle. Length and breadth of rectangle are entered through keyboard.

Q. 9. Write a program in java to add two distances in inch-feet by creating a class named 'AddDistance'.

Q. 10. State the output-

```
public class A
{
  static
  {
    System.out.println(1);
  }
  static
  {
    System.out.println(2);
  }
  static
  {
    System.out.println(3);
  }
  public static void main(String[] args)
  {
    A a;
  }
}
```

Q. 11. State the output-

```
class A
{
  static int first;
  static String second;
  static
```

```
    {
        System.out.println(1);
        first = 100;
    }
    static
    {
        System.out.println(2);
        second = "SECOND";
    }
}
public class StaticInitializationBlock
{
    static
    {
        System.out.println(3);
    }
    public static void main(String[] args)
    {
        System.out.println(4);
        System.out.println(A.first);
        System.out.println(A.second);
    }
}
```

UNIT-5 INHERITANCE AND POLYMORPHISM

Structure :

5.1 OBJECTIVE

Inheritance is one of the most powerful technique for reusability so its implementation is also very important. This unit will discuss the inheritance and its implementation in java programming language. Java also provides several modifiers along with inheritance so enhance stringiness in OOPs concept, those will also be discussed. Polymorphism in java is also going to be discussed here. Method overriding is very essential in java, so it will also be discussed in this unit.

5.2 INTRODUCTION

Inheritance provides the concept of reusability of code. And can easily debug error and duplicate codes. Because one class inherit all the features of existing class and add a new feature so this makes code cleaner, understandable and extendable. In this unit we will discuss the inheritance in java, its types and implementation in java. Besides that we will also discuss polymorphism and the implementation types of polymorphism.

5.3 INHERITANCE BASICS

Inheritance is the fundamental mechanisms for code reuse in OOP. It allows new classes to be derived from an existing class. Inheritance is the feature

by which members of one class may be reused in other class. The source class is called Super class or Base Class. The class in which one class is being inherited is called sub or derived class. In Java, implementation inheritance is achieved by extending classes.

```
class SUPER
{
  .......
}
class SUB extends SUPER                    // SUB class inherits SUPER class
{
  ......
}
```

Inheritance defines the relationship is-a (also called the superclass–subclass relationship). This means that an object of a subclass is-a superclass object, and can be used wherever an object of the superclass can be used.

Check Your Progress

1. What is inheritance?
2. How inheritance is implemented in java? Explain with example.

5.4 ACCESSING MEMBERS

In case of inheritance, members of class in which they are defined, can be accessed directly inside the class. If we want to access the member of super class then they may be accessed with their names.

For example, if we have defined input ( ) in super class and want to access that method in sub class then it may be called directly. But we must make sure that that method should not be private in the super class. Private members can't be accessed I the sub class.

Here is the example to show the basic functioning of inheritance-

Example: Unit-5 (Program-01)

/* Create a person class in java which contains name and age. Inherit that class in to student class which contains roll number and course. Create the object of both classes.*/

```
import java.util.*;
class Person                              // SUPER CLASS
{
```

```
    private String name;   // SUPER CLASS PRIVATE MEMBER
    private int age;       // SUPER CLASS PRIVATE MEMBER

    public void input()    // SUPER CLASS PUBLIC METHOD
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter name ");
        name=sc.next();
        System.out.print("Enter age ");
        age=sc.nextInt();
    }

    public void display()  // SUPER CLASS PUBLIC METHOD
    {
        System.out.println("Name : "+name);
        System.out.println("Age : "+age);
    }
}
class Student extends Person          // SUB CLASS INHERITING
SUPER CLASS
{
    private String course;        // SUB CLASS PRIVATE MEMBERS
    private int rno;              // SUB CLASS PRIVATE
MEMBERS
    public void getData()         // SUB CLASS PUBLIC METHOD
    {
        input();                  // ACCESSING SUPER CLASS METHOD
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter course");
        course=sc.next();
        System.out.print("Enter roll number ");
        rno=sc.nextInt();
    }
    public void putData()         // SUB CLASS PUBLIC METHOD
    {
```

```
            display();              // ACCESSING SUPER CLASS METHOD
            System.out.println("Course : "+course);
            System.out.println("Roll Number : "+rno);
        }
    }
    class unit05p01
    {
        public static void main(String[ ] args)
        {
            Person p=new Person();              // Super Class Object
            Student s=new Student();    // Sub Class Object
            System.out.println("Person Record ");
            p.input();
            p.display();
            System.out.println("Student Record ");
            s.getData();
            s.putData();
        }
    }
```

Result :

```
Person Record
Enter Name : Ankit
Enter Age : 18
Name : Ankit
Age: 18
Student Record
Enter Name : Mohit
Enter Age :21
Enter course : MBA
Enter roll number :  101

Name : Ankit
Age: 18
```

Course : MBA
Roll number : 101

There are 4 access specifier in java class

- public
- protected
- private
- friendly (default for package access)

We will discuss only public, private and protected here. friendly will be discussed at the time of packages.

Public visibility mode is used to assign the public accessibility to objects. It means public members may be accessed outside the class with the help of objects.

Private visibility mode will restrict the accessing of members within the class only. Outside the class such members can't be used.

All visibility modes are given in following table with their uses-

		Same Packages		Different Packages	
Modifers	**Same Class**	**Sub Class**	**Non-sub Class**	**Sub Class**	**Non-sub Class**
public	Yes	Yes	Yes	Yes	Yes
private	Yes	No	No	No	No
protected	Yes	Yes	Yes	Yes	No
default	Yes	Yes	Yes	No	No

Table 5.1

Above table clearly shows that private member can't be accessed in the sub class. It must be public or private. If we have not defined any modifier in class member, even in that case that may be accessed in the same package. Packages will be discussed later. But for now, we may assume one package as one folder.

Accessing Super class member when sub class and super class member name is same:

There may be situation when we have one super class having member with same name as member of subclass is defined. That may be resolved with this keyword and super keyword. this keyword is used to refer to current object and super keyword is used to access the member of super class.

For example:

```
class Person                        // SUPER CLASS
{
    private String name;
    private int age;
    public void input()
    {       }
    public void display()
    {       }
}

class Student extends Person
{
    private String course;
    private int rno;

    public void input()             // input method in sub class with same as in
super class
    {       }

    public void putData()
    {       }
}
```

In the above scenario we may access the input() of person class in Student class with super keyword.

```
super.input();
```

now in method call via object, if we call the method via super class object than method of super class will be called and if we call the method via object of sub class than method of sub class will be called.

Example: Unit-5 (Program-02)

/* Create a person class in java which contains name and age. Inherit that class in to student class which contains roll number and course. Create the object of both classes.*/

```
import java.util.*;
class Person
{
    private String name;
    private int age;

    public void input()
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter name ");
        name=sc.next();
        System.out.print("Enter age ");
        age=sc.nextInt();
    }
    public void display()
    {
        System.out.println("Name : "+name);
        System.out.println("Age : "+age);
    }
}
class Student extends Person
{
    private String course;
    private int rno;

    public void input()
    {
```

```
            super.input();  // CALLING METHOD OF SUPER CLASS
            Scanner sc=new Scanner(System.in);
            System.out.print("Enter course");
            course=sc.next();
            System.out.print("Enter roll number ");
            rno=sc.nextInt();
        }
        public void putData()
        {
            display();      // CALLING METHOD OF SUPER CLASS
            System.out.println("Course : "+course);
            System.out.println("Roll Number : "+rno);
        }
    }
    class unit05p02
    {
        public static void main(String[] args)
        {
            Person p=new Person();
            Student s=new Student();
            System.out.println("Person Record ");
            p.input();              // SUPER CLASS METHOD CALL
            p.display();
            System.out.println("Student Record ");
            s.input();              // SUB CLASS METHOD CALL
            s.putData();
        }
    }
```

Result:

```
Person Record
Enter Name : Ankit
Enter Age : 18
Name : Ankit
```

```
Age: 18
Student Record
Enter Name : Mohit
Enter Age :21
Enter course : MBA
Enter roll number :  101
Name : Ankit
Age: 18
Course : MBA
Roll number :  101
```

Note: In the above example, we must remember that if we call the input() via super class object than method of super class will be called but if we call the input() via sub class object than method of sub class will be called.

Check Your Progress

1. Create the employee class which contains employee id, name and salary. Inherit that class in to manager class which additionally contains area. Create the object of both class. Make sure that to display the values of both object method name must be display().

5.5 TYPES OF INHERITANCE

Inheritance is of five types:

1) Single Inheritance
2) Multilevel Inheritance
3) Multiple Inheritance
4) Hierarchical Inheritance
5) Hybrid Inheritance

	Single Inheritance	**Hierarchical Inheritance**	**Multilevel inheritance**	**Multiple Inheritance**	**Hybrid Inheritance**
Super class	1	1	1	Many	1

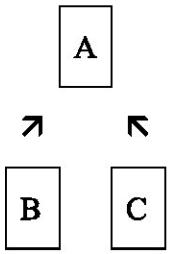
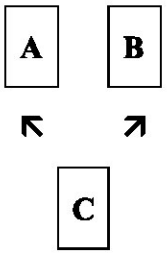
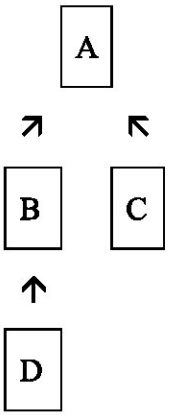
Sub class	1	Many	1	1	Any
Structure					
Syntax	class A {……..} class B extends A {……..}	class A {……..} class B extends A {……..} class C extends A {……..}	class A {……..} class B extends A {……..} class C extends B {……..}	Not available in java via classes	class A {……..} class B extends A {……..} class C extends A {……..} class D extends B {……..}

Table 5.2

Single Inheritance

When one derive class is inherit from one base class, then this type of inheritance is known as *Single Inheritance.*

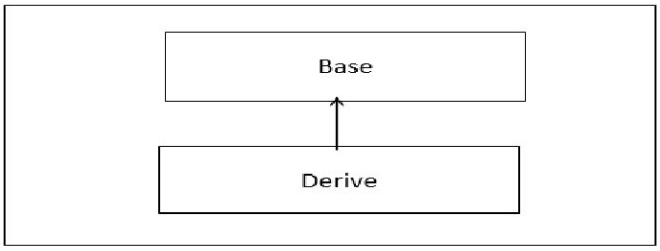


Figure 5.1

Multilevel Inheritance

When derive class inherit from the base class and another derive class inherit from this derived class. This form of inheritance is known as *Multilevel Inheritance*. Because more than one class is derived at different level.

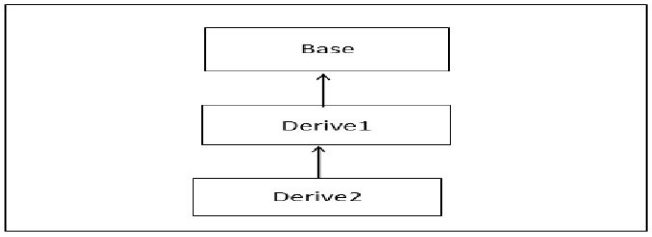


Figure 5.2

Multiple Inheritance

When a derive class inherit from more than one base class than this type of inheritance is known as *Multiple Inheritance.* This is not available in java via classes.

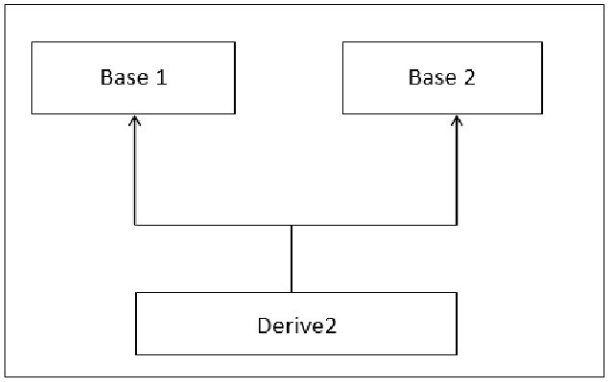


Figure 5.3

Hierarchical Inheritance

When more than one derive class inherit from one base class. This form of inheritance is known as *Hierarchical Inheritance.*

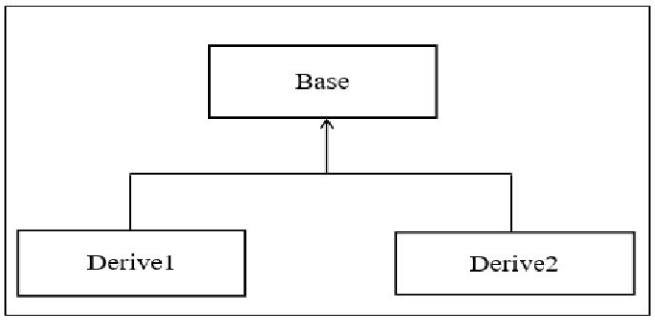


Figure 5.4

Hybrid Inheritance

When derive class inherit from the base class with the combination of two or more different type of inheritance. This form of inheritance is known as *Hybrid Inheritance*.

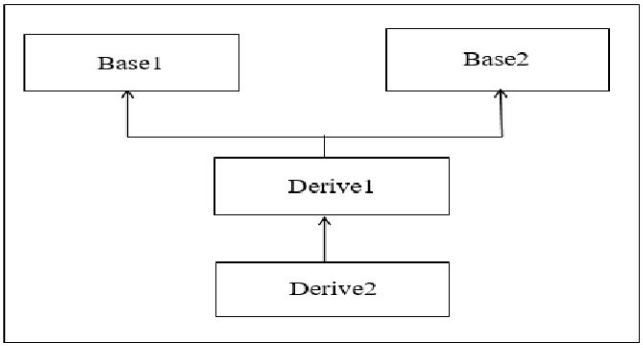


Figure 5.5

Constructor and Destructor in derived class

As we know that constructor plays a vital role in initializing an object. Constructors cannot inherit, while applying inheritance; we usually create objects of derived class. But, if a base class has a parameterised constructor with one or more parameter, then value of these parameters are passed with the help of initialiser list, which is associated in derived class constructor so it is compulsory to have a constructor in derived class and pass the arguments to the base class constructor. When both the base class and derived class have constructors, the base class constructor is executed first and then the constructor in the derived class is executed.

If we have defined constructor in super class and sub class; and we have created the object of super class then constructor of super class will be invoked.

But if we have created the object of sub class than constructor of sub class and super class will be invoked. The sequence in this case will be from super class to sub class. It means constructor of super class will be invoked first than constructor of sub class will be invoked. Destructor will be invoked in reverse order of constructor.

Example: Unit-5 (Program-03)

/* Create a person class in java which contains name and age. Inherit that class in to student class which contains roll number and course. Create the object of both classes.*/

```
import java.util.*;
class Person
{
    private String name;
    private int age;

    public Person()
    {
        System.out.println("Constructor of Person class");
        name="";
        age=0;
    }
    public void display()
    {
        System.out.println("Name : "+name);
        System.out.println("Age : "+age);
    }
}

class Student extends Person
{
    private String course;
    private int rno;

    public Student()
    {
        System.out.println("Constructor of Student class");
```

```
                course="";
                rno=0;
        }

        public void putData()
        {
                display();
                System.out.println("Course : "+course);
                System.out.println("Roll Number : "+rno);
        }
}

class unit05p03
{
        public static void main(String[] args)
        {
                System.out.println("Person Record ");
                Person p=new Person();
                System.out.println("Student Record ");
                Student s=new Student();
        }
}
```

Result:

```
Person Record
Constructor of Person Class

Student Record
Constructor of Person Class
Constructor of Student Class
```

Calling the parameterised Constructor of super class

If super class contains non-parameterised and parameterised constructor, and we have created the object of sub class. In this case non-parameterised constructor of super class will be invoked.

If we wish to invoke the parameterised constructor of super class than we may invoke it with than help of super keyword has be used in sub class constructor definition first line.

Example: Unit-5 (Program-04)

/* Create a person class in java which contains name and age. Inherit that class in to student class which contains roll number and course. Create the object of both classes.*/

```
import java.util.*;
class Person
{
    private String name;
    private int age;

    public Person()
    {
        System.out.println("Constructor of Person class");
        name="";
        age=0;
    }
    public Person(String name, int age)
    {
        System.out.println("Parameterised Constructor of Person class");
        this.name=name;
        this.age=age;
    }

    public void display()
    {
        System.out.println("Name : "+name);
```

```
                System.out.println("Age : "+age);
        }
}

class Student extends Person
{
        private String course;
        private int rno;
        public Student()
        {
                System.out.println("Constructor of Student class");
                course="";
                rno=0;
        }

        public Student(String name, int age, String course, int rno)
        {
                System.out.println("Parameterised Constructor of Student class");
                this.course=course;
                this.rno=rno;
        }

        public void putData()
        {
                display();
                System.out.println("Course : "+course);
                System.out.println("Roll Number : "+rno);
        }
}
```

```
class unit05p04
{
    public static void main(String[] args)
    {
        System.out.println("Student Record ");
        Student s1=new Student();
        Student s2=new Student("Amit",17,"MBA",101);
        s1.putData();
        s2.putData();
    }
}
```

Result:

```
Student Record
Constructor of Person Class
Constructor of Student Class
Constructor of Person Class
Parameterised Constructor of Student Class
Name :
Age: 0
Course :
Roll number :  0
Name :
Age: 0
Course : MBA
Roll number :  101
```

Note: * Please note that in above program, s1 object will invoke the constructor of person class and then Student class constructor will be invoked.

* s2 object will invoke the non-parameterized constructor of person class and parameterized constructor of sub class.

If we wish to invoke the parameterized constructor of super class then it may be achieved with the following methodology-

Example: Unit-5 (Program-05)

/* Create a person class in java which contains name and age. Inherit that class in to student class which contains roll number and course. Create the object of both classes.*/

```
import java.util.*;
class Person
{
    private String name;
    private int age;

    public Person()
    {
        System.out.println("Constructor of Person class");
        name="";
        age=0;
    }

    public Person(String name, int age)
    {
        System.out.println("Parameterised Constructor of Person class");
        this.name=name;
        this.age=age;
    }

    public void display()
    {
        System.out.println("Name : "+name);
        System.out.println("Age : "+age);
    }
}
```

```
class Student extends Person
{
    private String course;
    private int rno;

    public Student()
    {
        System.out.println("Constructor of Student class");
        course="";
        rno=0;
    }
    public Student(String name, int age, String course, int rno)
    {
        super(name,age);    //Invoking parameterised constructor of
super class
        System.out.println("Parameterised Constructor of Student class");
        this.course=course;
        this.rno=rno;
    }
    public void putData()
    {
        display();
        System.out.println("Course : "+course);
        System.out.println("Roll Number : "+rno);
    }
}
class unit05p05{
    public static void main(String[] args)
    {
        System.out.println("Student Record ");
        Student s1=new Student();
```

```
        Student s2=new Student("Amit",17,"MBA",101);
        s1.putData();
        s2.putData();
    }
}
```

Result:

Student Record

Constructor of Person Class

Constructor of Student Class

Parameterised Constructor of Person Class

Parameterised Constructor of Student Class

Name :

Age: 0

Course :

Roll number : 0

Name : Amit

Age: 17

Course : MBA

Roll number : 101

Check Your Progress

1. Explain the types of inheritances in java?
2. Compare the visibility modes of java.
3. Explain the sequence in which constructor will be invoked for sub class object wth example.
4. State the output-

```
class A
{
   int i = 10;
```

```
}
class B extends A
{
   int i = 20;
}
public class MainClass
{
   public static void main(String[] args)
   {
      A a = new B();
      System.out.println(a.i);
   }
}
```

5. State the output-

```
class A
{
   String s = "Class A";
}
class B extends A
{
   String s = "Class B";
   {
      System.out.println(super.s);
   }
}
class C extends B
{
   String s = "Class C";
   {
      System.out.println(super.s);
   }
```

```
}
public class MainClass
{
   public static void main(String[] args)
   {
      C c = new C();

      System.out.println(c.s);
   }
}
```

6. Private members of a class are inherited to sub class. True or false?
7. Can a class extend itself?

5.6 METHOD OVERRIDING

There may be situations when super class has one method defined and subclass also has the same method but with different definition. If we have different parameter list than this situation may be resolved with method overloading.

But if we have same signature in both methods i.e. super class and sub class method, than it has be resolved with the concept of overriding.

Overriding is the concept in java which provides the way to redefine the method of super class into sub class. One thing is to make sure that method in both super class and sub class must have the same name and same argument list for overriding.

Now the question arises, which method will be invoked?

It is totally depends on the object which is invoking the method. There may be following situation in object initialization-

- Object declared with super class and memory allocated with super class-

  Super class method will be invoked
- Object declared with super class and memory allocated with sub class-

  Sub class method will be invoked
- Object declared with sub class and memory allocated with super class-

  Not possible. It will give compile time error.
- Object declared with sub class and memory allocated with sub class-

  Sub class method will be invoked.

It may be understand with following table-

Object declaration	Memory Allocation	Method invoke
Super Class	Super Class	Super class method
Super Class	Sub Class	Sub Class method
Sub Class	Super Class	ERROR
Sub Class	Sub Class	Sub Class

Table 5.3

From the above discussion it is clear that object of super class may be instantiated with super class or sub class. But object of sub class may be instantiated with sub class only.

A subclass may override instance methods that it would otherwise inherit from a superclass. Overriding such a method allows the subclass to provide its own implementation of the method. When the method is invoked on an object of the subclass, it is the method implementation in the subclass that is executed. The overridden method in the superclass is not inherited by the subclass.

- The new method definition must have the same method signature.
- A subclass must use the keyword super in order to invoke an overridden method in the superclass.
- An instance method in a subclass cannot override a static method in the superclass.
- A final method cannot be overridden.
- An abstract method requires non-abstract subclasses to override method to provide an implementation.

Example: Unit-5 (Program-06)

/* Create a person class in java which contains name and age. Inherit that class in to student class which contains roll number and course. Create the object of both classes.*/

```
import java.util.*;
class Person
{
    private String name;
    private int age;
```

```
    public Person()
    {
        name="";
        age=0;
    }

    public void display()
    {
        System.out.println("Name : "+name);
        System.out.println("Age : "+age);
    }
}

class Student extends Person
{
    private String course;
    private int rno;
    public Student()
    {
        course="";
        rno=0;
    }
    public void display()
    {
        super.display();
        System.out.println("Course : "+course);
        System.out.println("Roll Number : "+rno);
    }
}
```

```
class unit05p06
{
    public static void main(String[] args)
    {
        Person p;
        Student s;

        p=new Person();
        System.out.println("Super class with super class");
        p.display();
        p=new Student();
        System.out.println("Super class with sub class");
        p.display();
        System.out.println("Sub class with super class");
        //  s=new Person();
        System.out.println("Sub class with sub class");
        s=new Student();
        s.display();
    }
}
```

Result:

```
Super class with super class
Name :
Age: 0
Super class with sub class
Name :
Age: 0
Course :
Roll number :  0
Super class with super class
```

Sub class with sub class

Name :

Age: 0

Course :

Roll number : 0

Check Your Progress

1. What is overriding? Explain with example.
2. Compare overriding with over loading.
3. Can we override static method in Java?
4. Can you prevent overriding a method without using final modifier?
5. Can we override a private method in Java?
6. Can we change return type of method in subclass while overriding?
7. What are rules of method overriding in Java?
8. State the output-

```
class Derived
{
   public void getDetails()
   {
      System.out.println("Derived class");
   }
}
public class Test extends Derived
{
   protected void getDetails()
   {
      System.out.println("Test class");
   }
   public static void main(String[] args)
   {
```

```
        Derived obj = new Test();  // line xyz
        obj.getDetails();
    }
}
```

9. State the output-

```
class Derived
{
    public void getDetails()
    {
        System.out.printf("Derived class ");
    }
}
public class Test extends Derived
{
public void getDetails()
    {
        System.out.printf("Test class ");
        super.getDetails();
    }
    public static void main(String[] args)
    {
        Derived obj = new Test();
        obj.getDetails();
    }
}
```

10. State the output-

```
class Derived
{
    public void getDetails(String temp)
    {
        System.out.println("Derived class " + temp);
```

```
    }
}
public class Test extends Derived
{
    public int getDetails(String temp)
    {
        System.out.println("Test class " + temp);
        return 0;
    }
    public static void main(String[] args)
    {
        Test obj = new Test();
        obj.getDetails("GFG");
    }
}
```

5.7 ABSTRACT CLASSES

An abstract class is a class that is designed to be specifically used as a base class. And no object is created for this base class and no non-static data members of an abstract class can be declared.

Abstract class in Java is similar to interface except that it can contain default method implementation. An abstract class can have an abstract method without body and it can have methods with implementation also.

abstract keyword is used to create a abstract class and method. Abstract class in java can't be instantiated. An abstract class is mostly used to provide a base for subclasses to extend and implement the abstract methods and override or use the implemented methods in abstract class

- abstract keyword is used to create an abstract class in java.
- Abstract class in java can't be instantiated.
- We can use abstract keyword to create an abstract method, an abstract method doesn't have body.
- If a class have abstract methods, then the class should also be abstract using abstract keyword, else it will not compile.

- It's not necessary for an abstract class to have abstract method. We can mark a class as abstract even if it doesn't declare any abstract methods.
- If abstract class doesn't have any method implementation, its better to use interface because java doesn't support multiple class inheritance.
- The subclass of abstract class in java must implement all the abstract methods unless the subclass is also an abstract class.
- All the methods in an interface are implicitly abstract unless the interface methods are static or default.
- Java Abstract class can implement interfaces without even providing the implementation of interface methods.
- Java Abstract class is used to provide common method implementation to all the subclasses or to provide default implementation.
- We can run abstract class in java like any other class if it has main() method.

Example: Unit-5 (Program-07)

/* Create a shape class in java as abstract. Inherit that class into rectangle and circle class to define their drawing methods*/

```
import java.util.*;
abstract class Shape
{
    void draw()
    {}
}
class Rectangle extends Shape
{
    void draw()          // overriding draw( ) method but not mandatory
    {
        System.out.println("drawing rectangle");
    }
}
class Circle extends Shape
{
    void draw()          // overriding draw( ) method
```

```
    {
        System.out.println("drawing circle");
    }
}
class unit05p07
{
    public static void main(String args[])
    {
        Shape s=new Circle();
        s.draw();
        s=new Rectangle();
        s.draw();
    }
}
```

Result :

Drawing Circle

Drawing Rectangle

Abstract Methods :

Abstract methods are those where overriding is mandatory in sub class. In super class, they can't have definition. But in sub class, their definition is mandatory. In super class, abstract keyword is mandatory before method signature. Bu in sub class, it should not be there for definition. One more thing is mandatory that abstract method may be defined only inside the abstract class. On the other side abstract class are not mandatory to have abstract methods.

Example: Unit-5 (Program-08)

/* Create a shape class in java as abstract. Inherit that class into rectangle and circle class to define their drawing methods*/

```
import java.util.*;
abstract class Shape
{
    abstract void draw();    // abstract method with no definition
```

```
}
class Rectangle extends Shape
{
    void draw()          // overriding draw( ) method as it is mandatory
    {
        System.out.println("drawing rectangle");
    }
}
class Circle extends Shape
{
    void draw()          // overriding draw( ) method as it is mandatory
    {
        System.out.println("drawing circle");
    }
}
class unit05p08
{
    public static void main(String args[])
    {
        Shape s=new Circle();
        s.draw();
        s=new Rectangle();
        s.draw();
    }
}
```

Result:

Drawing Circle

Drawing Rectangle

Now we will show the difference between abstract class and non-abstract class.

Abstract class vs. Concrete Class

Sr No.	Abstract Class	Concrete Class
1	An abstract class is declared using abstract modifier.	A concrete class is note declared using abstract modifier.
2	An abstract class cannot be directly instantiated using the new keyword.	A concrete class can be directly instantiated using the new keyword.
3	An abstract class may or may not contain abstract methods.	A concrete class cannot contain an abstract method.
4	An abstract class cannot be declared as final.	A concrete class can be declared as final.
5	Interface implementation is not possible	Interface implementation is possible.

Table 5.4

Now we will show the difference between abstract class and interface.

Abstract Class vs. Interface		
Sr No.	**Abstract class**	**Interface**
1	Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods. Since Java 8, it can have default and static methods also.
2	Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
3	Abstract class can have final, non-final, static and non-static variables.	Interface has only static and final variables.
4	Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class.
5	The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.

6	An abstract class can extend another Java class and implement multiple Java interfaces.	An interface can extend another Java interface only.
7	An abstract class can be extended using keyword "extends".	An interface can be implemented using keyword "implements".
8	A Java abstract class can have class members like private, protected, etc.	Members of a Java interface are public by default.

9	Example:
	public abstract class Shape
	{
	public abstract void draw();
	}
		Example:
		public interface Drawable
		{
		void draw();
		}

Table 5.5

Check Your Progress

1. What is abstract class?
2. Compare abstract class with normal classes.
3. Compare abstract class with interface.
4. State the reason of error-

```
abstract class AbstractClass
{
   abstract void abstractMethod()
   {
      System.out.println("First Method");
   }
}
```

5. Which class is instantiable -

```
abstract class A
{ }
class B extends A
{ }
```

6. Can we write explicit constructors in an abstract class?
7. Can we declare protected methods in an interface?

8. State the output of the program-

```
abstract class A
{
   abstract void firstMethod();
   void secondMethod()
   {
      System.out.println("SECOND");
      firstMethod();
   }
}
 abstract class B extends A
{
   void firstMethod()
   {
      System.out.println("FIRST");
      thirdMethod();
   }
   abstract void thirdMethod();
}
class C extends B
{
   @Override
   void thirdMethod()
   {
      System.out.println("THIRD");
   }
}
public class MainClass
{
   public static void main(String[] args)
   {
      C c = new C();
      c.firstMethod();
      c.secondMethod();
```

```
            c.thirdMethod();
        }
    }
```

5.8 POLYMORPHISM

Polymorphism is an important feature of object oriented programming. It tells the programmer that multiple definition may be defined for one method. The calling of that method will be decided at compile time or runtime. The feature which enables the programmer to set multiple faces for one name, is called polymorphism. Actually polymorphism tells us that binding of method with its call may vary. It may depends on its argument list or memory allocation. If we define multiple definition for one method, than java has to decide which one to invoke.

It is of two types:

Compile-Time Polymorphism

This is also achieved via static or early binding. In early binding method will be binded with its call at the time of the compilation. It is called early or static binding because binding is decided before execution at the time of compilation.

In this case binding depends on the number of parameters or the type of parameters. Method name is same for multiple method definition in this case.

Method overloading and constructor overloading come in this category.

Method overloading is already discussed in this unit.

Runtime Polymorphism

This is also known as dynamic or late binding. In late binding member function bind during run time that means after compilation. This binding is decided with the allocated memory. If the memory is allocated for super class than method of super class is bind. If the memory of sub class is allocated to object than method of sub class will be invoked.

It is also known as overriding. This is already discussed in the unit.

OVERLOADING vs. OVERRIDING		
Criteria	**Overriding**	**Overloading**
Method name	Must be the same	Must be the same
Argument list	Must be the same	Must be different
Return type	Can be the same type or a covariant type	Can be different

throws clause	Must not throw new checked exceptions Can narrow exceptions Thrown	Can be different
Accessibility	Can make it less restrictive but not more restrictive	Can be different
Declaration Context	A method can only be overridden in a subclass not in same class	A method can be overloaded in the same class or in a subclass
Method call resolution	The runtime type of the reference, i.e., the type of the object referenced at runtime, determines which method is selected for execution	At compile time, the declared type of the reference is used to determine which method will be executed at runtime

Table 5.6

Check Your Progress

1. Compare over loading and overriding.
2. What is polymorphism? Explain its types.

5.9 FINAL KEYWORD

final keyword in java is used to make the constants. It mabe used with member, variable, methods and class.

Final keyword with data member or variable :

If this keyword is used with data member or variable, it makes that member as constant. But we have to make sure that values must be assigned at the time of declaration.

```
class calculate
{
    final float PI=3.14f;    // this member is now constant.
    ……
}
```

Final keyword with method :

Final keyword may be used with method. If it is used, in that case that method can't be override in sub class. It restrict the overriding. At times, it is very useful

feature in inheritance. It is the opposite to abstract keyword. Abstract makes the method mandatory to override in sub class but final restricts the overriding.

```
class Student
{
    final void input( )  // this method can't be override in subclass
   {
       ……
    }    …..
}
```

Final keyword with class :

Final keyword may be used with class as well. If it is used, in that case that class can't be inherited in any sub class. It can only be used to create objects. It is the opposite of abstract class. Abstract class can only be inherited but not instantiated. Final class can only be instantiated, it can't be inherited.

```
final class Student    // it will restrict this class for inheritance
{
  …
}
```

Final keyword with class :

In the method definition, we can change the value of parameters. If we want to restrict the changes in parameters, we can achieve this with the help of final modifier before parameter declaration in the method declaration.

The compiler can treat final variables as constants for code optimization purposes. Declaring parameters as final prevents their values from being changed inadvertently. For example: if we wish to compare two objects of student on the basis of their average marks -

```
public boolean compareStudent (final Student s)
{
  …
}
```

// in this method definition s can't be modified

Example: Unit-5 (Program-09)

/* Create a shape class in java as abstract. Inherit that class into rectangle and circle class to define their drawing methods*/

```
import java.util.*;
```

```
abstract class Shape                    // abstract class for inheritance only
{
    abstract void area();   // abstract method for overriding
}
final class Circle extends Shape   // final class, now Circle can't be inherited further
{
    float r;
    final float PI=3.14f;   // constant declaration with final
    float a;

    public Circle(float r)  // parameterised constructor
    {
        this.r=r;
    }
    void area()             // overriding method
    {
        a=PI*r*r;
        System.out.println("Area of circle : "+a);
    }
}
class unit05p09
{
    public static void main(String args[])
    {
        // Shape s= new Shape()              // abstract class so can't be instantiated
        Shape s=new Circle(4.25f);   // memory allocation with sub class only
        s.area();
    }
}
```

Result:

Area of circle : 56.71625

Check Your Progress

1. What is final keyword? Compare final class with abstract class.
2. Compare final method with abstract method.

5.10 SUMMARY

The ability of a class to inherit the properties and characteristics from another class is called **Inheritance.** Inheritance is one of the most important feature of Object Oriented Programming. The class that inherits properties from another class is called Sub class or new or Derived Class. The class whose properties are inherited by sub class is called Super class or old class or Base Class. This unit defines about the inheritance, types of derivation types of Inheritance, how Constructor and Destructor invoke during inheritance.

5.11 EXERCISE

Q. 1. Explain different types of inheritance with block diagram and an example for each.

Q. 2. Explain the importance and role of Access specifier.

Q. 3. What is the difference between friend class and containership?

Q. 4. When we should use containership and inheritance?

Q. 5. Write a program of Constructor in a derived class.

Q. 6. What is virtual base class and abstract class? Explain with help of program.

Q. 7. Does java support multiple inheritances? And why?

Q. 8. Write a program in java to implement the multilevel inheritance.

UNIT-6 PACKAGES AND INTERFACES

Structure :

6.1 OBJECTIVE

Using the library is very important part of programming so is the user defined library. This unit is focused on predefined packages of java as well as user defined packages of java. In this unit we will came to know how to create new hierarchy of packages in java. Setting the class path and using that path, is also elaborated in this unit.

To implement the multiple inheritance, interfaces will also be explained in this unit and those will be compared with classes of java.

6.2 INTRODUCTION

Java is one of the object oriented programming languages. It is also used for developing very lengthy and complicated projects. So there is a team involved in the small project of java. In large project multiple teams are involved. That sort of situation is managed via packages in java. Some packages are predefined in java library. Some packages we may define. This un it will cover packages. Besides that we will also discuss the interfaces.

6.3 PACKAGE

Java is one of the most popular and widely used technology for project development from small level to large level. So there may be multiple teams are involved in one large project.

Package is actually a collection of classes, interfaces and packages etc. Packages may be nested to any level.

Package in java can be categorized in two form-

- built-in package and
- user-defined package.

Advantages / Uses of Java Packages :

- ***Maintenance***: Java packages are used for proper maintenance. If any developer newly joined a company, he can easily reach to files needed.
- ***Reusability***: We can place the common code in a common folder so that everybody can check that folder and use it whenever needed.
- ***Name conflict***: Packages help to resolve the naming conflict between the two classes with the same name. Assume that there are two classes with the same name Student.java. Each class will be stored in its own packages such as stdPack1 and stdPack2 without having any conflict of names.
- ***Organized***: It also helps in organizing the files within our project.
- ***Access Protection***: A package provides access protection. It can be used to provide visibility control. The members of the class can be defined in such a manner that they will be visible only to elements of that package.

Predefined packages in java :

There are 6 predefined packages in java. There are many but in core java, there are 6 basic packages in java library.

PACKAGE NAME	EXAMPLE CLASSES	FUNCTIONALITY (PURPOSE)
java.lang	System, String, Object, Thread, Exception etc.	These classes are indispensable for every Java program. For this reason, JVM automatically imports it.
java.util	These are called as utility (service) classes and are used very frequently in coding.	We generally use Scanner class from this package.
java.io	FileInputStream, FileOutputStream, FileReader, FileWriter, RandomAccessFile, BufferedReader, BufferedWriter etc.	These classes are used in all I/O operations including keyboard input.

java.net	URL, ServerSocket, Socket, DatagramPacket, DatagramSocket etc.	Useful for writing socket programming (LAN communication).
java.applet	AppletContext, Applet, AudioStub, AudioClip etc	Required for developing applets that participate on client-side in Internet (Web) programming.
java.awt	Button, Choice, TextField, Frame, List, Checkbox etc.	Essential for developing GUI applications.

Table 6.1

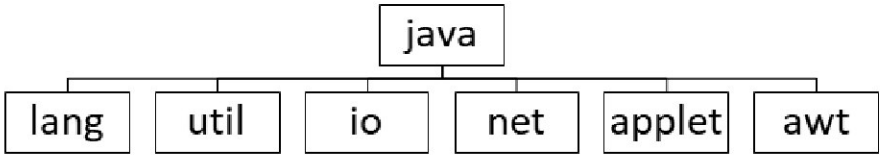


Figure 6.1

Using built-in java package :

Predefined packages are easy to use. We just have to find the suitable package for our requirement and import it. To import any package, the syntax is

```
import packageName.*;              // to import all members of the package
import packageName.MemberName;     // to import only one member (class) of the package.
```

e.g.

```
import java.util.*;          // imports all classes and other members of util packages
import java.util.Scanner; // imports only Scanner class of the package.
```

Once imported, we may use the imported members anywhere in that program for any number of times.

Defining user defined package (New Package) :

Java provides the facility to create new packages as well. This resolves the problem of large projects to an extent. There may be multiple classes required in one project. For this we need a structure of classes. This user defined package structure may be assumed as the structure of folders in windows. It means we may subgroup the classes as well like folders.

One thing is very important that, we have to first analyse and decide the hierarchy of user defined packages to group classes.

For example, we are defining few classes for college management system in one packages.

The structure is like this-

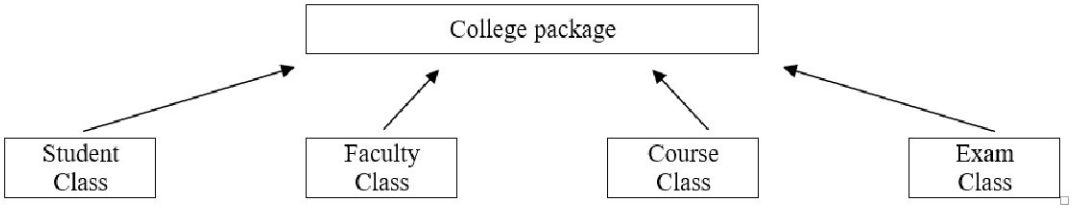


Figure 6.2

In the above example we have to define 4 classes – Student, Faculty, Course and Exam in the package called College.

There are few steps for defining and using user defined packages-

Step 1: Defining Package :

To define the package we have to create the folder in windows with the name of the package. Make sure that package name and folder name is same (it is case sensitive).

Step 2: Defining class inside the Package folder :

To define the class, we have to define the class in the same way like we have done earlier but we just have to add one line at the top of program-

```
package packageName;
class ClassName
{
   ……
}
```

e.g.

```
package College;
class Student
{
    …..
}
```

We have to compile the all classes in the same way like we have done earlier with the help of javac command followed by classname.

Step 3: importing package in destination program code :

Once the package is created and we have defined required classes inside that package folder, we may import the classes in any program other than that package folder.

Importing the user defined package is same as for library package. We just have to find the suitable package for our Defining Package:

To import any user defined package, the syntax is

```
import packageName.*;              // to import all members of the package
import packageName.MemberName;     // to import only one member (class) of the package.
```

e.g.

```
import College.*;          // imports all classes and other members of College packages
import College.Student;    // imports only student class of the package College.
```

Once imported, we may use the imported members anywhere in that program for any number of times.

Step 4: Setting classpath for user defined package :

Once the package is created and we have defined required classes inside that package folder. But how that java compiler will bind those user defined classes with the current class. Compiler just has the path of built-in packages. For user defined packages, we have to define the path where the folder of package is stored.

The name of the classpath variable is to be defined in the same way like it has been done for path variable.

Java steps to set the classpath-

- Right click on my computer icon
- Select properties
- Select "Advance System Settings"
- Select "Environment Variables." button in advanced tab
- Select "new" button
- Specify the "classpath" in the variable name box
- Specify "C:\.....\College" or the path of your package in value box
- Click "OK" on all dialogue boxes opened you in this process.

Java steps to set the via command prompt-

Run the command

```
>    set classpath=%classpath%;C:\....\Student
```

"C:\....\Student" is the path of your package folder.

Please note that classpath variable from the dialogue box is permanent but classpath via command prompt is temporarily, if we close the prompt, we have to define it again.

Example: Unit-6 (Program-01)

/* Create a student class in java which contains name and roll number in the package College.*/

```
package College;            // to declare this class in the package College.
import java.util.*;
class Student
{
    private int rno;
    private String name;

    public void input( )
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter Student's name : ");
        name=sc.next();
        System.out.print("Enter Student's Roll number : ");
        rno=sc.nextInt();
    }
    public void display( )
    {
        System.out.println("\nStudent's Name : "+name);
        System.out.println("Student's Roll number : "+rno);
    }
}
```

Result:

No result as this program can't be executed. It may be compiled only.

Note: * Please make sure that this file is saved in the folder called "College".

* We have to assign the location of this folder into classpath variable.

* This program is to be compiled but not executed. Actually it can't be run as there is no main().

* This class can be used in other program by importing this package.

To import the above Student class in any other program, this has to be imported.

Example: Unit-6 (Program-02)

/* Create a student class in java which contains name and roll number in the package College.*/

```
import College.*;
import java.util.*;
class unit06p02
{
    public static void main(String[] args)
    {
            Student s=new Student();
            s.input();
            s.display();
    }
}
```

Result :

Enter Student's Name: Ankit

Enter Student's Roll number : 101

Student's Name: Ankit

Student's Roll number : 101

Nested packages :

We may define nested packages as well. But we need not to define the classpath for nested packages.

To declare nested package at the top of program use the syntax-

package MainPackageName.SubPackageName;

```
class ClassName
{
   ....
}
```

For example we have to define the nested package with name Exam which contains the class ClassTest1-

```
package College.Exam;
class ClassTest1
{
   ....
}
```

Main points to remember about packages-

- While importing another package, package declaration must be the first statement and followed by package import.
- A class can have only one package statement but it can be more than one import package statements.
- import can be written multiple times after the package statement and before the class statement.
- You must declare the package with root folder name (No subfolder name) and last file name must be class name with a semicolon.
- When you import, does mean that memory is allocated. It just gives the path to reach the file.
- import com.scientecheasy.state.cityinfo.dhanbad; is always better than import com.scientecheasy.state.cityinfo.*;

Check Your Progress

1. What is package in java? Explain with example.
2. Compare java packages and C header file.
3. What are the built-in java packages? Describe their uses.
4. Write the steps to set the classpath.

6.4 ACCESSIBILITY OF PACKAGES USING PACKAGE MEMBERS

In java, there are 4 types of modifiers- public, private, protected, friendly. Every modifier is different in its working. We have already discussed public, private and protected earlier. Now we will discuss friendly/default modifier.

If we have used packages, we may use one class for inheritance or for object creation. These operations may be performed in the same program or in different program. Different program may be in same folder or in different folder. If the program is in same folder than it may be accessed directly. If this class is in the different folder than we have to take care of the basics of packages because different folder means package in java. For that we have to set the class path first.

As a programmer we have to take care of the visibility of the members of the class well. All the members are accessible in same class, so there is no question

Modifer	Same Packages	Different Packages
private	No inherit, no access by object	No inherit, no access by object
protected	inherit, access by object	inherit, no access by object
default	inherit, access by object	No inherit, no access by object
public	inherit, access by object	inherit, access by object

Table 6.2

It is clear from the above table that if we are using packages than protected, public and default are different otherwise there is no difference between them.

Check Your Progress

1. Define the rules for accessing members if we are using packages.
2. What are the advantages of a java package?
3. What is the base class of all classes?
4. Which of these is a mechanism for naming and visibility control of a class and its content?
   a) Object
   b) Packages
   c) Interfaces
   d) None of the Mentioned
5. State the output-

```
package pkg;
class display
{
  int x;
  void show()
            {
              if (x > 1)
```

```
                System.out.print(x + " ");
            }
        }
        class packages
        {
            public static void main(String args[])
            {
                display[] arr=new display[3];
                for(int i=0;i<3;i++)
                    arr[i]=new display();
                arr[0].x = 0;
                arr[1].x = 1;
                arr[2].x = 2;
                for (int i = 0; i < 3; ++i)
                    arr[i].show();
            }
        }
```

6. Which package is always imported by default?

6.5 INTERFACES

Interface is one of the user defined data type in java. It is the collection of abstract methods. In interface methods can't have definition.

An interface in java is a blueprint of a class. It has static constants and abstract methods. The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Java Interface also represents the IS-A relationship. It cannot be instantiated just like the abstract class. Since Java 8, we can have default and static methods in an interface. Since Java 9, we can have private methods in an interface. Interface supports inheritance concept. It additionally supports multiple inheritance in java.

Interface Declaration :

Interface can be declared with the help of interface keyword. It provides total abstraction, it means we have to declare the methods only, no definition is to be provided.

```
interface <InterfaceName>
{
   // declare constant fields
   // declare methods that are abstract by default
}
```

e.g.

```
interface Student
{
  void input();
  void display();
}
```

Class Vs. Interface		
Sr No.	**Class**	**Interface**
1	Supports only multilevel and hierarchical inheritances but not multiple inheritance	Supports all types of inheritance – multilevel, hierarchical and multiple
2	"extends" keyword should be used to inherit	"implements" keyword should be used to inherit
3	Should contain only concrete methods (methods with body)	Should contain only abstract methods (methods without body)
4	The methods can be of any access specifier (all the four types)	The access specifier must be public only
5	Methods can be final.	Methods should not be final.
6	Data members can be private	Data members are public only
7	Data members can be final or not	Data members can be final only
8	Can have constructors	Cannot have constructors
9	Can have main() method	Cannot have main() method as main() is a concrete method

Table 6.3

Check Your Progress

1. What is interface? Explain its uses.
2. Compare interface with class.
3. Compare interface with abstract class.

6.6 IMPLEMENTING INTERFACES

Class may inherit other class. Interface may inherit other interface. But in heritance there may be only one inheritance with extends keyword. Interface may be inherited with implements keyword. With implements keyword, we may implement multiple interface separated with comma.

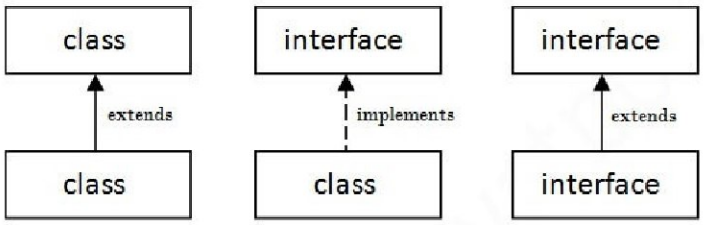


Figure 6.3

Single Inheritance via interface extends

```
interface <SuperInterfaceName>
{
   // declare constant fields
   // declare methods that are abstract by default
}

interface <SubInterfaceName> extends SuperInterfaceName
{
   // declare constant fields
   // declare methods that are abstract by default
}
```

Single Inheritance via interface extends

```
interface <SuperInterfaceName>
{
   // declare constant fields
   // declare methods that are abstract by default
```

```
}
class <SubClassName> extends SuperInterfaceName
{
   // ClassMembers
   // ClassMethods
   //  Implementing interface methods
```

Example: Unit-6 (Program-03)

/* Create a shape interface and inherit that interface into Circle class and create its instance to implement the area of circle*/

```
interface Shape
{
 float PI=3.14f;
 void area();
}
interface Circle extends Shape          // Shape interface extends into interface
{
 void display();
}
class CalculateArea implements Circle // Circle interface implements into class
{
    float r,a;
    public CalculateArea(float r)
    {
        this.r=r;
    }
    public void area()
    {
        a=PI*r*r;
    }

    public void display()
    {
```

```
                System.out.println("Area of circle="+a);
        }
    }

    class unit06p03
    {
        public static void main(String[] args)
        {
                CalculateArea obj;
                obj=new CalculateArea(2.5f);
                obj.area();
                obj.display();
        }
    }
```

Result:

Area of circle=19.625

Interface implements into class :

Example: Unit-6 (Program-04)

/* Create a shape interface and inherit that interface into Circle class and create its instance to implement the area of circle*/

```
interface Shape
{
 float PI=3.14f;
 void area();
}
class Circle implements Shape          // Shape interface implements into class
{
    float r,a;
    public Circle(float r)
    {
            this.r=r;
```

```
    }

    public void area()
    {
        a=PI*r*r;
    }

    public void display()
    {
        System.out.println("Area of circle="+a);
    }
}
class unit06p04
{
    public static void main(String[] args)
    {
        Circle obj;
        obj=new Circle(2.5f);
        obj.area();
        obj.display();
    }
}
```

Result:

Area of circle=19.625

Please note that :

- Class may extends other class (at most one)
- Interface may extends other interface (may be multiple)
- Class may implements other interface (may be multiple)
- Class can't extends interface
- Interface can't implement other interface

Check Your Progress

1. What is the procedure to implement the interfaces?
2. One interface can implements other interface. True | False
3. Create an interface Shape in java which declares method volume(). Inherit this interface into cube, cuboid and circle class. Override the method volume() in all classes. Create one reference of Shape interface and one object of each sub classes. Input the values in objects. Now assign the object reference of each class one by one into reference of interface and call the method volume().

6.7 INTERFACE AND ABSTRACT CLASSES

Interface are somehow similar to abstract class. Although there are several differences between them-

Abstract Class Vs. Interface

Sr No.	Abstract Class	Interface
1	Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods. Since Java 8, it can have default and static methods also.
2	Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
3	Abstract class can have final, non-final, static and non-static variables.	Interface has only static and final variables.
4	Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class.
5	The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.
6	An abstract class can extend another Java class and implement multiple Java interfaces.	An interface can extend another Java interface only.
7	An abstract class can be extended using keyword "extends".	An interface can be implemented using keyword "implements".

8	A Java abstract class can have class members like private, protected, etc.	Members of a Java interface are public by default.

9	Example:
	public abstract class Shape
	{
	public abstract void draw();
	}
		Example:
		public interface Drawable
		{
		void draw();
		}

Table 6.4

Similarities between interface and abstract class:

There are few similarities between interface and abstract class.

- Both are user defined data types.
- Both can contain data members and methods.
- Both supports inheritance.
- Both supports overriding.
- Both can't be instantiated. It means, we can create the object of neither interface nor abstract class. We can only create reference and memory has to be allocated for their sub classes.
- Interface may extend other interface while class may extends other class. Interface can't implements other interface and class can't implements other class. Although Class may implement interfaces.

Check Your Progress

1. Compare abstract class and interface with example
2. Give some common characteristics between interface and abstract class.
3. Abstract class can implements other abstract class. True | False
4. Create an abstract class Shape in java which contains abstract method volume(). Inherit this class into cube, cuboid and circle class. Override the method volume() in all classes. Create one reference of Shape class and one object of each sub classes. Input the values in objects. Now assign the object reference of each class one by one into reference of Shape class and call the method volume().

6.8 EXTENDS AND IMPLEMENTS TOGETHER

One class in java may extends and implements at a time. Although extends may be done for one class but implements may use multiple interfaces. It supports multiple inheritance.

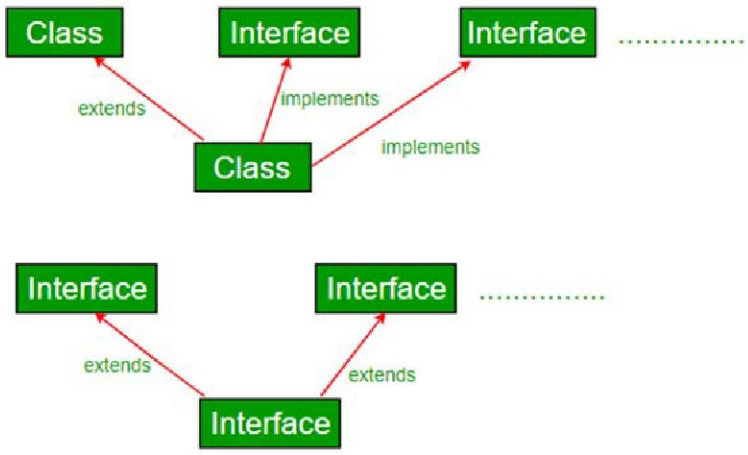


Figure 6.4

One interface may implements multiple interfaces. For example-

Example: Unit-6 (Program-05)

/* Create an interface interfaceA and interfaceB. Implement those two interfaces into one class.*/

```
interface interfaceA
{
   void m1();
}
 interface interfaceB
{
   void m2();
}
class Sample implements interfaceA, interfaceB
{
   @Override
   public void m1()
   {
      System.out.println("Welcome: inside the method m1");
   }
   @Override
   public void m2()
   {
```

```
        System.out.println("Welcome: inside the method m2");
    }
}
class unit06p05
{
    public static void main(String[] args)
    {
        // calling the method implemented within the class.
            Sample obj=new Sample();
        obj.m1();
            obj.m2();
    }
}
```

Result:

Welcome: inside the method m1

Welcome: inside the method m2

Default method in interface (jdk 8 feature) :

Interface may also define method from jdk 1.8. Such methods are called default methods in interfaces. Such methods must have default keyword before method declaration. Default methods must have definition in interface. Second important thing about default method is that they can't be override in sub classes. If we try to do so, they will produce error.

For example :

Example: Unit-6 (Program-06)

/* Create an interface Drawable. Declare one method draw() and define one msg() method as default method inside the interface. Implement the interface into Rectangle and Circle class and override the method of interface. */

```
interface Drawable
{
    void draw();
    default void msg() // cann't override in sub class
    {
        System.out.println("default method");
    }
```

```
}
class Rectangle implements Drawable
{
    public void draw()
    {
        System.out.println("drawing rectangle");
    }
}
class Circle implements Drawable
{
    public void draw()
    {
        System.out.println("drawing circle");
    }
}
class unit06p06
{
    public static void main(String args[])
    {
        Drawable d;
        d=new Rectangle();
        d.draw();
        d.msg();
        d=new Circle();
        d.draw();
        d.msg();
    }
}
```

Result:

drawing rectangle
default method
drawing circle
default method

static method in interface (jdk 8 feature) :

Interface may also be defined as static in interface from jdk 1.8. Such methods are called static methods in interfaces. Such methods must have static keyword before method declaration. Default methods must have definition in interface. Second important thing about default method is that they can't be override in sub classes. If we try to do so, they will produce error. Besides this static methods are to be invoked via interface name like we do in static method of class.

For example :

Example: Unit-6 (Program-07)

/* Create an interface Drawable. Declare one method draw() and define one msg() method as default method inside the interface. Implement the interface into Rectangle and Circle class and override the method of interface. */

```
interface Drawable
{
    void draw();
    static int cube(int x)        // static method in interface with definition
    {
        return x*x*x;
    }
}
class Rectangle implements Drawable
{
    public void draw()
    {
        System.out.println("drawing rectangle");
    }
}
 class unit06p07
{
    public static void main(String args[])
    {
        Drawable d=new Rectangle();
        d.draw();
        int x=Drawable.cube(3);
```

```
            System.out.println("Volume of cube : "+x);
        }
    }
```

Result:

drawing rectangle

Volume of cue : 27

Check Your Progress

1. How multiple inheritance is implements in java via interface explain with example.
2. What is static method in interface? Explain its conditions with example.
3. What is default method in interface in java? Explain its condition with example.
4. State the output-

```
interface Drawable
{
        void draw();
}
class Rectangle implements Drawable
{
        public void draw()
        {System.out.println("drawing rectangle");}
}
class Circle implements Drawable
{
        public void draw()
        {System.out.println("drawing circle");}
}
class TestInterface1
{
        public static void main(String args[])
        {
                Drawable d=new Circle();
```

```
                d.draw();
            }
        }
```

5. Define the bank interface which contains method called RateofInterest(). Implement that interface into class called SBI and PNB. Override the method into classes and create the object of sub classes.

6.9 SUMMARY

Java is one of the most popular object oriented programming language. It supports almost all features of Object Orientation. Java supports single, multilevel and hierarchical inheritance via classes. Multiple inheritance may be implemented via interfaces in java. Java version jdk1.8 introduces default method and static method in interface with definition and no overriding condition.

6.10 EXERCISE

Q. 1. What is package? Describe the benefits of package.

Q. 2. What are the built-in packages? Make a list of 10 methods generally used from build-in packages.

Q. 3. Which method is implicitly imported in java programs?

Q. 4. What is abstract method in Java?

Q. 5. Can abstract class have constructors in Java?

Q. 6. Can abstract class implements interface in Java? Does they require to implement all methods?

Q. 7. Can abstract class be final in Java?

Q. 8. Can you create instance of abstract class?

Q. 9. Can abstract class contains main method in Java?

Create the person interface which contains input() and display() methods. Implement this interface into employee and student class. Employee class contains name, id and salary as data member. Student class contains name, roll number and course as data members. Define the object of both classes.

UNIT-7 EXCEPTIONS HANDLING

Structure:

7.1 OBJECTIVE

Exception handling is very important task as our program should not be terminated without the consent of user. Whenever end user wishes to close the program, it should only be stops in that condition only. In this unit, we will understand the exceptions, its types and the various methods to handle the exceptions.

7.2 INTRODUCTION

Java is one of the Object oriented programming languages. It supports the compile time language translation. If we are developing the programs, errors may arise. Errors may arise at compile time or at runtime. Runtime errors are called exceptions. Exceptions stops us from executing the program smoothly because as exception occurs, our program gets terminated instantly. However, if we know the type and situation of exception, we may try to handle the exception so that program may not terminates.

7.3 EXCEPTIONS

Our program get executed in the way we have coded that. But there may be some situations, in which the control of flow of execution gets changed. It means there may be situations, when execution is done in the way that is other than normal course of action. For example, we have to input the age of any person, but end user specify the age in text form rather than numerical form. This is such a situation in which, end user is correct at his side but technically that should not be in text form but should be in numerical form.

When such a situation happens, that is known as exception. Exceptions are not compile time errors, these are runtime errors. These runtime errors may occur in program at one time but it is not necessary that those errors occur at every time we run the program.

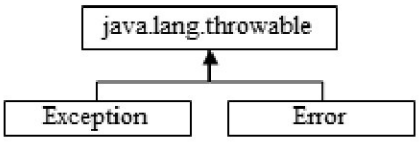


Figure 7.1

Simple we may say that Exception is the action taken by user that is other than normal course of action.

An exception signals the occurrence of some unexpected error situation during execution. Exceptions in Java are objects. All exceptions are derived from the java.lang.Throwable class. The Throwable class provides a String variable that can be set by the subclasses to provide a detail message. The purpose of the detail message is to provide more information about the actual exception. By several sources exception may be thrown.

There are two sources in major:

- Either it is the JVM that is responsible for throwing the throwable
- That the throwable is explicitly thrown programmatically by the code in the application or any API used by the application.

The Exception Class

The class Exception represents exceptions that a program would normally want to catch.

The Error Class

An Error indicates serious problems that a reasonable application should not try to catch and are usually irrecoverable. Most such errors are signaled by the JVM.

Example: Unit-7 (Program-01)

/* Write a program in java to divide two integers.*/

```
import java.util.*;
class unit07p01
{
    public static void main(String args[])
    {
        int a,b,c;
        Scanner sc=new Scanner(System.in);
```

```
            System.out.println("Enter two numbers");
            a=sc.nextInt();
            b=sc.nextInt();
            c=a/b;
            System.out.println("Result="+c);
        }
    }
```

Result :

```
Enter two numbers
12
4
Result=3
```

Explanation: above program will accept two numbers and produce the result accordingly.

Suppose we change the input values of the above program with 12 and 0.

Result (unit07p01) :

```
Enter two numbers
12
0
Exception in thread "main" java.lang.ArithmeticException: / by zero
            at unit07p01.main (unit07p01.java:11)
```

So if we are giving 0 as denominator in the program, it is not possible but for values other than 0 it is possible. And in case of zero it will produce the Exception called ArithmeticException and program will be terminated without excuting rest of the program.

Check Your Progress

1. What is exception? Describe the sources of exceptions in java.
2. Explain the types of exceptions in java.
3. Compare compile time and runtime errors.
4. Write a program in java to accept the 10 numbers in array and try to print the 11 elements. Check weather exception is generated by JVM or not. If yes than which exception is generated.

7.4 HANDLING OF EXCEPTION

Java is one of the most popular object oriented programming language. It supports compile time error detection and display. But at times we have to be sure that program does not stops working at runtime. To achieve this, we may use exception handling. It restrict the closing of program from certain pre-decided exceptions. Actually exceptions are also called runtime errors. If we does not handle the exception, software may be terminated from execution mode.

There are some important points to be remembered before we go through the exception handling.

- **Try**: The try block establishes a context for exception handling.
- **Catch**: Only an exit from a try block resulting from an exception can transfer control to a catch block, if the exception is assignable to the parameter in the catch block.
- **Finally**: If the try block is executed, then the finally block is guaranteed to be executed, regardless of whether any catch block was executed.
- **throw Statement**: throw statement is used to explicitly throw an exception. If it is in the try block then it transfer the control to respective catch block. If it is used in the throws method, then it returns the control to calling method.
- **throws clause**: throws clause can be specified in the method header. Explicitly throwing an exception with throws statement, makes that exception checked for that method. Throws statement does not handle exception, it just propagate the exception.

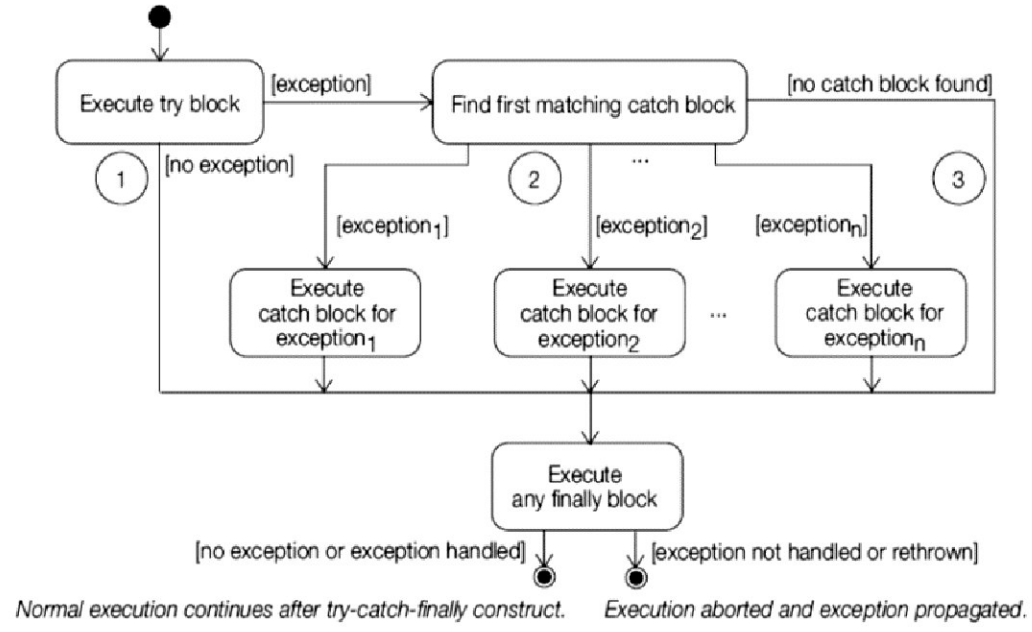


Figure 7.2

The syntax is

```
try
{
        statements(s);
}
catch (ExceptionClass ex)
{
        statements(s);
}
finally
{
        statements(s);
}
```

In the above syntax, there may be multiple catch statements and finally is optional.

try vs. catch		
S.No.	**catch**	**finally**
1	There may be multi catch blocks.	There may at most one finally block.
2	Catch block contains parameter	Finally does not have any parameter
3	In case of multiple catch block, all are not executed, they are compared with propagated exception one by one like switch statement.	There are no multiple finally blocks so no question arise for comparison.
4	Catch block is optional for exception as it will execute only when propagated exception is match with its parameter exception.	If finally block is given with try block, then it will definitely execute irrespective of exception generation.

Table 7.1

Please note that there must be at least one catch or finally block with any try block.

throw vs. throws		
S.No.	**throw**	**throws**
1	Throw is the statement which manually propagate the exception.	Throws statement is used to indicate that containing method may propagate the exception.
2	It is an independent statement.	It is at the end of method name line.
3	One throw statement may propagate only one exception at a time.	One throws may indicate multiple exceptions propagation.
4	if exception is generated in one method, throw statement allows, exception to be handled in the same method.	If exception is generated in the method with throws, exception must be handled in its method, which call that method.

Table 7.2

Please note that there is a basic difference in the purpose of throw and throws. Both have different purpose so both may be used in one program for exception handling.

Example: Unit-7 (Program-02)

/* Write a program in java to divide two integers. Check the exception if denominator is 0*/

```
import java.util.*;
class unit07p02
{
    public static void main(String args[])
    {
        int a,b,c;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter two numbers");
        a=sc.nextInt();
        b=sc.nextInt();
        try
        {
```

```
                c=a/b;
            }
            catch(ArithmeticException ex)
            {
                System.out.println("Denominator should not be zero");
                c=0;
            }
            System.out.println("Result="+c);
        }
    }
```

Result:

```
Enter two numbers
12
0
Denominator should not be zero
Result=0
```

Please note that even if we input 0 as denominator, JVM will not produce the exception as we have checked the ArithemticException in catch block. If denominator is 0 than ArithmeticException will be generated in the try block and it will be thrown to catch block. Catch block contains ArithemticException, so program will not be terminated and will continue as a normal program.

throws keyword is used to indicate that current method may generate the exception, there may be multiple exception names. If exception arises, control is transferred to the location from where it is called. Throws keyword will not handle the exception. Moreover if exception arises, it will not execute the remaining method as well.

Example: Unit-7 (Program-03)

/* Write a program in java to divide two integers. Check the exception if denominator is 0*/

```
import java.util.*;
class unit07p03
{
    public static void main(String args[]) throws ArithmeticException
    {
```

```
            int a,b,c;
            Scanner sc=new Scanner(System.in);
            System.out.println("Enter two numbers");
            a=sc.nextInt();
            b=sc.nextInt();
            c=a/b;
            System.out.println("Result="+c);
        }
    }
```

Result:

```
Enter two numbers
12
0
Exception in thread "main" java.lang.ArithmeticException: / by zero
            at unit07p01.main (unit07p01.java:11)
```

Check Your Progress

1. Compare try block with catch block.
2. Compare catch block with finally block.
3. Compare throw and throws.
4. What are main classes in exception handling? Explain them with example.
5. Write a program in java to accept the 10 numbers in array and try to print the 11 elements. Check and handle the exception if array index number is not a valid index number.

7.5 TYPES OF EXCEPTIONS

There are different exceptions those may arise at runtime. For the programmer, some exceptions are mandatory to handle while others are not mandatory to handle. So on this basis Java contains two types of exceptions: Checked exception and unchecked exception.

Checked Exceptions :

Checked exceptions are those exceptions which must be dealt. The method must either catch the exception and take the appropriate action, or pass the exception on to its caller.

There are some checked exception :

IOException

FileNotFoundException

ParseException

ClassNotFoundException

CloneNotSupportedException

InstantiationException

InterruptedException

NoSuchMethodException

NoSuchFieldException

Unchecked Exceptions:

Unchecked Exceptions are those exceptions which a method is not obliged to deal with. They are either irrecoverable and the program should not attempt to deal with them.

There are some unchecked exceptions-

ArrayIndexOutOfBoundsException

ClassCastException

IllegalArgumentException

IllegalStateException

NullPointerException

NumberFormatException

ExceptionInInitializerError

StackOverflowError

NoClassDefFoundError

Example: Unit-7 (Program-04)

/* Write a program in java to divide two integers. Input the values with InputStreamReader*/

```
import java.io.*;
class unit07p04
```

```
{
    public static void main(String args[])
    {
        int a,b,c;
        InputStreamReader isr=new InputStreamReader(System.in);
        BufferedReader br=new BufferedReader(isr);
        System.out.println("Enter two numbers");
        a=Integer.parseInt(br.readLine());
        b=Integer.parseInt(br.readLine());
        c=a/b;
        System.out.println("Result="+c);
    }
}
```

Result :

```
unit07p04.java:10: error: unreported exception IOException; must be caught or declared to be thrown
        a=Integer.parseInt(br.readLine());
unit07p04.java:11: error: unreported exception IOException; must be caught or declared to be thrown
        b=Integer.parseInt(br.readLine());
```

Above program will generate IOException at compile time because IOException is being generated by readLine() method which is checked Exception here. So it must be checked bt throws or try catch block.

Example: Unit-7 (Program-05)

/* Write a program in java to divide two integers. Input the values with InputStreamReader*/

```
import java.io.*;
class unit07p05
{
    public static void main(String args[]) throws IOException
    {
        int a,b,c;
        InputStreamReader isr=new InputStreamReader(System.in);
        BufferedReader br=new BufferedReader(isr);
```

```
            System.out.println("Enter two numbers");
            a=Integer.parseInt(br.readLine());
            b=Integer.parseInt(br.readLine());
            c=a/b;
            System.out.println("Result="+c);
        }
    }
```

Result:

```
Enter two numbers
12
0
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at unit07p05.main(unit07p05.java:11)
```

Above program will remove compile time error but exception is not being handled here. So program will be terminated. We must handle it with try catch clock for handling.

Example: Unit-7 (Program-06)

/* Write a program in java to divide two integers. Input the values with InputStreamReader*/

```
import java.io.*;
class unit07p06
{
    public static void main(String args[])
    {
        int a,b,c=0;
        InputStreamReader isr=new InputStreamReader(System.in);
        BufferedReader br=new BufferedReader(isr);
        System.out.println("Enter two numbers");
        try
        {
            a=Integer.parseInt(br.readLine());
            b=Integer.parseInt(br.readLine());
            c=a/b;
```

```
            }
            catch(IOException ex)
            {
                System.out.println("Illegal Input");
            }
            catch(ArithmeticException ex)
            {
                System.out.println("Denominator can not be zero");
                c=0;
            }
            System.out.println("Result="+c);
        }
    }
```

Result:

```
Enter two numbers
12
0
Denominator can not be zero
Result=0
```

Check Your Progress

1. What are the types of exceptions?
2. Differentiate between checked and unchecked exceptions.
3. Write a program to show the IOException as checked exception in student class input method. Student class contains his name, age and rollnumber as data member. Besides input() method it also contains display method.
4. How we can manage the checked exceptions? Explain with example.

7.6 THROWING EXCEPTION

There are time when JVM generates the exceptions at runtime. If exception is generated, it may be handled.

We may also generate the exception, before JVM generates it. Although it is not much beneficial for predefined exceptions but it is mandatory for user defined

exceptions. If we wish to generate the exception, it is known as throwing exception.

Exception may be throw with throw statement. We must know which exception we have to throw. There is a class name for each predefined exception.

The syntax is :

```
throw ExceptionObject;                          // predefined object
or
throw new ExceptionClassName( );                // anonymous object
```

We may create the object of the exception class, we want to throw. We may also create the anonymous object of that class for exception generation.

Example: Unit-7 (Program-07)

/* Write a program in java to divide two integers. Check the exception if denominator is 0*/

```
import java.util.*;
class unit07p07
{
    public static void main(String args[])
    {
        int a,b,c;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter two numbers");
        a=sc.nextInt();
        try
        {
            b=sc.nextInt();
            if(b==0)
                throw new ArithmeticException(); // anonymous
Object
            c=a/b;
        }
        catch(ArithmeticException ex)
        {
            System.out.println("Denominator should not be zero");
            c=0;
```

```
            }
            System.out.println("Result="+c);
        }
    }
```

Result :

```
Enter two numbers
12
0
Denominator should not be zero
Result=0
```

We may also create the object of the Exception class first and then throw it.

Example: Unit-7 (Program-08)

/* Write a program in java to divide two integers. Check the exception if denominator is 0*/

```
import java.util.*;
class unit07p08
{
    public static void main(String args[])
    {
        int a,b,c;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter two numbers");
        a=sc.nextInt();
        try
        {
            b=sc.nextInt();
            if(b==0)
            {
                ArithmeticException aex=new ArithmeticException();
                throw aex;
            }
            c=a/b;
```

```
            }
            catch(ArithmeticException ex)
            {
                System.out.println("Denominator should not be zero");
                c=0;
            }
            System.out.println("Result="+c);
        }
    }
```

Result:

Enter two numbers

12

0

Denominator should not be zero

Result=0

Check Your Progress

1. How exceptions are managed at runtime once they are propagated.
2. Compare throwing exception by JVM and by throw statement.
3. Write a program to perform the linear search on 10 integer array elements. Manage the invalid array index number exception at runtime during the linear search.
4. Write a program to manage the NullPointerException in a program. This program is used to store and display the employee record. Employee records contains employee name, age, employee id, salary and designation.

7.7 WRITING EXCEPTION SUBCLASSES

Till now we have gone through the exceptions which are defined in java compiler. But there are situation when we need to define the exceptions which are not defined in java compiler.

Java as a object oriented programming language provides the facility to create the new exception which are not defined in the compiler. These exception may be related to any problem of our programming. In fact exception handling is one of the most powerful tool to handle the user interaction as user is not the

programmer, he may use the software in any format so exceptions are likely to be generated and as we know once exception is generated, program will terminate without the consent of the user.

So to remove the abnormal termination of our program when used by end user, we must manage all possible exceptions.

New exceptions are usually defined to provide fine-grained categorization of error situations. New exceptions can either extend the Exception class directly or one of its checked subclasses.

We must create the class for managing user defined exception first then we may handle that exception. Exception or its subclass is mandatory to be extended in the class.

Syntax for defining exception class is:

```
class ExceptionClassName extends Exception
{
    :
    :
}
```

Once we extend Exception class in any class, it indicates the java compiler that this class is purposely used for user defined exception in the programming. This class is generally not used for normal course of action. This class is used to manage the exception handling.

Its object may be used with throw keyword throw the exception. This class may also be used with throws keyword to indicate the method for exception propagation. This class can also be used in catch block for exception handling. In totality we may say this class will be an exception handling class like predefined exception handling classes.

Example: Unit-7 (Program-09)

/* Write a program in java to create the student class which contains name and age. Age must be between 13-19. Check the age with AgeException. Exception should be handled at source of propagation. */

```
import java.util.*;
class AgeException extends Exception
{
    String msg;
    public AgeException()
    {
```

```
            msg="Age must be between 13 to 19";
        }
        public String toString()
        {
            return msg;
        }
}
class student
{
        private int age;
        private String name;
        public void input()
        {
            Scanner sc=new Scanner(System.in);
            System.out.print("Enter name ");
            name=sc.next();
            System.out.print("Enter age ");
            age=sc.nextInt();
            try
            {
                if(age<13 || age >19)
                {
                    throw new AgeException();
                }
            }
            catch(AgeException aex)
            {
                System.out.println(aex);
                System.out.print("Enter age ");
                age=sc.nextInt();
            }
        }
        public void display()
        {
```

```
                System.out.println("Name :"+name);
                System.out.println("Age :"+age);
        }
}
class unit07p09
{
        public static void main(String args[])
        {
                student s=new student();
                s.input();
                s.display();
        }
}
```

Result:

```
Enter name Rahul
Enter age 12
Age must be between 13 to 19
Enter age 14
Name :Rahul
Age :14
```

In the above example, exception is handled at the source of generation. We may also throw it from that place and handle it at the calling part.

Example: Unit-7 (Program-10)

/* Write a program in java to create the student class which contains name and age. Age must be between 13-19. Check the age with AgeException. Exception should not be handled at the source of exception propagation*/

```
import java.util.*;
class AgeException extends Exception
{
        String msg;
        public AgeException()
```

```
        {
                msg="Age must be between 13 to 19";
        }
        public String toString()
        {
                return msg;
        }
}
class student
{
        private int age;
        private String name;
        public void input() throws AgeException
        {
                Scanner sc=new Scanner(System.in);
                System.out.print("Enter name ");
                name=sc.next();
                System.out.print("Enter age ");
                age=sc.nextInt();
                if(age<13 || age >19)
                {
                        throw new AgeException();
                }
        }
        public void display()
        {
                System.out.println("Name :"+name);
                System.out.println("Age :"+age);
        }
}
```

```
class unit07p10
{
    public static void main(String args[]) throws AgeException
    {
        student s=new student();
        s.input();
        s.display();
    }
}
```

Result:

```
Enter name Amit
Enter age 23
Exception in thread "main" Age must be between 13 to 19
    at student.input(unit07p10.java:28)
    at unit07p10.main(unit07p10.java:43)
```

Please note that in the above example, we have just propagated the exception in the input() method with throw keyword. throw keyword is not in the try block so we must use throws keyword at the top of method with AgeException so that calling part of input() must handle it. In the above example, we have just identified it even in the main().

Example: Unit-7 (Program-11)

/* Write a program in java to create the student class which contains name and age. Age must be between 13-19. Check the age with AgeException. Exception should be handled at the callng of input() method*/

```
import java.util.*;
class AgeException extends Exception
{
    String msg;
    public AgeException()
    {
        msg="Age must be between 13 to 19";
```

```
    }
    public String toString()
    {
        return msg;
    }
}
class student
{
    private int age;
    private String name;
    public void input() throws AgeException
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter name ");
        name=sc.next();
        System.out.print("Enter age ");
        age=sc.nextInt();
        if(age<13 || age >19)
        {
            throw new AgeException();
        }
    }
    public void display()
    {
        System.out.println("Name :"+name);
        System.out.println("Age :"+age);
    }
}
class unit07p11
{
    public static void main(String args[])
```

```
    {
        student s=new student();
        try
        {
            s.input();
        }
        catch(AgeException aex)
        {
            System.out.println(aex);
        }
        s.display();
    }
}
```

Result:

```
Enter name Amit
Enter age 23
Age must be between 13 to 19
Name :Amit
Age :23
```

In the above example, we have propagated the exception from the input() and we have handled it with try block in the calling main().

Check Your Progress

1. What is user defined Exception?
2. How we can throw the user defined exception in java. Explain with example.
3. Describe the use of constructor and toSring() in user defined exception class.
4. Create the user defined exception for roll number of student. Roll number should not be negative. Create the student class which contains student's name, age, roll number and course. Implement that exception in student class. Store and display the record of one student.

7.8 SUMMARY

Java is one of the most popular object oriented programming language. So java supports the complete support so that its program should not terminate the runtime without the consent of user. To achieve this java offers exception handling. Exceptions are there which are generated in the program when any action that is other than the normal course of action is taken. Java provides predefined and user defined exceptions.

There are several steps related to exception handling in JVM. First it may be propagated, then it may be trapped and after identifying it may be handled. If exception is handled, program will not be terminated abnormally.

7.9 EXERCISE

Q. 1. What Exception in java? Explain the structure of the Exception class in java.

Q. 2. What are the types of exceptions in java? Compare checked and unchecked exceptions.

Q. 3. Compare throw and throws in regard to java.

Q. 4. Compare catch and finally block in java.

Q. 5. What is user defined exception in java? How we can create any user defined exception? Explain with example.

Q. 6. Create the salary exception in java. This exception must should be propagated when basic salary in less than 4,000 or it exceeds 50,000. In case basic salary is not following this criteria, assign the default basic salary for the employee i.e. Rs. 8,500. Define the employee class which contains employee name, id, age, basic salary, HRA, DA, PF, net salary and address. Net salary includes HRA and DA but PF is deducted from net salary. Create one object of employee and display the record.

Q. 7. Create the Book status exception. This exception must check that book may be issued, available or lost only. Any other status is not allowed in the field. Create the book class that contains book ID, ISBN, price, author, title, publication, edition and status.

Q. 8. Create the Quantity exception for product class. Quantity of sell should be less than or equal to available quantity. Product class contains product id, cost, available qty, selling qty and selling price. Create the object of product class to store the information of product and display its information.

Uttar Pradesh Rajarshi Tandon
Open University

Master in Computer Applications

MCS-112

Java Programming

BLOCK

3

MULTITHREADING, I/O AND STRINGS HANDLING

UNIT-8

Multithreaded Programming

UNIT-9

I/O in Java

UNIT-10

Strings and Characters

UNIT-11

Exploring Java I/O

Course Design Committee

Prof. Ashutosh Gupta — **Chairman**
Director (In-charge)
School of Computer and Information Science, UPRTOU Allahabad

Prof. Suneeta Agarwal — **Member**
Department of CSE
MNNIT Allahabad, Prayagraj

Dr. Upendra Nath Tripathi — **Member**
Associate Professor, Department of Computer Science
Deen Dayal Upadhyaya Gorakhpur University, Gorakhpur

Dr. Ashish Khare — **Member**
Associate Professor, Department of Computer Science
University of Allahabad, Prayagraj

Dr. Marisha — **Member**
Assistant Professor (Computer Science),
School of Science, UPRTOU Allahabad

Mr. Manoj Kumar Balwant — **Member**
Assistant Professor (computer science),
School of Sciences, UPRTOU Allahabad

Course Preparation Committee

Er. Pooja Yadav — **Author (Unit 1 to 14)**
Asst Professor, Dept. of Computer Science & Information Technology, MJPRU, Bareilly

Prof. Abhay Saxena — **Editor**
Professor, DSVV, Shantikunj, Haridwar (UK)-249411

Prof. Ashutosh Gupta — **Director (In-charge)**
School of Computer and Information Science, UPRTOU Allahabad

Dr. Marisha, — **Coordinator**
Asstt. Professor (Computer Science), School of Science, UPRTOU, Prayagraj

Mr. Manoj Kumar Balwant — **Coordinator**
Asstt. Professor (Computer Science), School of Science, UPRTOU, Prayagraj

ISBN :

Printed and Published by Dr. Arun Kumar Gupta Registrar, Uttar Pradesh Rajarshi Tandon Open University, 2020.
Printed By : Chandrakala Universal Pvt. 42/7 Jawahar Lal Neharu Road, Prayagraj.

UNIT-8 MULTITHREADED PROGRAMMING

Structure :

8.1 OBJECTIVE

This unit will cover the concept of multithreading and its basic concept. Implementation of multithreading in java will also be covered. This unit will also cover the methods to implement multithreading. Application areas of multithreading will also be covered. How to synchronize the multithreading, is also essential so it will also be implemented by various methods.

8.2 INTRODUCTION

Java program is treated as a thread during execution. Main is a thread in java. This will be a single thread application. If we need to have parallel execution, we may use multiple threads to be executed simultaneously. This will be known as multi-threading in java. For this we just need to create minimum one more thread as main thread is already available there. In game programming or animation, we generally use multi-threading.

8.3 MULTITHREADING

Multithreading in java is the process of executing multiple threads simultaneously. In context of operating system and java, a thread is a light weight process. Threads of one process may share memory area. So multithreading saves memory. Threads take less time in context switching. So multithreading saves time as well. Multi-threading is used in java programming, animation, transaction processing system, banking system, online fund transfer systems, mobile recharge systems etc. Threads are independent so they don't block the users. Exception in one thread does not affect other as they are independent.

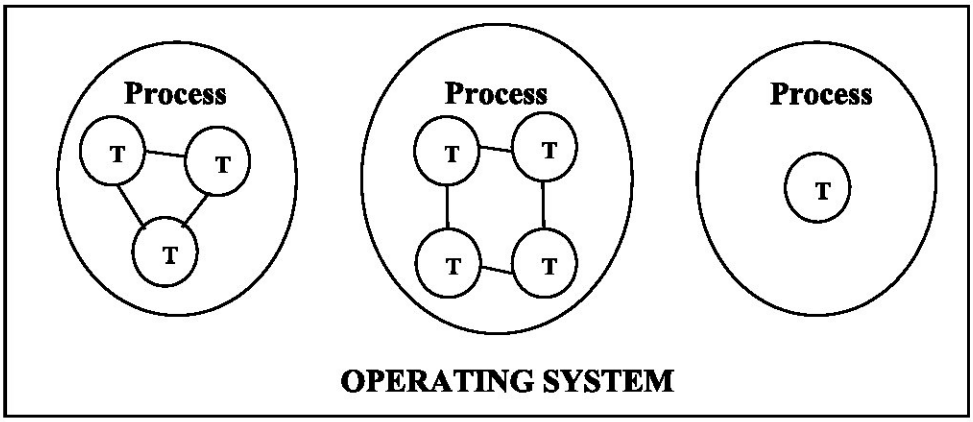


Figure 8.1

Above diagram shows the relation of multiple threads and multiple processes in the operating system. There are two types of multitasking-

Process based vs. thread based multi-tasking		
S. No.	**Process based multi-tasking**	**Thread based multi-tasking**
1	Process is heavy-weight.	Thread is light weight.
2	Cost of communication between processes is high	Cost of communication between threads is low.
3	Each process allocates a separate memory area	Threads share the same address space.

Table 8.1

So thread based multi-tasking is better as compared to process based multi-tasking.

Characteristics of Thread :

- Threads are independent.
- If there occurs exception in one thread, it doesn't affect other threads.
- It uses a shared memory area.
- There may be multi threads in one process.
- Threads may communicate to each other easily.

Benefits of Multi-Threading:

- Responsiveness
- Resource sharing
- Economy
- Scalability

Check Your Progress

1. What is thread in java? Explain its characteristics.
2. What is multi-threading? Explain its benefits.
3. Compare process based and thread based multi-threading.

8.4 THE MAIN THREAD

Java supports built-in support for multi-threading. One multi-thread application in java contains more than two threads. One thread is main thread in java. So for multi-threading java requires at least one more thread. Each thread defines a separate path of execution. Whenever we run a java program, one thread is started immediately. This thread is known as main thread of java. This thread executes of our java program.

Characteristics of main thread :

- Main thread is the first thread of java program
- Multiple threads may be as child threads may be generated from this thread.
- This main thread must be the last thread of execution.

Managing and controlling main thread :

This main thread is started as program is started. If we are looking to control it, we must first reference it. To reference this thread, this may be referenced by method currentThread() of Thread class. This reference may be assigned in the object of the Thread class as well.

For example :

Thread t=Thread.currentThread();

Some methods for threads are:

S.No.	Modifier	Method	Description
1.	static Thread	currentThread()	It returns a reference to the currently executing thread object.
2.	int	getPriority()	It returns the priority of the thread.
3.	void	setPriority()	It changes the priority of the thread.
4.	String	getName()	It returns the name of the thread.
5.	void	setName()	It changes the name of the thread.
6.	long	getId()	It returns the id of the thread.
7.	boolean	isAlive()	It tests if the thread is alive.
8.	Thread.State	getState()	It is used to return the state of the thread.
9.	String	toString()	It is used to return a string representation of this thread, including the thread's name, priority, and thread group.
10.	void	join()	It waits for a thread to die.
11.	static void	sleep()	It sleeps a thread for the specified amount of time.

Table 8.2

The concept of multi-threading needs proper understanding of these two things – a process and a thread. A process is a program in execution. A process can be further divided into independent units known as threads.

Example: Unit-8 (Program-01)

/* Write a program display main thread properties*/

```
class unit08p01
{
```

```
    public static void main(String args[])
    {
        Thread t = Thread.currentThread();
        System.out.println("Current thread: " + t);
        t.setName("My Thread");      // change the name of the thread
        System.out.println("After name change: " + t);
        System.out.println("Prioirty: "+t.getPriority());
        System.out.println("Thread ID: "+t.getId());
        System.out.println("Thread status: "+t.isAlive());
        System.out.println(t);
    }
}
```

Result:

```
Current thread: Thread[main,5,main]
After name change: Thread[My Thread,5,main]
Prioirty: 5
Thread ID: 1
Thread status: true
Thread[My Thread,5,main]
```

Above program shows some properties of main thread. We may also use some more properties and methods.

Example: Unit-8 (Program-02)

/* Write a program to implement sleep() method in main thread. */

```
class unit08p02
{
    public static void main(String args[])
    {
        Thread t = Thread.currentThread();
        System.out.println("Current thread: " + t);
        try
```

```
            {
                for(int n = 5; n > 0; n--)
                {
                    System.out.println(n);
                    Thread.sleep(1000);  //1000 is in milliseconds
                }
            }
            catch (InterruptedException e)
            {
                System.out.println("Main thread interrupted");
            }
        }
    }
```

Result:

```
Current thread: Thread[main,5,main]
5
4
3
2
1
```

Above program have shown the implementation of sleep() method in current thread. Current thread is main thread here. We may also implement such functionality in other child threads as well.

Check Your Progress

1. What is main thread in java? List 5 methods which may be used with current thread.
2. Write a program in java to implement and display the properties of main thread?
3. Write a program in java to change the name of the main thread.
4. Write a program in java to implement the sleep method to display the array elements of with the gap of 500 milliseconds.

8.5 JAVA THREAD MODEL

Main the current thread for each java program. By default java is single thread application. In that case main thread will be the only thread. But java supports built-in multi-thread programming. To implement multi-thread, we just need to create new thread.

There are two models to implement multi-threading in java.

1. via Thread class
2. via Runnable interface

Both are given in the java.lang package of java. So we need not to import any extra package for multithreading as lang package is by default imported in java compiler.

If we compare multithreading via Thread or Runnable, there are some differences:

Thread Class vs. Runnable Interface		
S.No.	**Thread class**	**Runnable Interface**
1	Thread is a class	Runnable is interface
2	It does not support multiple inheritance as it is class	It supports multiple inheritance as it is interface
3	It uses extends keyword	It used implements keyword
4	Each thread creates a unique object in separate memory locations	Each Tread share common memory locations
5	It consumes more memory	It consumes less memory
6	It introduces tight coupling	It introduces loose coupling

Table 8.3

Implementing multithreading via Thread Class :

A thread can be defined in two ways. First, by extending a Thread class that has already implemented a Runnable interface. Second, by directly implementing a Runnable interface. When you define a thread by extending Thread class you have to override the run() method in Thread class. On extending Thread class, we must override run() method in the class because it is an abstract method in super class. It will be the entry point for any thread execution. This will be called on calling start() method. One point should be keep in mind that run() method is to be defined but it can't be called while start() method is to be called but it should not be defined.

Syntax for multithreading via Thread class is as:

```
class ClassName1 extends Thread
{


}
```

Please note the in the above example, ClassName1 will be used in multithreading as it extends Thread class. If we are extending Thread class, it will indicate the java compiler that this class will be used in multithreading and this class will override run() method.

Example: Unit-8 (Program-03)

/* Write a program to create a new class for multi-threading. Display the number in reverse orders with the delay of some milliseconds. In the main thread display the numbers in normal order with delay of same milliseconds. */

```
class NewThread extends Thread
{
    public NewThread()          // Create a new, second thread
    {
        super("My Thread 1");
        System.out.println("Child thread: " + this);
    }
    // This is the entry point of thread.
    public void run()
    {
        try
        {
            for(int i = 5; i > 0; i--)
            {
                System.out.println("Child Thread: " + i);
                Thread.sleep(500);
            }
        }
```

```
            catch (InterruptedException e)
            {
                System.out.println("Child interrupted.");
            }
            System.out.println("Exiting child thread.");
        }
    }
    class unit08p03
    {
        public static void main(String args[])
        {
            NewThread t2=new NewThread(); // create a new thread
            t2.start();
            try
            {
                for(int i = 1; i <=5; i++)
                {
                    System.out.println("Main Thread: " + i);
                    Thread.sleep(1000);
                }
            }
            catch (InterruptedException e)
            {
                    System.out.println("Main thread interrupted.");
            }
            System.out.println("Main thread exiting.");
        }
    }
```

Result :

```
Child thread: Thread[My Thread 1,5,main]
```

```
Main Thread: 1
Child Thread: 5
Child Thread: 4
Main Thread: 2
Child Thread: 3
Child Thread: 2
Main Thread: 3
Child Thread: 1
Exiting child thread.
Main Thread: 4
Main Thread: 5
Main thread exiting.
```

Implementing multithreading via Runnable Interface :

When you define a thread implementing a Runnable interface you have to implement the only run() method of Runnable interface. The basic difference between Thread and Runnable is that each thread defined by extending Thread class creates a unique object and get associated with that object. On the other hand, each thread defined by implementing Runnable interface shares the same object.

Example: Unit-8 (Program-04)

/* Write a program to create a new class for multi-threading. Display the number in reverse orders with the delay of some milliseconds. In the main thread display the numbers in normal order with delay of same milliseconds. Implement the multithreading via Runnable interface. */

```
class NewThread implements Runnable
{
    Thread t;
    NewThread()
    {
        t = new Thread(this, "My Thread");  // Create a new, second thread
        System.out.println("Child thread: " + t);
        t.start();                          // Start the thread
    }
```

```
    public void run()      // This is the entry point for the second thread.
    {
        try
        {
            for(int i = 5; i > 0; i--)
            {
                System.out.println("Child Thread: " + i);
                Thread.sleep(500);
            }
        }
        catch (InterruptedException e)
        {
            System.out.println("Child interrupted.");
        }
        System.out.println("Exiting child thread.");
    }
}
class unit08p04
{
    public static void main(String args[])
    {
        NewThread t1=new NewThread(); // create a new thread
        try
        {
            for(int i = 1; i <=5; i++)
            {
                System.out.println("Main Thread: " + i);
                Thread.sleep(1000);
            }
        }
        catch (InterruptedException e)
        {
            System.out.println("Main thread interrupted.");
        }
```

```
                System.out.println("Main thread exiting.");
        }
    }
```

Result:

Child thread: Thread[My Thread,5,main]

Main Thread: 1

Child Thread: 5

Child Thread: 4

Main Thread: 2

Child Thread: 3

Child Thread: 2

Main Thread: 3

Child Thread: 1

Exiting child thread.

Main Thread: 4

Main Thread: 5

Main thread exiting.

Above example is implemented via runnable interface. Please note one more point in the example that sleep() method is checked exception so we must handle the InterruptedException in catch block.

It is preferred to implement a Runnable interface instead of extending Thread class. As implementing Runnable makes your code loosely coupled as the code of thread is different from the class that assign job to the thread. It requires less memory and also allows a class to inherit any other class.

There are some methods used in multithreading:

S. No.	Modifier	Method	Description
1.	void	start()	It is used to start the execution of the thread.
2.	void	run()	It is used to do an action for a thread.
3.	static void	yield()	It causes the currently executing thread object to pause and allow other threads to execute temporarily.

4.	void	suspend()	It is used to suspend the thread.
5.	void	resume()	It is used to resume the suspended thread.
6.	void	stop()	It is used to stop the thread.
7.	void	destroy()	It is used to destroy the thread group and all of its subgroups.
8.	boolean	isDaemon()	It tests if the thread is a daemon thread.
9.	void	setDaemon()	It marks the thread as daemon or user thread.
10.	void	interrupt()	It interrupts the thread.
11.	boolean	isinterrupted()	It tests whether the thread has been interrupted.
12.	static boolean	interrupted()	It tests whether the current thread has been interrupted.
13.	void	notify()	It is used to give the notification for only one thread which is waiting for a particular object.
14.	void	notifyAll()	It is used to give the notification to all waiting threads of a particular object.

Table 8.4

Stages of any Thread :

- **New:** The thread is in new state if you create an instance of Thread class but before the invocation of start() method.
- **Runnable:** The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.
- **Running:** The thread is in running state if the thread scheduler has selected it.
- **Non-Runnable (Blocked):** This is the state when the thread is still alive, but is currently not eligible to run.
- **Terminated:** A thread is in terminated or dead state when its run() method exits.

When a new thread is created, it is in the new state. The thread has not yet started to run when thread is in this state. When a thread lies in the new state, it's code is yet to be run and hasn't started to execute.

Check Your Progress

1. Compare thread class and Runnable interface.
2. What are the methods used in multi-threading. Explain any 5 with example.
3. Create the class with Runnable interface. The class contains an array of 10 integers. Input the record and sort the elements with run() method to implementing multithreading. After sorting, display the records.
4. What are the states of any thread in java? Explain with example.

8.6 THREAD PRIORITIES

There are times when we may need to change the priority of threads because some threads may be of more importance or they should be completed before other thread. This sequence may also be affected with thread priorities.

Thread priority is the value assigned to each thread for deciding its execution sequence via OS. This priority is an integer value which ranges from 1 to 10. 1 is the minimum priority while 10 is the maximum priority. 5 is the normal priority. 5 is the default priority assigned to each new thread.

We may change this default priority from 5 to high or low. Before changing the priority, it is recommendable to view the current priority of the thread. To view the priority, we use the method getPriority().

Syntax to view the priority of any thread:

```
ThreadObject.getPriority();
```

This method will return the priority. It may range from 1 to 10. By default it is 5.

To change the priority of thread, use the setPriority() method.

Syntax to change the priority of thread in java:

```
ThreadObject.setPriority( new priority );
```

Please note the new priority in the above syntax must be a integer value ranging from 1 to 10.

1 is represented via MIN_PRIORITY and 1 is represented by MAX_PRIORITY. Default values 5 is represented by NORM_PRIORITY. These priorities are defined as final variables within Thread.

Example: Unit-8 (Program-05)

/* Write a program display read and assign the thread priorities*/

```
class unit08p05
{

    public static void main(String args[])
    {
        Thread t=Thread.currentThread();
        System.out.println("Priority of current thread "+ t.getPriority());
        t.setPriority(10);
        System.out.println("Updated Priority of current thread "+
t.getPriority());
        t.setPriority(1);
        System.out.println("Updated Priority of current thread "+
t.getPriority());
        t.setPriority(5);
        System.out.println("Updated Priority of current thread "+
t.getPriority());
    }
}
```

Result:

Priority of current thread 5

Updated Priority of current thread 10

Updated Priority of current thread 1

Updated Priority of current thread 5

Please note that in the above example, priority is assigned to current thread, it may be assigned to any existing object.

Check Your Progress

1. What is thread priority? Explain all with the method to access and edit it.
2. How the priority of any thread is managed. How could we change it?
3. Create the student class which contains name, age, roll number and course. Change the priority of this thread in main() method. Store and display one record of any student

8.7 SYNCHRONIZATION IN JAVA

When multiple thread are being executed in parallel, they may requires the same recourse at the same time. This may lead to produce erroneous and unforeseen results. If this situation occurs then we have to find the way out to resolve this problem. When two or more threads need access to a shared resource, they need some way to ensure that the resource will be used by only one thread at a time. The process by which this is achieved is called synchronization.

Synchronization method make sure that only one thread can access the resource at a given point of time.

Key to synchronization is the concept of the monitor. A monitor is an object that is used as a mutually exclusive lock, or mutex. Only one thread can own a monitor at a given time. When a thread acquires a lock, it is said to have entered the monitor. All other threads attempting to enter the locked monitor will be suspended until the first thread exits the monitor. These other threads are said to be waiting for the monitor. A thread that owns a monitor can reenter the same monitor if it so desires.

Thread synchronization mainly resolve these two problems:

- To prevent thread interference
- To prevent consistency problem.

8.7.1 TYPES OF SYNCHRONIZATION

There are mainly two types of synchronization in java.

1. Process Synchronization
2. Thread Synchronization
    a. Mutual Exclusive
        i. Synchronized method
        ii. Synchronized block
        iii. Static Synchronization
    b. Cooperation (Inter-thread communication in java)

8.7.2 THREAD SYNCHRONIZATION

8.7.2.1 SYNCHRONIZED METHOD

Thread synchronization method that only one thread can access the resource at a given point of time. This synchronized method will make sure that at a time one thread will enter the method. If any other thread tries to enter that

method at the same time, then new method has to wait in queue till the completion of current method. For this we use synchronised keyword in the method signature.

Syntax for method synchronisation is:

```
synchronised methodName()
{
    :
    :
}
```

In the above method name may be any name as per our requirement and it may have parameter list or it may return the value as well.

Example: Unit-8 (Program-06)

/* Write a program display the pattern via different threads in non-synchronised method. */

```
class Series
{
    void display(int x)    // non-synchronised method
    {
        int i;
        System.out.println("This is the method of series ");
        for(i=1;i<=x;i++)
        {
            try
            {
                Thread.sleep(100);
            }
            catch(InterruptedException ex)
            {
                System.out.println(ex);
            }
            System.out.print(i);
```

```
            }
        }
}
class SeriesThread1 extends Thread
{
    int x;
    Series s;
    public SeriesThread1(int x)
    {
            s=new Series();
            this.x=x;
    }
    public void run()
    {
            System.out.println("This is thread of SeriesThread");
            s.display(x);
    }
}
class SeriesThread2 extends Thread
{
    int x;
    Series s;
    public SeriesThread2(int x)
    {
            s=new Series();
            this.x=x;
    }
    public void run()
    {
            System.out.println("This is thread of SeriesThread");
```

```
            s.display(x);
        }
    }
    class unit08p06
    {
        public static void main(String args[])
        {
            SeriesThread1 s1=new SeriesThread1(5);
            SeriesThread2 s2=new SeriesThread2(3);
            s1.start();
            s2.start();
        }
    }
```

Result:

```
This is thread of SeriesThread
This is thread of SeriesThread
This is the method of series
This is the method of series
11223345
```

In the above example, we have created the object of Series class which contains a non-synchronised method display. So display method will be call for each its call with parallel call if via multiple threads.

Example: Unit-8 (Program-07)

/* Write a program display the pattern via different threads in synchronised method. */

```
class Series
{
    synchronized void display(int x)      // synchronised method
    {
        int i;
```

```
                System.out.println("This is the method of series ");
                for(i=1;i<=x;i++)
                {
                        try
                        {
                                Thread.sleep(100);
                        }
                        catch(InterruptedException ex)
                        {
                                System.out.println(ex);
                        }
                        System.out.print(i);
                }
        }
}
class SeriesThread1 extends Thread
{
        int x;
        Series s=new Series();
        public SeriesThread1(int x)
        {
                this.x=x;
        }
        public void run()
        {
                System.out.println("This is thread of SeriesThread");
                s.display(x);
        }
}
class SeriesThread2 extends Thread
```

```
{
    int x;
    Series s=new Series();
    public SeriesThread2(int x)
    {
        this.x=x;
    }
    public void run()
    {
        System.out.println("This is thread of SeriesThread");
        s.display(x);
    }
}
class unit08p07
{
    public static void main(String args[])
    {
        SeriesThread1 s1=new SeriesThread1(5);
        SeriesThread2 s2=new SeriesThread2(3);
        s1.start();
        s2.start();
    }
}
```

Result:

```
This is thread of SeriesThread
This is thread of SeriesThread
This is the method of series
This is the method of series
11223345
```

In the above example, we have created the display() method in Series class as synchronised method. But we have called this method via different threads. Both

threads t1 and t2 are having different object in them so we need not to worry for synchronization as values for objects of Series in t1 and t2 are different. So Synchronization will not affect here.

Example: Unit-8 (Program-08)

/* Write a program display the pattern via different threads in synchronised method with same object reference. */

```
class Series
{
    synchronized void display(int x)     // synchronised method
    {
        int i;
        System.out.println("\nThis is the method of series ");
        for(i=1;i<=x;i++)
        {
            try
            {
                Thread.sleep(100);
            }
            catch(InterruptedException ex)
            {}
            System.out.print(i);
        }
    }
}
class SeriesThread1 extends Thread
{
    int x;
    Series s;
    public SeriesThread1(Series s,int x)
    {
        this.s=s;
```

```
            this.x=x;
        }
        public void run()
        {
            System.out.println("This is thread of SeriesThread");
            s.display(x);
        }
}
class SeriesThread2 extends Thread
{
        Series s;
        int x;
        public SeriesThread2(Series s,int x)
        {
            this.s=s;
            this.x=x;
        }
        public void run()
        {
            System.out.println("This is thread of SeriesThread");
            s.display(x);
        }
}
class unit08p08
{
        public static void main(String args[])
        {
            Series s=new Series();
            SeriesThread1 s1=new SeriesThread1 (s,5);      // object s as
parameter
            SeriesThread2 s2=new SeriesThread2 (s,3);      // object s as
parameter
```

```
            s1.start();
            s2.start();
        }
    }
```

Result:

```
This is thread of SeriesThread
This is thread of SeriesThread
This is the method of series
123
This is the method of series
12345
```

So from the above example it is clear that if we call any synchronised method by multiple thread with same object reference then only one thread have the access to that method at a time. On the other hand if multiple threads with difference object reference tries to call the same synchronised method then it is possible simultaneously as object reference is different.

We can understand this by the example of withdrawal from bank. If one account holder tries to withdraw the amount from his account at bank counter and in parallel his family member tries to access his ATM card from ATM machine then only one person will be able to do the transaction in parallel. If there is time difference then there is no problem otherwise, only one person will be able to withdraw the amount at a time. On the other side if two different account holder tires too withdraw their amount from bank counter and ATM machine then there is no problem as both account holders are different. In our above example of program unit0807 and unit0708 shows the same difference.

8.7.2.2 SYNCHRONIZED BLOCK

If we don't need entire method to be accessed by one object at a time, it will increase the access time for new objects. Synchronized blocks in Java are marked with the synchronized keyword. All synchronized blocks synchronized on the same object can only have one thread executing inside them at a time. All other threads attempting to enter the synchronized block are blocked until the thread inside the synchronized block exits the block.

Example: Unit-8 (Program-9)

```
/* Write a program display the pattern via different threads in synchronised block
```

```
with same object reference. */

class Series
{
    void display(int x)
    {
        int i;
        System.out.println("\nThis is the method of series ");
        synchronized (this) // synchronised block with current object reference
        {
          for(i=1;i<=x;i++)
          {
              try
              {
                  Thread.sleep(100);
              }
              catch(InterruptedException ex)
              {}
              System.out.print(i);
          }
        }
    }
}
class SeriesThread1 extends Thread
{
    int x;
    Series s;
    public SeriesThread1(Series s,int x)
    {
        this.s=s;
        this.x=x;
```

```
        }
        public void run()
        {
                System.out.println("This is thread of SeriesThread");
                s.display(x);
        }
}
class SeriesThread2 extends Thread
{
        Series s;
        int x;
        public SeriesThread2(Series s,int x)
        {
                this.s=s;
                this.x=x;
        }
        public void run()
        {
                System.out.println("This is thread of SeriesThread");
                s.display(x);
        }
}
class unit08p09
{
        public static void main(String args[])
        {
                Series s=new Series();
                SeriesThread1 s1=new SeriesThread1 (s,5);
                SeriesThread2 s2=new SeriesThread2 (s,3);
                s1.start();
```

```
            s2.start();
        }
    }
```

Result:

This is thread of SeriesThread

This is thread of SeriesThread

This is the method of series

This is the method of series

12312345

In the above example, we have implemented one block rather than the entire method to be synchronised. As in case of synchronised methods, entire method is set to waiting for all other methods call, this may be very time consuming and besides this we generally need some statements to be synchronised. So whenever we need some statements to be synchronised rather than entire method, we may use synchronised blocks. We have also specified the object reference for synchronisation of block. This is more powerful than synchronised method as we may also apply multiple synchronised blocks in one class.

8.7.2.3 STATIC SYNCHRONIZATION

Till now we have implemented synchronization for the threads having similar object reference. If this is applied then two threads with same object reference may not access the same synchronised block or method. But if multiple threads with different object reference tries to access the synchronised method or block than there is no problem.

At times, we may need to synchronised the method or block for all threads irrespective of object reference, than we may use static synchronization. Static synchronization is applicable to all threads which tries to access the synchronized methods irrespective of object reference. This is one of the solution to the lock problem which is caused by non-static synchronisation.

Example: Unit-8 (Program-10)

```
/* Write a program display the pattern via different threads in synchronised static
method with difference object reference. */
class Series
```

```
{
    synchronized static void display(int x)
    {
        int i;
        System.out.println("\nThis is the method of series ");
        for(i=1;i<=x;i++)
        {
            try
            {
                Thread.sleep(100);
            }
            catch(InterruptedException ex)
            {}
            System.out.print(i);
        }
    }
}
class SeriesThread1 extends Thread
{
    int x;
    Series s;
    public SeriesThread1(Series s,int x)
    {
        this.s=s;
        this.x=x;
    }
    public void run()
    {
        System.out.println("This is thread of SeriesThread");
        s.display(x);
    }
}
class SeriesThread2 extends Thread
{
```

```
        Series s;
        int x;
        public SeriesThread2(Series s,int x)
        {
                this.s=s;
                this.x=x;
        }
        public void run()
        {
                System.out.println("This is thread of SeriesThread");
                s.display(x);
        }
}
class unit08p10
{
        public static void main(String args[])
        {
                Series sr1=new Series();
                Series sr2=new Series();
                SeriesThread1 s1=new SeriesThread1 (sr1,5);
                SeriesThread2 s2=new SeriesThread2 (sr2,3);
                s1.start();
                s2.start();
        }
}
```

Result:

```
This is thread of SeriesThread
This is thread of SeriesThread
This is the method of series
123
This is the method of series
12345
```

In the above example, method is defined as synchronised static method. This will allow only one thread to access this method irrespective of their object reference.

Example: Unit-8 (Program-11)

/* Write a program display the pattern via different threads in synchronised static block with different object reference. */

```
class Series
{
    static void display(int x)
    {
        int i;
        System.out.println("\nThis is the method of series ");
        synchronized (Series.class)  // synchronised static block in class reference
        {
          for(i=1;i<=x;i++)
          {
            try
            {
                Thread.sleep(100);
            }
            catch(InterruptedException ex)
            {}
            System.out.print(i);
          }
        }
    }
}
class SeriesThread1 extends Thread
{
    int x;
    Series s;
    public SeriesThread1(Series s,int x)
    {
        this.s=s;
        this.x=x;
```

```
    }
    public void run()
    {
        System.out.println("This is thread of SeriesThread");
        s.display(x);
    }
}
class SeriesThread2 extends Thread
{
    Series s;
    int x;
    public SeriesThread2(Series s,int x)
    {
        this.s=s;
        this.x=x;
    }
    public void run()
    {
        System.out.println("This is thread of SeriesThread");
        s.display(x);
    }
}
class unit08p11
{
    public static void main(String args[])
    {
        Series sr1=new Series();
        Series sr2=new Series();
        SeriesThread1 s1=new SeriesThread1 (sr1,5);
        SeriesThread2 s2=new SeriesThread2 (sr2,3);
        s1.start();
        s2.start();
    }
}
```

Result:

This is thread of SeriesThread

This is thread of SeriesThread

This is the method of series

This is the method of series

12312345

In the above example, synchronised block is declared as static synchronised because its parameter is specified in regard of the class rather than then the current object reference. So it will restrict the access of this block for one method call only, if other calls are made than they have to wait for this block.

Check Your Progress

1. What is synchronisation? Describe the types of thread synchronization in java.
2. Differentiate between method synchronisation and block synchronization.
3. Differentiate between static and non-static synchronisation.
4. Write a program in java to create a class which display the matrix with different thread. Display method should be synchronised.
5. Write a program in java to display the table of any number via different threads at a time but this method should be synchronised so that one table is properly printed.

8.8 INTER THREAD COMMUNICATION

Multithreading in java is its built-in feature. Threads are independent to each other. But threads may communicate to each other. There are some methods which allows threads to communicate to each other. Although one thread is designed for single purpose only and if we need other task than we design other thread for that purpose, yet at times they may need to communicate to each other.

When different threads communicate to each other, it is called inter thread communication. By default thread execution is asynchronous. We may apply synchronisation to restrict the execution of parallel thread execution for any common code. Multithreading divides our problem into discrete and logical units.

Threads may also be used for polling as they may repeatedly check for any specific criteria after specific interval and if criteria is matched, that specific code may execute. But this pooling wastes CPU time. Suppose we take the example of classic queuing problem. In this problem, one thread generates some data and that data is consumed by other thread. To make it better, suppose producer has to generate new data only when consumer consumes its existing data. In polling

system consumer wastes many CPU cycles for producer to produce. Once the producer has finished, it would start polling, wasting more CPU cycles waiting for the consumer to finish.

To avoid polling, java provides elegant inter-process communication mechanism. For this java give three methods: wait(), notify() and notifyAll(). These methods are implemented as final in object as all classes have them. All these methods may be called only in synchronisation context. Rules for these methods are –

- *wait( ) method*: it tell the calling thread to give up the monitor and move into sleep till any other thread moves into same method and revokes (notify) it.
- *notify( )*: This method wakes up the earlier thread for which wait() was called on the same object.
- *notifyAll( )*: Like notify() method, this method wakes up but it wakes up all pervious thread which were called by wait() method on same object.
- *join( )*: This method is applicable to all child threads. If we wish to wait for the current thread till the termination of its parent thread than we may call the join() method. This method allows us the maximum amount of time that we want to wait specific thread to terminate. This is again the final method.

There are some more methods for inter thread communication-

- *suspend( )*: This method is depreciated in java2 as this method may lead to serious system failures. This method suspends the execution of current thread and later on it may be resumed. This method call is not recommended in new java versions. The major drawback with suspend was that even when we suspend the thread, its locks are not released on resources so other threads could not get the access to those locked resources.
- *resume( )*: Once thread is suspended, it may be wakes up by resume() method. As suspend() method is depreciated, so this resume() method is also depreciated because resume() can't be called without suspend() method. In new java versions, so this methods is also not used these days.
- *stop( )*: This method is used to stop the execution of the thread. Once this method is called, this will stop the execution of current thread.

Example: Unit-8 (Program-12)

/* Write a program to create the thread class with wait() in its run() method. Create two more thread classes which has first thread class object as its data member. In the second thread class call wait() in its run() and in the third thread class call notify() in its run() method. */

```
class BaseThreadSeries extends Thread
{
```

```
    public void run()
    {
      synchronized(this)
      {
          System.out.println("Thread Started
"+Thread.currentThread().getName());
          try
          {
            System.out.println("wait() called for
"+Thread.currentThread().getName());
            this.wait();
          }
          catch(InterruptedException ex)
          {
            System.out.println(ex);
          }
          System.out.println("notify()
"+Thread.currentThread().getName());
      }
    }
}
class ThreadSeries1 extends Thread
{
    BaseThreadSeries bts;
    public ThreadSeries1(BaseThreadSeries bts)
    {
          this.bts=bts;
    }
    public void run()
    {
      synchronized(this.bts)
      {
          System.out.println("Thread Started
"+Thread.currentThread().getName());
          try
```

```
            {
               System.out.println("wait()                called                for
"+Thread.currentThread().getName());
               this.bts.wait();
            }
            catch(InterruptedException ex)
            {
               System.out.println(ex);
            }
            System.out.println("notify()
"+Thread.currentThread().getName());
         }
      }
}

class ThreadSeries2 extends Thread
{
      BaseThreadSeries bts;
      public ThreadSeries2(BaseThreadSeries bts)
      {
            this.bts=bts;
      }
      public void run()
      {
         synchronized(this.bts)
         {
            System.out.println("Thread                              Started
"+Thread.currentThread().getName());
            System.out.println("wait()                called                for
"+Thread.currentThread().getName());
            this.bts.notify();
            System.out.println("notify()
"+Thread.currentThread().getName());
         }
      }
}
```

```
class unit08p12
{
    public static void main(String args[]) throws InterruptedException
    {
            BaseThreadSeries bts=new BaseThreadSeries();
            ThreadSeries1 ts1=new ThreadSeries1(bts);
            ThreadSeries2 ts2=new ThreadSeries2(bts);
            Thread t1=new Thread(bts,"T1");
            Thread t2=new Thread(ts1,"T2");
            Thread t3=new Thread(ts2,"T3");
            t1.start();
            t2.start();
            Thread.sleep(1000);
            t3.start();
    }
}
```

Result:

```
Thread Started T1
wait() called for T1
Thread Started T2
wait() called for T2
Thread Started T3
wait() called for T3
notify()  T3
notify()  T1
```

In the above example, we have called wait() method in the BaseThreadSeries class run() and same in the ThreadSeries1 class run(). But in ThreadSeries2 class run() method, we have called notify() method. This method will notify one thread which enter before this thread to same object reference. Hence it is clear two threads are set to wait() but we have called notify(0 once so it will notify only one thread which was sent to wait() first (in this case T1). It will not notify the any other thread (in this case T2). So our program will not terminates normally, it will keep on waiting for the notify of T2.

Example: Unit-8 (Program-13)

/* Write a program to create the thread class with wait() in its run() method. Create two more thread classes which has first thread class object as its data member. In the second thread class call wait() in its run() and in the third thread class call notify() in its run() method. */

```
class BaseThreadSeries extends Thread
{
    public void run()
    {
      synchronized(this)
      {
            System.out.println("Thread Started
"+Thread.currentThread().getName());
            try
            {
              System.out.println("wait() called for
"+Thread.currentThread().getName());
              this.wait();
            }
            catch(InterruptedException ex)
            {
              System.out.println(ex);
            }
            System.out.println("notify()
"+Thread.currentThread().getName());
      }
    }
}
class ThreadSeries1 extends Thread
{
    BaseThreadSeries bts;
    public ThreadSeries1(BaseThreadSeries bts)
    {
            this.bts=bts;
    }
```

```
    public void run()
    {
      synchronized(this.bts)
      {
            System.out.println("Thread Started
"+Thread.currentThread().getName());
            try
            {
              System.out.println("wait() called for
"+Thread.currentThread().getName());
              this.bts.wait();
            }
            catch(InterruptedException ex)
            {
              System.out.println(ex);
            }
            System.out.println("notify()
"+Thread.currentThread().getName());
      }
    }
}
class ThreadSeries2 extends Thread
{
    BaseThreadSeries bts;
    public ThreadSeries2(BaseThreadSeries bts)
    {
            this.bts=bts;
    }
    public void run()
    {
      synchronized(this.bts)
      {
            System.out.println("Thread Started
"+Thread.currentThread().getName());
            System.out.println("wait() called for
```

```
"+Thread.currentThread().getName());
                this.bts.notifyAll();     // NOTIFY ALL
                System.out.println("notify()
"+Thread.currentThread().getName());
          }
       }
}
class unit08p13
{
       public static void main(String args[]) throws InterruptedException
       {
                BaseThreadSeries bts=new BaseThreadSeries();
                ThreadSeries1 ts1=new ThreadSeries1(bts);
                ThreadSeries2 ts2=new ThreadSeries2(bts);
                Thread t1=new Thread(bts,"T1");
                Thread t2=new Thread(ts1,"T2");
                Thread t3=new Thread(ts2,"T3");
                t1.start();
                t2.start();
                Thread.sleep(1000);
                t3.start();
       }
}
```

Result:

```
Thread Started T1
wait() called for T1
Thread Started T2
wait() called for T2
Thread Started T3
wait() called for T3
notify()  T3
notify()  T2
notify()  T1
```

In the above example, we have called notifyAll() method, so it will notify all thread for same object reference from waiting. So nothing is left for waiting here.

Check Your Progress

1. What is inter-thread communication?
2. Explain wait(), notify() and notifyAll() methods of java.
3. What were the drawbacks of suspend() method of java due to which it was depreciated.
4. What is the purpose of join() method?
5. Create a class in java to implement the notify() along with wait() method.

8.9 SUMMARY

Multithreading is one of the most exiting feature of java. It allows light weight application for games, transaction processing, banking and other areas. It also allows inter-process communication between threads so that they may communicate to each other. Java cares for deadlock so that programmer may avoid it.

8.10 EXERCISE

Q.1 What is multithreading? Explain the states of a thread.

Q.2 Describe the types of multithreading with example.

Q.3. Create a java program to show the properties of the current thread.

Q.4 Create a java program to display the thread priority and modify it.

Q.5 Create a java program to implement notifyAll() method with wait() method.

Q.6 Compare notify() and notifyAll() methods.

Q.7 What are the disadvantages associated with suspend() and resume() due to which they were depreciated from the java library.

Q.8 List the application areas of multithreading by java.

Q.9 Create a java program to implement block synchronization in multithreading.

Q.10 Create a java program to implement synchronised method in multithreading.

Q.11 Create a java program to implement static synchronization in multithreading.

Q. 12. What is deadlock? How it is linked with multithreading?

UNIT-9 I/O IN JAVA

Structure :

9.1 OBJECTIVE

Objective of this unit is to explore the sources and destinations of data for the program. This unit will cover how sources will be the input for program and destinations will be output of our program. This unit will cover the various classes for input and output of the programs and how they will be implemented. This unit will also tell the methodology to read and write data from programs in the form of file handling.

9.2 INTRODUCTION

Java is one of the most popular programming languages used these days. So it is full of approximate all features which are required. Input and Output is one of the basic feature that should be available in any programming language. So java provides that IO support in its programming language. Java is object oriented so Input and Output is also available in the form of class and objects. Java supports exception handling, multithreading etc., so these features are also available in java IO.

9.3 IO BASICS

Accepting values from standard input device that is keyboard and producing output to standard output device that is visual display unit, is one of the most basic thing available in all programming languages. As java is object oriented language, so these features are available in the form of class and objects in java.

9.3.1 IO PACKAGE IN JAVA (JAVA.IO)

In java java.io package is used for IO related classes. It contains a collection of classes to be used during IO operations. All classes are given in a pair. It means for every input class there is a related output class. If we accept the data from one input class then output has to be performed from same pair of output class.

9.3.2 TYPES OF IO OPERATION IN JAVA

Java supports 2 types of IO operations. First is byte based IO and second is character based IO. Key difference between both is on the basis of number of bits processing at a time.

Byte based IO: performs IO in bytes (8 bit) at a time

Character based IO: performs IO in character (16 bits unicode) at a time.

Here is a table which shows the classes available in both segments.

Purpose	Byte Based Classes		Character Based Classes	
	Input	Output	Input	Output
Basic	InputStream	OutputStream	Reader	Writer
			InputStreamReader	OutputStreamWriter
Arrays	ByteArrayInputStream	ByteArrayOutputStream	CharArrayReader	CharArrayWriter
Files	FileInputStream	FileOutputStream	FileReader	FileWriter
	RandomAccessFile	RandomAccessFile		
Pipes	PipedInputStream	PipedOutputStream	PipedReader	PipedWriter
Buffering	BufferedInputStream	BufferedOutputStream	BufferedReader	BufferedWriter
Filtering	FilterInputStream	FilterOutputStream	FilterReader	FilterWriter
Parsing	PushbackInputStream		PushbackReader	
	StreamTokenizer		LineNumberReader	
Strings			StringReader	StringWriter
Data	DataInputStream	DataOutputStream		
Data - Formatted		PrintStream		PrintWriter
Objects	ObjectInputStream	ObjectOutputStream		
Utilities	SequenceInputStream			

Table 9.1

From the above table it is clear that all classes are there in pair for their use. At different situation, we use different set of classes. We use the same set input form

keyboard or input from file. In the same way output may be performed on file or onto computer screen.

9.3.3 SOURCE AND DESTINATION IN JAVA IO

We are taking the data from keyboard into program will be treated as input and sending data to other program will be treated as output. It might be a little confusing in some conditions. So we may explain this via given diagram-

Figure 9.1

Above diagram shows the principle of program reading from source and writing to a destination. Source or destination may be-

- In-memory Buffers (e.g. variables, arrays)
- Files
- Pipes
- Network Connections
- System.in, System.out, System.error

Check Your Progress

1. Explain the standard input device, standard output device and standard error.
2. Describe the types of IO operation in java.
3. Compare byte oriented and character oriented operation in java.
4. Explain the sources and destinations of program for data.

9.4 STREAMS AND STREAM CLASSES

Stream is the sequence of data. In java stream is based on bytes. We may assume stream as a stream of water that flows continuously.

In java, IO stream is the core concept. If we say conceptually, java stream is endless flow of data. We may read from java stream or we may write to java stream. For reading, stream is connected to source and for writing, stream is connected to destination. As we know source and destination may be file, array or variable etc.

9.4.1 TYPES OF STREAM CLASSES IN JAVA

Java supports two types of stream classes in programming.

1. Byte Stream
2. Character Stream

9.4.2 BYTE STREAM IN JAVA

Byte stream provides processing in bytes in java. It is used for reading or writing data in binary format as data is processed in bytes. On the top of byte stream there are two abstract classes in java -

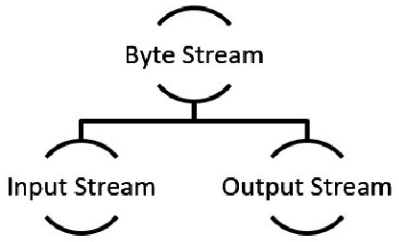


Figure 9.2

Each byte stream (input or output stream) has several subclasses to handle the data for various sources and destinations. The abstract classes has several methods which are overridden in sub classes for the programmers to use.

Here is the list of byte stream classes in java-

Byte Stream Class	Purpose
BufferedInputStream	Buffered input stream
BufferedOutputStream	Buffered output stream
ByteArrayInputStream	Input stream that reads from a byte array
ByteArrayOutputStream	Output stream that writes to a byte array
DataInputStream	An input stream that contains methods for reading the Java standard data types
DataOutputStream	An output stream that contains methods for writing the Java standard data types
FileInputStream	Input stream that reads from a file
FileOutputStream	Output stream that writes to a file
FilterInputStream	Implements InputStream
FilterOutputStream	Implements OutputStream
InputStream	Abstract class that describes stream input

OutputStream	Abstract class that describes stream output
PipedInputStream	Input pipe
PipedOutputStream	Output pipe
PrintStream	Output stream that contains print( ) and println( )
PushbackInputStream	Input stream that supports one-byte "unget," which returns a byte to the input stream
RandomAccessFile	Supports random access file I/O
SequenceInputStream	Input stream that is a combination of two or more input streams that will be read sequentially, one after the other

Table 9.2

9.4.3 CHARACTER STREAM IN JAVA

Character stream provides processing in characters in java. It is used for reading or writing data in character format as data is processed in characters. Java supports Unicode format so in java one character consumes 2 bytes. That's why in character stream classes java process 2 bytes at a time.

There are two character stream in java-

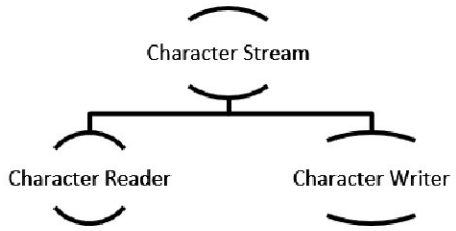


Figure 9.3

Here is the list of character stream classes available in java-

Stream Class	**Purpose**
BufferedReader	input character stream
BufferedWriter	Buffered output character stream
CharArrayReader	Input stream that reads from a character array
CharArrayWriter	Output stream that writes to a character array

FileReader	stream that reads from a file
FileWriter	stream that writes to a file
FilterReader	reader
FilterWriter	Filtered writer
InputStreamReader	Input stream that translates bytes to characters
LineNumberReader	Input stream that counts lines
OutputStreamWriter	Output stream that translates characters to bytes
PipedReader	Input pipe
PipedWriter	Output pipe
PrintWriter	Output stream that contains print( ) and println( )
PushbackReader	Input stream that allows characters to be returned to the input stream
Reader	Abstract class that describes character stream input
StringReader	stream that reads from a string
StringWriter	Output stream that writes to a string
Writer	Abstract class that describes character stream output

Table 9.3

9.4.4 THE PREDEFINED STREAM

As we are already aware of with three standard devices:

Standard Input (stdin): This is used to input the data to program. As a standard input device, generally keyboard is used and represented as stdin.

Standard Output (stdout): This is used to produce the data from program. As a standard output device, generally VDU is usedand represented as stdout.

Standard Error (stderr): During the program execution, error is generally identified a standard error and represented as stderr and display on standard output device.

As we know java implicitly imports java.lang package. It package defines a package called System. This class encapsulates many aspects of runtime requirement of java. System class also defines three predefined stream variable- in, out and err. These three are also declared as public so they may be accessed from outside the class.

These three stream variables are those three standard devices used by other programming language-

- **System.in**: used for standard input stream. By default, it is the console. It is the object of InputStream class.
- **System.out**: used for standard output stream. By default, it is keyboard in java. It is the object of PrintStream class.
- **System.err**: used for standard error stream. It is also console by default in java. It is the object of PrintStream class.

Check Your Progress

1. What is stream in java? Explain the types of streams in java.
2. List the abstract classes of java streams. Explain why they are abstract.
3. What are the predefined streams in java? Explain in detail.
4. Compare byte oriented streams and character oriented streams

9.5 READING FROM AND WRITING TO CONSOLE

In java console is represented by System.out. out is the object of PrintStream class inside the java.lang package. It is the standard output device.

Java does not have any traditional input method like in 'C' programming, we have scanf() function and in 'C++', we have cin object.

Java is truly object oriented, so everything is done with class/objects. System class inside the lag package encapsulates this in basic cases.

9.5.1 READING DATA FROM CONSOLE

To read the data from console, we have to import java.io package first, as those classes are available in java.io package.

We have to use two classes for this purpose.

1. InputStreamReader class
2. BufferedReader classs

InputStreamReader is the class which accepts the parameter which connect this input stream with source from where we have to accept the data. In this case we have to take the input from keyboard so we will specify System.in as parameter at the time of object creation.

General syntax for this object creation is-

```
InputStreamReader isr=new InputStreamReader(System.in);
```

Note that isr is the name of the object. It may be changed as per the need.

BufferedReader is the class which is connected to object of InputStreamReader class. At the time of object creation of BufferedReader, we have to specify the object of the InputStreamReader class.

General syntax for this will be-

```
BufferedReader br=new BufferedReader(isr);
```

Please note that br is the name of the object of BufferedReader class while isr is the name of the object of InputStreamReader that was already declared.

Object of InputStreamReader will accept the data from keyboard and handover it to object of BufferedReader class. Object of BufferedReader class than take this data and forward to the variable in which we wish to input the data.

Please make sure that java.io package is imported as both classes are given in that package only. To read the data we have to use the method readLine(). readLine() will input the data in string format. So we have to parse the data into the required format.

One more thing is to be taken care during this process that we also have to take care of exception handling. We have to care for IOException. This may be taken care by throws clause or via try/catch block, depending on the programmer.

Here is the syntax to read the data-

```
InputStreamReader isr=new InputStreamReader(System.in);
BufferedReader br=new BufferedReader(isr);
int x=Integer.parseInt(br.readLine());
floar y=Float.parseFloat(br.readLine());
double z=Double.parseDouble(br.readLine());
String str=br.readLine();
```

Example: Unit-9 (Program-01)

```
/* Write a program to perform the sum operation by taking input from keyboard*/
import java.io.*;
class unit09p01
```

```
{
    public static void main(String args[]) throws IOException
    {
        int x,y,z;
        InputStreamReader isr=new InputStreamReader(System.in);
        BufferedReader br=new BufferedReader(isr);
        System.out.print("Enter First number ");
        x=Integer.parseInt(br.readLine());
        System.out.print("Enter Second number ");
        y=Integer.parseInt(br.readLine());
        z=x+y;
        System.out.println("Sum="+z);
    }
}
```

Result :

Enter First number 12

Enter Second number 23

Sum=35

Please note that for input from keyboard, we have to take care of following steps-

1. import java.io package
2. use throws clause at the method for IOException
3. if throws clause is not used then use try/catch block for readLine() method
4. Create the object of InputStreamReader class with System.in as argument in constructor
5. Create the object of BufferedReader Class with object of InputStreamReader class as argument in constructor
6. One object of BufferedReader class is created, call the read Line() whenever input fom keyboard is required.
7. We must convert the data type as read Line() always return string value.

9.5.2 WRITING DATA TO CONSOLE

Console may be used to accept the data from keyboard and it may also be used to display the data. For that we have several methods. Some methods are-

System.out.write() to print integer

System.out.print() to print values in string format

System.out.println() to print values in string format with new line at the end

Example: Unit-9 (Program-02)

/* Write a program to implement the display methods of console. */

```
import java.io.*;
class unit09p02
{
    public static void main(String args[]) throws IOException
    {
        int x=65,y=97;
        System.out.write(x);
        System.out.write('\n');
        System.out.write(y);
         System.out.write('\n');
    }
}
```

Result:

A

a

Console may be printed with PrintWriter Class as well. General syntax for this is-

```
PrintWriter pw = new PrintWriter(System.out, true);
```

In the above example pw is the object of PrintWriter class and this name be changed. System.out is the destination where we have to put the data. We are displaying it on console so we have used System.out. true is used to mark for append. This is very useful when we are putting data in file. If we put false in case of file, it will erase the existing data and true will append the data without erasing the existing data.

Example: Unit-9 (Program-03)

/* Write a program to perform the sum of two integers and display with PrintWriter. */

```
import java.io.*;
class unit09p03
{
```

```
    public static void main(String args[]) throws IOException
    {
        int x=65,y=97,z;
        z=x+y;
        PrintWriter pw=new PrintWriter(System.out,true);
        pw.print("Value of x="+x);
        pw.print("\n");
        pw.println("Value of x="+y);
        pw.println("Sum="+z);
    }
}
```

Result:

Value of x=65

Value of x=97

Sum=162

Check Your Progress

1. What are the various ways to read the data from the keyboard?
2. Explain various methods to display data onto computer screen.
3. Write a program in java to perform the linear search with the help of console input and console output.
4. Write a program in java to perform the multiplication of two matrices with the help of console input and console output.

9.6 READING AND WRITING FILES

Java supports various ways to perform IO operations. We have already understand various ways to read and write data from source and to destination. This time we will write some data on the file and later on we will read the same data from different program. Writing data onto the files gives the permanency to data and it may be read anytime with the help of any java program. We must keep in mind that writing data on the file in java will enable the data to be read from java program. It's not possible to read the data of java program with the help of other programming language. And its vice-versa is also true. Java program can read the data from the flat file only if it is written with the java program. One thing we also have to take care that java provides a segment of classes to read and write the

data. If we write the data with one segment of class than we have to read the data from the same segment other segment will not work even in java programming.

9.6.1 WRITING DATA TO FILE

This time we are writing data onto the file with the help of FileOutputStream class. For that we have to create the object of FileOutputStream class. As a programmer, we must also remember the file name which is used to write the data because same name will be used to read or append the data.

There are some methods which will be used during data writing onto the file:

FileOutputStream("FileName")	Constructor used to create the file for writing data. File name used here is the name of the file to store the data.
write(int)	To store data to the file
close()	To close the file

Table 9.4

When we create the object of the FileOutputStream, it will automatically open the file. Once the file is open, we may write as much data as we need. After writing the complete data, we have to close the file. It means we have to open & close the file and between these methods we may write any amount of data on that file. One thing is to make sure that in this case, file must not be created before. It will create a new file to write the data.

Example: Unit-9 (Program-04)

/* Write a program to write 10 integers to the file. */

```
import java.io.*;
import java.util.*;

class unit09p04
{
    public static void main(String args[])
    {
        FileOutputStream fout;
        Scanner sc=new Scanner(System.in);
        int x,i;
```

```
            try
            {
                fout=new FileOutputStream("data.txt");
                for(i=1;i<=10;i++)
                {
                    System.out.print("Enter any number ");
                    x=sc.nextInt();
                    fout.write(x);
                }
                fout.close();
            }
            catch(IOException ex)
            {
                System.out.println(ex);
            }
            System.out.println("Data stored in the file successfully");
        }
    }
```

Result :

```
Enter any number 12
Enter any number 23
Enter any number 34
Enter any number 45
Enter any number 56
Enter any number 66
Enter any number 56
Enter any number 67
Enter any number 78
Enter any number 89
Data stored in the file successfully
```

The above program will create the file with the name "data.txt" and store the 10 integers in the file. After writing the data, file will be closed.

9.6.2 READING DATA FROM FILE

This time we are reading data onto the file with the help of FileInputStream class. For that we have to create the object of FileInputStream class. As a programmer, we must also remember the file name on which we have written the data. Same file has to be open here so that data may be reda.

There are some methods which will be used during data writing onto the file:

FileInputStream("FileName")	Constructor used to open the file for reading the data. File must be created with data so that it may be read.
read()	To read data to the file and return it
close()	To close the file

Table 9.5

When we create the object of the FileInputStream, it will automatically open the file. Once the file is open, we may read the data that is available in the file. After reading the required data, we have to close the file. It means we have to open & close the file and between these methods we may read any amount of available data on that file. One thing is to make sure that in this case, file must be created before.

Example: Unit-9 (Program-05)

/* Write a program to read 10 integers to the file and calculate their sum. */

```
import java.io.*;
import java.util.*;

class unit09p05
{
    public static void main(String args[])
    {
        FileInputStream fin;
        int x,i,s=0;
        try
        {
            fin=new FileInputStream("data.txt");
```

```
                    for(i=1;i<=10;i++)
                    {
                        x=fin.read();
                        s=s+x;
                        System.out.println("Number "+x);
                    }
                    System.out.println("Sum="+s);
                    fin.close();
                }
                catch(IOException ex)
                {
                    System.out.println(ex);
                }

        }
    }
```

Result :

```
Number 12
Number 23
Number 34
Number 45
Number 56
Number 66
Number 56
Number 67
Number 78
Number 89

Sum=526
```

The above program will open the file with the name "data.txt" and read the 10 integers in the file. After reading the data, it will calculate the sum of all numbers and display it. File will be closed after reading the data.

9.6.3 APPENDING DATA ON FILE

We have used FileOutputStream class to write the data on the file. Butif we have written data once and tried to write the data again, old data will be lost and new data will be written.

In such scenario, we have to append the data in the file so that existing data may not erase from the file and new data gets written at the end of existing data.

For this purpose same FileOutputStream class will be used. But at the time of object creation, we have to specify two parameter in the constructor. Second parameter will be a Boolean value true. This true will indicate the JVM that file has to be opened in append mode. So new data will be written at the end of existing data.

There are some methods which will be used during data writing onto the file:

FileOutputStream("FileName",true)	Constructor used to open file for writing the data in append mode. File must exist to append data.
write()	To write data to the file
close()	To close the file

Table 9.6

When we create the object of the FileInputStream, it will automatically open the file. Once the file is open, we may read the data that is available in the file. After reading the required data, we have to close the file. It means we have to open & close the file and between these methods we may read any amount of available data on that file. One thing is to make sure that in this case, file must be created before.

Example: Unit-9 (Program-06)

/* Write a program to append 10 integers on the file. */

```
import java.io.*;
import java.util.*;


class unit09p06
{
    public static void main(String args[])
    {
```

```
            FileOutputStream fout;
            Scanner sc=new Scanner(System.in);
            int x,i;
            try
            {
                  fout=new FileOutputStream("data.txt",true);
                  for(i=1;i<=10;i++)
                  {
                        System.out.print("Enter any number ");
                        x=sc.nextInt();
                        fout.write(x);
                  }
                  fout.close();
            }
            catch(IOException ex)
            {
                  System.out.println(ex);
            }
            System.out.println("Data appended in the file successfully");
      }
}
```

Result :

```
Enter any number 12
Enter any number 22
Enter any number 33
Enter any number 44
Enter any number 55
Enter any number 66
Enter any number 77
Enter any number 88
```

```
Enter any number 99
Enter any number 00
Data appended in the file successfully
```

The above program will open the file with the name "data.txt" and append the 10 integers in the file. After appending the data, it will close the file.

9.6.4 READING DATA FROM FILE TILL EOF

If we know the number of records in file, then we may use for loop at the reading time. But at times, we don't know the number of records stored in the file. In that case we may use the do-while or while loop. But to control this we have to check the EOF (End of File). This may be checked with the help of value returned by read(). If file reached to the end of filr, it will return -1. We may check this value to identify the EOF in the file.

There are some methods which will be used during data writing onto the file:

FileInputStream("FileName")	Constructor used to open the file for reading the data. File must be created with data so that it may be read.
read()	To read data to the file and return it
close()	To close the file

Table 9.7

Example: Unit-9 (Program-07)

/* Write a program to read all integers from the file and calculate their sum. */

```
import java.io.*;
import java.util.*;

class unit09p07
{
    public static void main(String args[])
    {
        FileInputStream fin;
        int x,i,s=0;
        try
```

```
        {
            fin=new FileInputStream("data.txt");
            x=fin.read();
            do
            {
                System.out.println("Number "+x);
                s=s+x;
                x=fin.read();
            }while(x!=-1);
            System.out.println("Sum="+s);
            fin.close();
        }
        catch(IOException ex)
        {
            System.out.println(ex);
        }

    }
}
```

Result :

```
Number 12
Number 23
Number 34
Number 45
Number 56
Number 66
Number 56
Number 67
Number 78
Number 89
Number 12
Number 22
Number 33
```

```
Number 44
Number 55
Number 66
Number 77
Number 88
Number 99
Number 0
Sum=1022
```

The above program will open the file with the name "data.txt" and read the all integers from the file. After reading the data, it will calculate the sum of all numbers and display it. File will be closed after reading the data.

Check Your Progress

1. What are the various ways to store data in the file?
2. Explain the method to read the data from file.
3. Write a program in java to store n integers in a file.
4. Write a java program to append some integers in the existing file.
5. Write a program to display all elements from file.

9.7 THE TRANSIENT AND VOLATILE MODIFIERS

There are two specialized keywords in Java. They are used in special situations. These keywords are transient and volatile.

transient keyword is a special keyword which declares the variable in some special manner. It adds a feature to the variable. This feature indicates the compiler that specific variable's value will not be persistent when an object is stored.

For example:

```
class test
{
        transient int x;
        int y;
}
```

Above example will indicate the compiler that object of class test will be stored in persistent storage area where value of y will be saved but value of x will not be saved.

volatile is again a special keyword. it indicates the compiler that if any variable is declared with volatile than it may be modified from any part of the program. These situations generally occurs in multithreading when multiple thread tries to access the same variable and want to edit its value. In such a scenarios we may declare the variable as volatile variable.

```
volatile int x;
```

In multithreading, when more than one thread tries to access the same instance variable. In such a scenarios, we may use the volatile keyword. It will increase the efficiency of the multithreaded application as JVM stores that volatile variable at a separate location.

9.7.1 USING INSTANCEOF

It may happen in some scenarios that we may want to identify the type of any object. Sometimes we lost the type of object type. This can be accessed with the help of instanceof operator. Yes, instanceof is the operator. It may be used compare the type of any object. At times, this is a very useful operator.

Example: Unit-9 (Program-08)

/* Write a program to create the objects of various classes and compare it with class name */

```
import java.io.*;
import java.util.*;


abstract class person
{ }
class student extends person
{ }
class emp extends person
{ }
class unit09p08
{
    public static void main(String args[])
    {
        person p;
        p=new student();
```

```
            if(p instanceof student)
            {
                System.out.println("This is the object of student");
            }
            else
            {
                System.out.println("This is the object of employee");
            }
            p=new emp();
            if(p instanceof student)
            {
                System.out.println("This is the object of student");
            }
            else
            {
                System.out.println("This is the object of employee");
            }

        }
    }
```

Result :

This is the object of student

This is the object of employee

Above example shows the use of instanceof operator.

9.7.2 NATIVE METHODS

Java is the programming language in which we may reuse the code of java in various formats. But if some code is written in any language other than java, than it becomes almost impossible to call that code. It is so because java compiler compiles the cod of the java only. It can't compile the code other than java.

This problem may be resolved via native method. If we wish to call a method that is not written in java than we must declare that method with the native keyword first. Then we have to load the library of that method at the runtime. This may be errorprone sometimes, but this is the only solution and it is very beneficial at times.

This may be beneficial when any library or API has to be called but that is not available in java.

Example: Unit-9 (Program-09)

/* Write a program to call any native method in java code*/

```
public class unit09p09
{
    int i;
    public static void main(String args[])
    {
        unit09p09 ob = new unit09p09();
        ob.i = 10;
        System.out.println("This is ob.i before the native method:" +ob.i);
        ob.test(); // call a native method
        System.out.println("This is ob.i after the native method:" +ob.i);
    }
    public native void test() ;
    static   // load DLL that contains static method
    {
        System.loadLibrary("NativeDemo");
    }
}
```

Above program contains static code. It means it will be executed before the start of main() and this code will load the "NativeDemo" library. If this library is not available then "java.lang.UnsatisfiedLinkError" exception will be generated.

There are some problems with the native methods. Very first is the security risk to our application. Second problem is the loss of portability as java code is portable but new code might not be portable.

Check Your Progress

1. Explain the transient and volatile Modifiers.
2. What is the use of instance of operator?
3. Write a program in java to check the type of an object against various classes.
4. What is native method? What are its drawbacks?
5. Write a program in java to call any native method in java program.

9.8 SUMMARY

Java is different in terms of input via keyboard in the way like we do in other programming languages. Here several methods are provided to accept the values from keyboard. Displaying data onto the computer screen is also provided with several methods in java. In fact java program identifies various sources for input data and various destination for output data. Input data may be keyboard, file, or any other source. In the same way all sources may be destination if we wish to store the data on such locations. Java also supports special modifiers like transient, volatile and native. Transient and volatile are modifiers for variables. Native is the modifier for the methods.

9.9 EXERCISE

Q. 1. What is stream in java?

Q. 2. Explain the types of streams of java in detail?

Q. 3. Explain them methodology to read the data from keyboard in java.

Q. 4. Explain the various classes to display data on console.

Q. 5. Explain the various method to store and read data from file.

Q. 6. What are the predefined streams? Explain them.

Q. 7. Write a program to count the occurrence of any number in the file.

Q. 8. Write a program in java to count and display all even number from a file.

Q. 9. Write a program in java to find the average of stored numbers in java. Keep in mind that number of elements are not known at the start of program.

Q. 10. Write a program to accept the number from keyboard in array. Store all array elements in the file.

Q. 11. Write a program to read the elements from file and store them in array. Sort the array elements and display them.

UNIT-10 STRINGS AND CHARACTERS

Structure :

10.1 OBJECTIVE

There are some built-in data types in compiler, also known as primitive data types. In java there are numerical types (byte, short, int, long, float, double), Boolean type, and character data type (char). There is no string data type as primitive data type. As we know that in C/C++ there is character array for the string manipulation which was terminated with NULL character.

Java also supports character array but for string manipulation there is a separate data type known as String. This is one of the most promising type in java for string manipulation.

10.2 INTRODUCTION

There are some built-in data types in compiler, also known as primitive data types. In java there are numerical types (byte, short, int, long, float, double), Boolean type, and character data type (char). There is no string data type as primitive data type. As we know that in C/C++ there is character array for the string manipulation which was terminated with NULL character.

Java also supports character array but for string manipulation there is a separate data type known as String. This is one of the most promising type in java for string manipulation.

10.3 FUNDAMENTAL OF CHARACTERS AND STRINGS

String is the sequence of characters enclosed within "". Java implicitly imports java.lang package. In this package, there is a String class. This class is

used for String manipulations. Unlike C/C++, string is not terminated with NULL character in java.

One more difference is there in the string of java. Java supports UNICODE rather than ASCII and we know UNICODE is 16 bit code here. So each character in java consumes 2 bytes. We should remember that earlier in C/C++, one character data type consumes 1 byte for 1 character.

Java also have one more type for String that is known as StringBuffer. If we store the string in the object of String class than it is unchangeable (Immutable). It means if we edit the value of string object than, it will not edit the old string, it will create the new string in the memory and assign the new string reference to the object. Garbage collection will clear the old string from the memory. To resolve this issue, java supports StringBuffer class. The object of StringBuffer class is editable (mutable). We may modify the value of StringBuffer class without creating new string. But from the end user view point, garbage collection utility of java also resolves the issue of String object by clearing old value of String Object.

StringBuilder is one more class, which may be used to create the String in java. StringBuilder is also the mutable (editable) class.

Check Your Progress

1. What is String in java?
2. Compare string of java and string of C/C++.
3. Why there is need of 16 bits UNICODE in java when C/C++ are working on 8 bits ASCII code.

10.4 THE STRING CLASS

Java provides the string class to create and manage the string in the java. Object of String class is non-editable in java. Whenever we edit the object of string class, JVM creates a new string in the memory and assign its address to string object. Old string will be cleared by garbage collection utility.

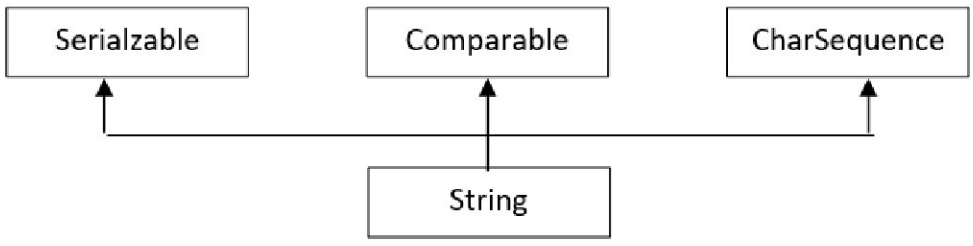


Figure 10.1

Sting class implements three interfaces- Serialzable, Comparable and CharSequence. The CharSequence interface is used to represent the sequence of characters.

There are several ways to create the sting in java. There are several constructor to create the sting. For example-

Sting str=new String(“Ankur Mittal”); //by using String in constructor
String str=”Ankur Mittal” // by assigning the string
String str; str=new String(“Ankur Mittal”); //using the String in constructor
char[] ch={‘A’,’N’,’K’,’U’,’R’}; String str=new String(ch); //using character array

byte[] bt={65,17,11,21,18}; // ASCII values of alphabets
String str=new String(bt); // with the help of byte array

Example: Unit-10 (Program-01)

/* Write a program display string value with various types*/

```
public class unit10p01
{
    public static void main(String args[])
    {
            String str1="Ankur Mittal";
            String str2=new String("Ankur Mittal");
            String str3;
            str3=new String("Ankur Mittal");  //using the String in
constructor
            char[] ch={'A','N','K','U','R'};
            String str4=new String(ch);  //using character array
            byte[] bt={65,78,75,85,82};  // ASCII values of alphabets
            String str5=new String(bt);  // with the help of byte array
            System.out.println(str1);
            System.out.println(str2);
```

```
            System.out.println(str3);
            System.out.println(str4);
            System.out.println(str5);
        }
    }
```

Result :

Ankur Mittal

Ankur Mittal

Ankur Mittal

ANKUR

ANKUR

Check Your Progress

1. What is the hierarchy of string class in java?
2. Compare String and character array in java.
3. Write a program in java to accept the string in various formats and display them.

10.5 STRING OPERATIONS

Here are some of the methods which are provided in the string class for string manipulation.

S.no	Method	Purpose
1.	char charAt(int index)	returns char value for the particular index
2.	int length()	returns string length
3.	String substring(int beginIndex)	returns substring for given begin index.
4.	String substring(int beginIndex, int endIndex)	returns substring for given begin index and end index.

5.	boolean contains(CharSequence s)	returns true or false after matching the sequence of char value.
6.	boolean equals(Object another)	checks the equality of string with the given object.
7.	boolean isEmpty()	checks if string is empty.
8.	String concat(String str)	concatenates the specified string.
9.	String replace(char old, char new)	replaces all occurrences of the specified char value.
10.	static String equalsIgnoreCase(String another)	compares another string. It doesn't check case.
11.	String[] split(String regex)	returns a split string matching regex.
12.	int indexOf(int ch)	returns the specified char value index.
13.	int indexOf(int ch, int fromIndex)	returns the specified char value index starting with given index.
14.	int indexOf(String substring)	returns the specified substring index.
15.	int indexOf(String substring, int fromIndex)	returns the specified substring index starting with given index.
16.	int lastIndexOf(int ch)	To search for the last occurrence of a character
17.	int lastIndexOf(String str)	To search for the last occurrence of a string
18.	String toLowerCase()	returns a string in lowercase.
19.	String toUpperCase()	returns a string in uppercase.
20.	String trim()	removes beginning and ending spaces of this string.
21.	static String valueOf(int value)	converts given type into string. It is an overloaded method.
22.	boolean startsWith(String str)	determines whether a given String begins with a specified string

23.	boolean endsWith(String str)	determines whether a given String ends with a specified string
24.	int compareTo(String str)	Compares two strings and returns- <0: The invoking string is less than str. >0: The invoking string is greater than str. =0: The two strings are equal.

Table 10.1

Example: Unit-10 (Program-02)

/* Write a program to perform basic operations on string in java.*/

```
import java.util.*;
public class unit10p02
{
    public static void main(String args[])
    {
        String str;
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter the string ");
        str=sc.nextLine();
        System.out.println("length of String : "+ str.length());
        System.out.println("Lower case of string : "+ str.toLowerCase());
        System.out.println("Upper case : "+ str.toUpperCase());
        System.out.println("Delimeted String : "+ str.trim());
    }
}
```

Result:

Enter the string Ankur MITTAL

length of String : 15

Lower case of string : ankur mittal

```
Upper case :    ANKUR MITTAL
Delimeted String : Ankur MITTAL
```

Example: Unit-10 (Program-03)

/* Write a program to compare and concatenate strings in java.*/

```
import java.util.*;
public class unit10p03
{
    public static void main(String args[])
    {
        String str1, str2;
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter first string ");
        str1=sc.nextLine();
        System.out.print("Enter Second string ");
        str2=sc.nextLine();
        if(str1.equals(str2))
        {
            System.out.println("Both strings are equal");
        }
        else
        {
            System.out.println("Both strings are not equal");
        }
        if(str1.equalsIgnoreCase(str2))
        {
            System.out.println("Both strings are equal");
        }
        else
        {
            System.out.println("Both strings are not equal");
        }
        str1=str1.concat(str2);
```

```
            if(str1.isEmpty())
            {
                System.out.println("New String is empty.");
            }
            else
            {
                System.out.println("New String is not empty.");
            }
                    System.out.println("First String : "+ str1);
            System.out.println("Second String : "+ str2);
        }
    }
```

Result:

```
Enter first string Ankur
Enter Second string ANKUR
Both strings are not equal
Both strings are equal
New String is not empty.
First String : AnkurANKUR
Second String : ANKUR
```

Example: Unit-10 (Program-04)

/* Write a program to print the short name of the user*/

```
import java.util.*;

public class unit10p04
{
    public static void main(String args[])
    {
        String str;
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter your Full Name ");
```

```
            str=sc.nextLine();
            char ch;
            System.out.print("\nShort Name of user : ");
            for(int i=0;i<str.length();i++)
            {
                  ch=str.charAt(i);
                  if(i==0 || str.charAt(i-1)==' ')
                        System.out.print(ch+".");
            }
      }
}
```

Result :

Enter your Full Name Ankur Mittal

Short Name of user : A.M.

Example: Unit-10 (Program-05)

/* Write a program to search the string in main string. */

```
import java.util.*;
public class unit10p05
{
      public static void main(String args[])
      {
            String str1, str2;
            Scanner sc=new Scanner(System.in);
            System.out.print("Enter first string ");
            str1=sc.nextLine();
            System.out.print("Enter Second string to search ");
            str2=sc.nextLine();
            int flag=str1.indexOf(str2);
            if(flag>=0)
            {
```

```
                    System.out.println("String found at "+flag +" position");
            }
            else
            {
                    System.out.println("String not found");
            }
        }
    }
```

Result :

Enter first string Ankur Mittal

Enter Second string to search Mittal

String found at 6 position

10.5.1 DATA CONVERSION USING VALUEOF( ) METHODS

We have gone through indexOf() and charAt() methods which are used to compare and extract with respect to string respectively.

If we wish to convert any value to string, than we have a method, toString( ). toSting() method will convert non-string value to string type and return it.

For example:

```
Sting x="10";
int a=Integer.parseInt(x);
```

If there is a requirement to convert values into String type than there is a method available called valueOf(). This method will conver non-string value to string type.

For example:

```
int x=10;
String str=String.valueOf(x);
```

Example: Unit-10 (Program-06)

/* Write a program to perform the sum of two numbers and store the result in string. */

```
import java.io.*;
public class unit10p06
{
    public static void main(String args[]) throws IOException
    {
        String str;
        int x,y,z;
        InputStreamReader isr=new InputStreamReader(System.in);
        BufferedReader br=new BufferedReader(isr);
        System.out.print("Enter two numbers ");
        x=Integer.parseInt(br.readLine());    // convert string to int
        y=Integer.parseInt(br.readLine());
        z=x+y;
        str=String.valueOf(z);                // convert int to string
        System.out.print("Sum "+str);
    }
}
```

Result :

```
Enter two numbers 12
34
Sum 46
```

Check Your Progress

1. List the basic operations provided by java in String class.
2. Write a program in java to convert the string into proper case. Proper case means first alphabet of each word should be converted to upper case and all other alphabets must be converted to lower case.
3. Write a program in java to count the vowels in string.
4. Write a program in java to reverse the string without the use of library class.
5. Write a program in java to replace the word in the string with new word without library function.
6. Write a program in java to count the number of words in string.

10.6 STRINGS BUFFER AND METHODS

StringBuffer is the mutable class in java. Mutable means string may be modified and after modification, it will not consume separate memory after modification.

Some methods used in StringBuffer class are as given-

Method	Purpose
StringBuffer append (String s)	is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
StringBuffer insert (int offset, String s)	is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
StringBuffer replace (int startIndex, int endIndex, String str)	is used to replace the string from specified startIndex and endIndex.
StringBuffer delete (int startIndex, int endIndex)	is used to delete the string from specified startIndex and endIndex.
StringBuffer reverse()	is used to reverse the string.
int capacity()	is used to return the current capacity.
void ensureCapacity (int minCapacity)	is used to ensure the capacity at least equal to the given minimum.
char charAt(int index)	is used to return the character at the specified position.
int length()	is used to return the length of the string i.e. total number of characters.
void setLength(int len)	To set the length of the buffer within a StringBuffer object
String substring (int beginIndex)	is used to return the substring from the specified beginIndex.
String substring (int beginIndex, int endIndex)	is used to return the substring from the specified beginIndex and endIndex.

Table 10.2

There are some more methods available in StringBuffer class as compared to String class. It gives an easy access to string manipulation rather to do that with the help of self-programming.

Example: Unit-10 (Program-07)

/* Write a program to perform operations on StringBuffer object. */

```
import java.util.*;
public class unit10p07
{
    public static void main(String args[])
    {
        StringBuffer str1,str2,str3;
        int x,y;
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter the string ");
        str1=new StringBuffer(sc.nextLine());
        System.out.println("Capacity of String "+str1.capacity());
        str1.ensureCapacity(50);
        System.out.println("Minimum Capacity of String: "+str1.capacity());
        System.out.println("Length of String: "+str1.length());
        System.out.print("Enter the new string to append: ");
        str2=new StringBuffer(sc.nextLine());
        System.out.println("Before appending String: "+str1);
        str1.append(str2);
        System.out.println("New String "+str1);
        System.out.print("Enter the new string to insert: ");
        str2=new StringBuffer(sc.nextLine());
        System.out.print("Enter the position to insert: ");
        x=sc.nextInt();
        str1=str1.insert(x,str2);
```

```
                System.out.println("New String after insert: "+str1);
                System.out.print("Enter the position to begin deletion: ");
                x=sc.nextInt();
                System.out.print("Enter the position to end deletion ");
                y=sc.nextInt();
                System.out.println("String for deletion is: "+str1.substring(x,y));
                str1=str1.delete(x,y);
                System.out.println("New String after delete: "+str1);
                str1=str1.reverse();
                System.out.println("Reversed String: "+str1);
        }
    }
```

Result :

```
Enter the string Ankur
Capacity of String 21
Minimum Capacity of String: 50
Length of String: 5
Enter the new string to append:  Mittal
Before appending String: Ankur
New String Ankur Mittal
Enter the new string to insert: Er.
Enter the position to insert: 0
New String after insert: Er.Ankur Mittal
Enter the position to begin deletion: 0
Enter the position to end deletion 2
String for deletion is: Er
New String after delete: .Ankur Mittal
Reversed String: lattiM ruknA.
```

In the above Example, we have implemented approx. all StringBuffer class methods for string manipulation.

Check Your Progress

1. What is StringBuffer class in java?
2. Compare String with StringBuffer class.
3. Write a program in java to replace one word with new word in java with stringbuffer class.
4. Write a program in java to find the word in string and display its number of existances.
5. Write a program in java to insert a new string in the existing string with the help of StringBuffers.
6. Write a program in java to remove all existence of any specific word in the string with the help of StringBuffer class.

10.7 SUMMARY

String is one of the most powerful data type in java as it was not available in C/C++. Java provides three classes to manage the string in java- String, StringBuffer and StringBuilder classes. All three classes are very powerful classes with predefined library associated with it.

String class is immutable and StringBuffer is mutable class. It means String class can't be modified at its source. If we try to edit it, new memory will be allocated for that. On the other hand, StringBuffer is mutable class, it means it may be modified at its origin. It is the library of the String and StringBuffer which makes it more powerful.

10.8 EXERCISE

Q. 1. What String in java? Compare String with StringBuffer.

Q. 2. Write a program in java to reverse the words in the string.

For Example: INPUT STRING: ANKUR MITTAL

OUTPUT STRING: RUKNA LATTIM

Q. 3. Write a program in java to reverse the string word by word.

For Example: INPUT STRING: ANKUR MITTAL

OUTPUT STRING: MITTAL ANKUR

Q. 4. Write a program in java to print the name of the user in proper case with full last name.

For Example: INPUT STRING: ANKUR KUMAR MITTAL

OUTPUT STRING: A. K. MITTAL

Q. 5. Write a program in java to encrypt the string with a key. Key is the numerical value. This key is added in the ascii value of each alphabet to produce the new string.

For Example: INPUT STRING: ANKUR

INPUT KEY: 2

OUTPUT STRING: CPMTW

Q. 6. Write a program in java to check for palindrome string.

Q. 7. Write a program in java to insert the new word in the string after any specific word.

For Example: INPUT STRING: ANKUR MITTAL

INPUT NEW WORD TO ADD: KUMAR

INPUT NEW WORD AFTER WHICH TO ADD: ANKUR

OUTPUT STRING: ANKUR KUMAR MITTAL

Q. 8. Write a program in java to reverse the string without library function.

Q. 9. Write a program in java to exchange the first and last alphabet of each word in the string.

For Example: INPUT STRING: ANKUR MITTAL

OUTPUT STRING: RNKUA LITTAM

Q. 10. Write a program in java to arrange the string on the basis of ASCII value of the alphabets.

UNIT-11 EXPLORING JAVA I/O

Structure :

11.1 OBJECTIVE

Java I/O (Input and Output) is **used** to process the input and produce the output. By using the concept of stream **Java** make I/O operation fast. The **java.io** packages have different classes which are used for performing input and output operations. Although file handling in Java can be performed by **Java** I/O API

11.2 INTRODUCTION

In all programming languages, it is mandatory to access the external data in the program. Without that, managing complex problems are almost impossible. In fact if we are accessing data via keyboard, it is also the form of accessing external data. In nutshell we may assume that anything that is not generated by the program is the outside data. It may be asked first time for the user. It may already be input by the user and stored at any place, even than we have to access

the data from that location. In some cased we may need the data from the data bases or any other external device.

Data is input from an input source. In the same way data is produced for an output destination. In java, there are several sources and destinations. Source or destination may be a memory buffer, network connection, disk, device, file etc. There may be any case for external data but there is a common abstraction for that and it is known as stream in java.

11.3 JAVA I/O CLASSES AND INTERFACES

There are several Stream classes defined in the java.io package:

classes in java.io package	
BufferedInputStream	BufferedOutputStream
BufferedReader	BufferedWriter
ByteArrayInputStream	ByteArrayOutputStream
CharArrayReader	CharArrayWriter
DataInputStream	DataOutputStream
FileInputStream	FileOutputStream
FileReader	FileWriter
FilterInputStream	FilterOutputStream
FilterReader	FilterWriter
InputStream	InputStreamReader
ObjectInputStream	ObjectOutputStream
ObjectStreamClass	ObjectStreamField
OutputStream	OutputStreamWriter
LineNumberReader	FilePermission
File	FileDescriptor

Table 11.1

There are some interfaces defined in the java.io package as well-

interface of java.io packages	
DataInput	DataOutput
ObjectInput	ObjectOutput
Externalizable	Serializable
FileFilter	FilenameFilter
ObjectInputValidation	

Table 11.2

11.3.1 FILE IN JAVA.IO PACKAGE

File is the main source of data for sources and destination in programming. Data stored on the disk is managed with file by operating system.

Most of the classes defined in the java.io package are streams but file is not a stream. File class in the java.io package is used to read the properties of any file and some properties may be altered as well. At a time one object File class reads the information of one file such as creation date & time, permissions, path etc. File does not specify the way how data is stored in the file or how data is read from the file. But it contains the properties of the file.

To create a file, there are several constructors-

File(String directoryPath)	// specifies the path of file/directory
File(String directoryPath, String filename)	// specifies the file at specific path
File(File dirObj, String filename)	// represents file object with a name

Director path is the location of the folder. In windows directory is known as folder. File may create with a file name or other file objects.

Methods of the File class-

FILE CLASS METHODS	
Method	**Purpose**
boolean canExecute()	Tests whether the application can execute the file denoted by this abstract pathname.
boolean canRead()	Whether the application can read the file denoted by this abstract pathname.

boolean canWrite()	Tests whether the application can modify the file denoted by t abstract pathname.
boolean exists()	Whether the file or directory denoted by this abstract pathna exists.
String getAbsolutePath()	Returns the absolute pathname string of this abstract pathname.
String getName()	Returns the name of the file or directory denoted by this abstr pathname.
String getParent()	Returns the pathname string of this abstract pathname's parent.
File getParentFile()	Returns the abstract pathname of this abstract pathname's parent
String getPath()	Converts this abstract pathname into a pathname string.
boolean isDirectory()	Tests whether the file denoted by this pathname is a directory.
boolean isFile()	Tests whether the file denoted by this abstract pathname is normal file.
boolean isHidden()	Whether the file named by this abstract pathname is a hidden file
long length()	Returns the length of the file denoted by this abstract pathname.
String[] list()	Returns an array of strings naming the files and directories in t directory.
File[] listFiles()	Returns an array of abstract pathnames denoting the files in t directory.
boolean createNewFile()	Atomically creates a new, empty file named by this abstr pathname.
boolean delete()	Deletes the file or directory denoted by this abstract pathname.
boolean renameTo(File dest)	Renames the file denoted by this abstract pathname.
boolean setExecutable (boolean executable)	A convenience method to set the owner's execute permission.
boolean setReadable	A convenience method to set the owner's read permission.

(boolean readable)	
boolean setReadOnly()	Marks the file or directory named so that only read operations are allowed.
boolean setWritable (boolean writable)	A convenience method to set the owner's write permission.
int compareTo (File pathname)	Compares two abstract pathnames lexicographically.
boolean mkdir()	Creates the directory named by this abstract pathname.
boolean equals(Object obj)	Tests this abstract pathname for equality with the given object.
long getFreeSpace()	Returns the number of unallocated bytes in the partition.
String toString()	Returns the pathname string of this abstract pathname.

Table 11.3

Above are the basic methods of File class. At times these are very useful methods.

Example: Unit-11 (Program-01)

/* Write a program display the properties of any file. */

```
import java.io.*;
public class unit11p01
{
    public static void main(String args[])
    {
        File f1 = new File("/javap/unit11p01.java");
        System.out.println("File Name: " + f1.getName());
        System.out.println("Path of file: " + f1.getPath());
        System.out.println("Absolute Path: " + f1.getAbsolutePath());
        System.out.println("Parent directory: " + f1.getParent());
        System.out.println("This exists : "+f1.exists());
        System.out.println("This may be write : "+f1.canWrite());
```

```
            System.out.println("This may be read : "+f1.canRead());
            System.out.println("This may be executed : "+f1.canExecute());
            System.out.println("This may be Directory : "+f1.isDirectory());
            System.out.println("This is a normal file : "+f1.isFile());
            System.out.println("This is Hidden : "+f1.isHidden());
            System.out.println("This is absolute : "+f1.isAbsolute());
            System.out.println("File last modified on: " + f1.lastModified());
            System.out.println("File size: " + f1.length() + " Bytes");
        }
    }
```

Result :

File Name: unit11p01.java

Path of file: \javap\unit11p01.java

Absolute Path: C:\javap\unit11p01.java

Parent directory: \javap

This exists : true

This may be write : true

This may be read : true

This may be executed : true

This may be Directory : false

This is a normal file : true

This is Hidden : false

This is absolute : false

File last modified on: 1577103944312

File size: 1005 Bytes

11.3.2 DIRECTORY

In java File class is also used for directory. In this case we have to specify the name of the directory along with the path of directory. From this directory we may get the directory properties and the files inside this directory.

Example: Unit-11 (Program-02)

/* Write a program display the properties of any directory*/

```
import java.io.*;


public class unit11p02
{
    public static void main(String args[])
    {
            File f1 = new File("/javap/");
            System.out.println("Directory Name: " + f1.getName());
            System.out.println("Path of directory: " + f1.getPath());
            System.out.println("Absolute Path: " + f1.getAbsolutePath());
            System.out.println("Parent directory: " + f1.getParent());
            System.out.println("This exists : "+f1.exists());
            System.out.println("This may be write : "+f1.canWrite());
            System.out.println("This may be read : "+f1.canRead());
            System.out.println("This may be executed : "+f1.canExecute());
            System.out.println("This may be Directory : "+f1.isDirectory());
            System.out.println("This is a normal file : "+f1.isFile());
            System.out.println("This is Hidden : "+f1.isHidden());
            System.out.println("This is absolute : "+f1.isAbsolute());
            System.out.println("File last modified on: " + f1.lastModified());
            System.out.println("File size: " + f1.length() + " Bytes");
    }
}
```

Result :

```
Directory Name: javap
Path of directory: \javap
Absolute Path: C:\javap
Parent directory: \
```

```
This exists : true
This may be write : true
This may be read : true
This may be executed : true
This may be Directory : true
This is a normal file : false
This is Hidden : false
This is absolute : false
File last modified on: 1577104481635
File size: 98304 Bytes
```

Example: Unit-11 (Program-03)

/* Write a program display the files inside any directory*/

```
import java.io.*;
public class unit11p03
{
    public static void main(String args[])
    {
        File f1 = new File("/javap/");
        String fn[]=f1.list();
        int count=fn.length;
        System.out.println("Total files in the directory ="+count);
        for(int i=0;i<count;i++)
        {
            System.out.println("File Name:" + fn[i]);
        }
    }
}
```

Result :

```
Total files in the directory =217
File Name:AgeException.class
```

File Name:BaseThreadSeries.class

File Name:CalculateArea.class

File Name:Circle.class

:

:

File Name:unit11p03.class

File Name:unit11p03.java

Check Your Progress

1. What is file? Explain the File class of java.
2. List the classes and interfaces of java IO package.
3. Write a program in java to list the java files in any given directory.
4. Write a program in java to find the name and size of the largest file in any folder.
5. Write a program in java to arrange the file names of any folder on the basis of their size.
6. Write a program in java to create a new directory.
7. Write a program in java to delete any existing file of a folder.
8. Write a program in java to set a file to read only mode.

11.4 STREAM CLASSES

As we know, Stream is a logical entity that either produces or consumes information. At source, it produces information but at destination, it consumes information. There are several stream classes in java. They are used for input and output function.

There are two types of streams in java

- Byte stream and
- Character stream

Byte stream process on byte at a time while character stream process one character (2 bytes) at a time.

Byte Stream classes give two abstract classes- InputStream and OutputStream class

Character stream classes give two abstract classes- Reader and Writer classes

Character stream classes work on character (text data) while byte stream classes work on binary (bytes).

Byte Stream vs. Character stream		
Sno.	**Byte Stream**	**Character Stream**
1	Process 8 bits at a time	Process 16 bits at a time
2	Abstract classes are InputStream and OutputStream	Abstract classes are Reader and Writer
3	Used for working in character or string	Used for working in bytes or binary objects

Table 11.4

Here is the list of Methods of InputStream Class-

Methods of InputStream Class	
Method	**Purpose**
int available( )	Returns the number of bytes of input currently available for reading.
void close( )	Closes the input source. Further read attempts will generate an IOException.
void mark(int numBytes)	Places a mark at the current point in the input stream that will remain valid until numBytes bytes are read.
boolean markSupported ( )	Returns true if mark( )/reset( ) are supported by the invoking stream.
int read( )	Returns an integer representation of the next available byte of input. –1 is returned when the end of the file is encountered.
int read(byte buffer[ ])	Attempts to read up to buffer.length bytes into buffer and returns the actual number of bytes that were successfully read. –1 is returned when the end of the file is encountered.

int read(byte buffer[ ], int offset, int numBytes)	Attempts to read up to numBytes bytes into buffer starting at buffer[offset], returning the number of bytes successfully read. –1 is returned when the end of the file is encountered.
void reset( )	long skip(long numBytes)
long skip(long numBytes)	Ignores (that is, skips) numBytes bytes of input, returning the number of bytes actually ignored.

Table 11.5

Here is the list of Methods of OutputStream Class-

Methods of OutputStream Class	
Method	**Purpose**
void close( )	Closes the output stream. Further write attempts will generate an IOException.
void flush( )	Finalizes the output state so that any buffers are cleared. That is, it flushes the output buffers.
void write(int b)	Writes a single byte to an output stream. Note that the parameter is an int, which allows you to call write( ) with expressions without having to cast them back to byte.
void write(byte buffer[ ])	Writes a complete array of bytes to an output stream.
void write(byte buffer[ ], int offset, int numBytes)	Writes a subrange of numBytes bytes from the array buffer, beginning at buffer[offset].

Table 11.6

We have already managed data with FileInputStream and FileOutputStream classes. So we will now manage the data with some other stream classes.

11.4.1 TEXT STREAMS

Text Streams are used to process the data in text formats. There are several classes to perform this operations.

11.4.2 FILEARRAYOUTPUTSTREAM CLASS

File ArrayOutputStream class is the sub class of OutputStream class. OutputStreamStream is an abstract class so when we create the object of OutputStream class, we have to allocate the memory with FileArrayOutputStream class. FileOutputStream class uses byte array as destination.

Example: Unit-11 (Program-04)

/* Write a program to store the data with object of ByteArrayOutputStream class*/

```
import java.io.*;
public class unit11p04
{
    public static void main(String args[]) throws IOException
    {
            String str="Ankur Mittal is the student of computer science";
            byte[] buff=str.getBytes();

            ByteArrayOutputStream baos=new ByteArrayOutputStream();
            baos.write(buff);
            OutputStream os=new FileOutputStream("data1.txt");
            baos.writeTo(os);
            os.close();
            System.out.println("Data stored in file");
            baos.reset();
    }
}
```

Result :

Data stored in file

Above program will write the data in the file in text format.

Example: Unit-11 (Program-05)

/* Write a program to read the data with object of ByteArrayInputStream class*/

```
import java.io.*;
public class unit11p05
```

```
{
    public static void main(String args[]) throws IOException
    {
        InputStream is=new FileInputStream("data1.txt");
        byte[] buff=new byte[is.available()];
        is.read(buff);
        is.close();
        ByteArrayInputStream bais=new ByteArrayInputStream(buff);
        System.out.println("Data read from file");
        int x;
        while((x=bais.read())!=-1)
            System.out.print((char)x);
    }
}
```

Result :

Data read from file

Ankur Mittal is the student of computer science

11.4.3 BUFFERED STREAM CLASS

In the byte oriented stream, a buffered stream extends a filtered stream class by attaching a memory buffer to the I/O streams. With the help of this buffer, multiple bytes may be processed at a time. So performance will be increased. As we have the buffers so we may also perform skipping, marking and resetting operations in stream.

There are two buffer stream classes-

BufferOutputStream class and;

BufferInputStream class

BufferedOutputStream class is used to write data to buffers. It is similar to any OutputStream class. But it has additionally flush() method. This flush() method makes sure that buffers are physically written to output device. This class reduces the number of times to write the data. With the flush() method all data is written to the output device at a time. It improves efficiency of the program.

There are two constructor in BufferedOutputStream class-

```
BufferedOutputStream(OutputStrea obj)
BufferedOutputStream(OutputStrea obj, int bufferSize)
```

Default buffer size if 512 bytes in java (for the first constructor).

Example: Unit-11 (Program-06)

/* Write a program to store data with object of bufferedOutputStream class*/

```
import java.io.*;
import java.util.*;

public class unit11p06
{
    public static void main(String args[]) throws IOException
    {
        FileOutputStream fos=new FileOutputStream("data2.txt");
        BufferedOutputStream bos=new BufferedOutputStream(fos);

        Scanner sc=new Scanner(System.in);
        int x,i;
        for(i=1;i<10;i++)
        {
            System.out.print("Enter any number ");
            x=sc.nextInt();
            bos.write(x);
        }
        bos.flush();
        bos.close();
        fos.close();
        System.out.println("Data stored in file");
    }
}
```

Result :

Enter any number 12

```
Enter any number 23
Enter any number 45
Enter any number 43
Enter any number 44
Enter any number 55
Enter any number 66
Enter any number 77
Enter any number 88
Data stored in file
```

BufferedInputStream class is generally used to improve the performance of the stream. It may be bundled with any InputStream class. Like BufferedOutputStream class, there are constructor for this class-

```
BufferedInputStream(OutputStrea obj)
BufferedInputStream(OutputStrea obj, int bufferSize)
```

Default buffer size if 512 bytes in java (for the first constructor).

With this stream class, low-level system can read blocks of data from disk/network and store the results in provided buffer. Buffering InputStream also provides the moving towards backward in the stream. This class provides the read(), mark(), skip(), reset() method. These four methods increase the performance of program.

Example: Unit-11 (Program-07)

/* Write a program to read data with object of bufferedInputStream class*/

```
import java.io.*;
public class unit11p07
{
    public static void main(String args[]) throws IOException
    {
        FileInputStream fis=new FileInputStream("data2.txt");
        BufferedInputStream bis=new BufferedInputStream(fis);

        System.out.println("Total Data elements in file: "+bis.available());
        int x=bis.read();
```

```
                do
                {
                        System.out.println(x);
                        x=bis.read();
                }while(x!=-1);
                bis.close();
                fis.close();
        }
  }
```

Result:

Total Data elements in file: 9

12

23

45

43

44

55

66

77

88

11.4.4 PRINT STREAM CLASS

The PrintStream class provides methods to write data to another stream without calling flush() method. Unlike other output streams, a PrintStream never throws an IOException. All characters printed by a PrintStream are converted into bytes. It provides all of the formatting capabilities of System.out.

Constructors of PrintStream class are :

Constructors of PrintStream class	
Constructor	**Purpose**
PrintStream(File file)	Creates a new print stream, without automatic line

	flushing, with the specified file.
PrintStream(File file, String csn)	Creates a new print stream, without automatic line flushing, with the specified file and charset.
PrintStream(OutputStream out)	Creates a new print stream.
PrintStream(OutputStream out, boolean autoFlush)	Creates a new print stream.
PrintStream(OutputStream out, boolean autoFlush, String encoding)	Creates a new print stream.
PrintStream(String fileName)	Creates a new print stream, without automatic line flushing, with the specified file name.
PrintStream(String fileName, String csn)	Creates a new print stream, without automatic line flushing, with the specified file name and charset.

Table 11.7

There are some methods of PrintStream Class-

Methods of PrintStream Class	
Constructor	**Purpose**
void print(boolean b)	Prints a boolean value.
void print(char c)	Prints a character.
void print(char[] s)	Prints an array of characters.
void print(double d)	Prints a double-precision floating-point number.
void print(float f)	Prints a floating-point number.
void print(int i)	Prints an integer.
void print(long l)	Prints a long integer.

void print(Object obj)	Prints an object.
void print(String s)	Prints a string.
void close()	Closes the stream
void flush()	Flushes the stream
boolean checkError()	Flushes the stream and checks its error state.
protected void clearError()	Clears the internal error state of this stream.
PrintStream append (char c)	Appends the specified character to this output stream.
PrintStream append (CharSequence csq, int start, int end)	Appends the specified character sequence to this output stream.
PrintStream format (Locale l, String format, Object… args)	Writes a formatted string to this output stream using the specified format string and arguments.
PrintStream format (String format, Object… args)	Writes a formatted string to this output stream using the specified format string and arguments.

Table 11.8

Example: Unit-11 (Program-08)

/* Write a program to write data in file with object of PrintStream class*/

```
import java.io.*;
public class unit11p08
{
    public static void main(String args[]) throws IOException
    {
        FileOutputStream fos=new FileOutputStream("data3.txt");
        PrintStream ps=new PrintStream(fos);
        ps.println("Ankur Mittal");    // writing string
```

```
            ps.println(2019);              //writing integer
            ps.println(false);             //writing boolean
            ps.println(25.23f);            //writing float
            ps.println('X');          //writing character
            ps.close();
            fos.close();
            System.out.println("Data Stored in the file successfully");
        }
    }
```

Result :

Data Stored in the file successfully

Above program will store the data in the file. File may be checked/read anytime.

11.4.5 STREAM TOKENIZER

StreamTokenizer class is one of the specialised class in java.io package. It connects with InputStream and parsed data into tokens. Tokens may be read one at a time. This StreamTokenizer class can identify numbers, identifiers, quoted string and various other styles.

This is the list of fields of StreamTokenizer class-

Fields of StringTokenizer class	
Field name	**Purpose**
double nval	If the current token is a number, this field contains the value of that number.
String sval	If the current token is a word token, this field contains a string giving the characters of the word token.
static int TT_EOF	A constant indicating that the end of the stream has been read.
static int TT_EOL	A constant indicating that the end of the line has been read.
static int TT_NUMBER	A constant indicating that a number token has been read.

static int TT_WORD	A constant indicating that a word token has been read.
int ttype	After a call to the nextToken method, this field contains the type of the token just read.

Table 11.9

Constructor of StreamTokenizer class is like this-

StreamTokenizer(Reader r)

This is the list of the methods available in StreamTokenizer class-

Method	Purpose
void commentChar(int ch)	Specified that the character argument starts a single-line comment.
void eolIsSignificant(boolean flag)	This method determines whether or not ends of line are treated as tokens.
int lineno()	This method returns the current line number.
void lowerCaseMode(boolean fl)	This method determines whether or not word token are automatically lowercased.
int nextToken()	This method parses the next token from the input stream of this tokenizer.
void ordinaryChar(int ch)	This method specifies that the character argument is "ordinary" in this tokenizer.
void ordinaryChars(int low, int hi)	This method specifies that all characters c in the range low <= c <= high are "ordinary" in this tokenizer.
void parseNumbers()	This method specifies that numbers should be parsed by this tokenizer.

void pushBack()	This method causes the next call to the nextToken method of this tokenizer to return the current value in the ttype field, and not to modify the value in the nval or sval field.
void quoteChar(int ch)	This method specifies that matching pairs of this character delimit string constants in this tokenizer.
void resetSyntax()	This method resets this tokenizer's syntax table so that all characters are "ordinary." See the ordinaryChar method for more information on a character being ordinary.
void slashSlashComments (boolean flag)	This method determines whether or not the tokenizer recognizes C++ style comments.
void slashStarComments (boolean flag)	This method determines whether or not the tokenizer recognizes C style comments.
String toString()	This method returns the string representation of the current stream token and the line number it occurs on.
void whitespaceChars (int low, int hi)	This method specifies that all characters c in the range low <= c <= high are white space characters.
void wordChars (int low, int hi)	This method specifies that all characters c in the range low <= c >= high are word constituents.

Table 11.10

Above is the list of the methods used in StreamTokenizer class.

Example: Unit-11 (Program-09)

/* Write a program to read data from file with object of StreamTokenizer class*/

```
import java.io.*;

public class unit11p09
```

```
{
    public static void main(String args[]) throws InterruptedException,
                                                   FileNotFoundException,
IOException
    {
        FileReader fr= new FileReader("data3.txt");
        BufferedReader br = new BufferedReader(fr);
        StreamTokenizer stoken = new StreamTokenizer(br);
        stoken.commentChar('a');
        int t;
        while ((t = stoken.nextToken()) != StreamTokenizer.TT_EOF)
        {
            switch (t)
            {
              case StreamTokenizer.TT_NUMBER:
                  System.out.println("Number : " + stoken.nval);
                  break;
              case StreamTokenizer.TT_WORD:
                  System.out.println("Word : " + stoken.sval);
                  break;
            }
        }
    }
}
```

Result :

Word : Ankur

Word : Mittal

Number : 2019.0

Word : f

Number : 25.23

Word : X

Check Your Progress

1. What is BufferedStream class in java? Explain with its benefits.
2. What is StreamTokenizer in java? Explain its benefits.
3. What is PrintStream class in java? How it can be used?
4. Write a program in java to store multiple numbers in a file with BurrferedInputStream class.
5. Write a program in java to read and search the number from the file with the help of BufferedOutputStream clas.
6. Write a program in java to count the number of punctuation marks in file wilt StreamTokenizer class.

11.5 SERIALIZATION

Serialization is the process of writing objects to files. In fact when we are trying to write the object to file, first we have to convert the object into binary format (bytes) so that it may be written to file. This conversion of object to bytes and writing to file is called serialization. It writes the state of an object to a byte stream.

And when we try to read object from the file, we have to follow the process of deserialization. Deserialization is opposite process of serialization. Deserialization reads the state of object from file and transfer to the specific object in the program.

Serialization will also be used for implementing RMI (Remote Method Innovation). As RMI is also the java object so it also needs serialization. And there after that object is transferred to other machine.

There is a benefit in serialization. If we have an object for serialization and some objects are dependent on that object. It means if we have a hierarchy of objects and we wish to serialize the top object of the hierarchy in that case, all other objects will automatically be serialized. We need not to serialize the dependent objects.

Serializable interface :

Serializable is the interface which needs to be implemented in the class which is to be serialized. This interface defines no member. It is only used to indicate the JVM that the object of this class may be written to destination and at the time of writing to destination, JVM will automatically serialize the object. If we have declared any variable with transient modifier, in that case value of that variable will not be serialized. Static variables will also not be serialized as they are not in regard of the object, they are in regard of the class.

For example-

```
class student implements Serializable
{
   :
}
```

In the above example object of the student class may directly be written onto file as in the definition of the student class we have implements Serialzible interface.

For writing object to file we have to use the Object of ObjectOutputStream. This object has to be connected with object of FileOutputStreamClass class.

```
FileOutputStream fos = new FileOutputStream("filename");
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(emp);
```

Once the object of ObjectOutputStream is created, it may directly be written with writeObject() method.

Example: Unit-11 (Program-10)

/* Write a program to write the object in file. */

```
import java.io.*;
import java.util.*;
class student implements Serializable
{
    int rno;
    String name;
    public void input()
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("\n Enter Name of the student ");
        name=sc.nextLine();
```

```
            System.out.print("\n Enter Roll number of the student ");
            rno=sc.nextInt();
        }
        public void display()
        {
            System.out.println("\n Name of the student "+name);
            System.out.println("\n Roll number of the student "+rno);
        }
    }
    public class unit11p10
    {
        public static void main(String args[]) throws IOException
        {
            FileOutputStream fos= new FileOutputStream("data4.txt");
            ObjectOutputStream oos=new ObjectOutputStream(fos);
            student s=new student();
            s.input();
            oos.writeObject(s);
            System.out.println("Record of student is stored");
            oos.close();
            fos.close();
        }
    }
```

Result :

```
Enter Name of the student Amit
Enter Roll number of the student 101
Record of student is stored
```

Above program will store the object in the file.

Example: Unit-11 (Program-11)

/* Write a program to read the object from file. */

```
import java.io.*;
import java.util.*;
```

```
class student implements Serializable
{
    int rno;
    String name;
    public void input()
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("\n Enter Name of the student ");
        name=sc.nextLine();
        System.out.print("\n Enter Roll number of the student ");
        rno=sc.nextInt();
    }
    public void display()
    {
        System.out.println("\n Name of the student "+name);
        System.out.println("\n Roll number of the student "+rno);
    }
}
public class unit11p11
{
    public static void main(String args[]) throws IOException,
ClassNotFoundException
    {
        FileInputStream fis= new FileInputStream("data4.txt");
        ObjectInputStream ois=new ObjectInputStream(fis);
        student s=new student();
        s=(student)ois.readObject();
        ois.close();
        fis.close();
        s.display();
    }
```

```
}
```

Result :

Name of the student Amit

Roll number of the student 101

Above program will read the object from the object.

Check Your Progress

1. What is serialization? How it is achieved?
2. Write a program in java to store 10 employee records in the file.
3. Write a program in java to read the employee records from file.
4. Write a program to search the record of the employee on the basis of employee name from the file. If found, display the record.

11.6 RANDOM ACCESS FILE

RandomAccessFile class is used for reading and writing to random access file. This system works with file in the form of a large array of bytes stored in the file system. There is a cursor too with the help of which we can move the file pointer position. It works like an array of byte stored in the File. It is not derived from InputStream or OutputStream. Instead, RandamAccessFile implements the interfaces DataInput and DataOutput

Constructor for this class is-

```
RandomAccessFile("fileName", "Mode")
```

To open the file for read only mode we have to use "r" and for read-write operations we have to use "rw".

This is the list of methods of this class-

Method	Purpose
void close()	It closes random access file stream and releases system resources (if any)
FileChannel getChannel()	It returns the unique FileChannel object associated with this file.
int readInt()	It reads a integer from this file.
String readUTF()	It reads in a string from this file.

void seek(long pos)	It sets the file-pointer offset, measured from the beginning of this file
void writeDouble (double v)	It converts the double argument to a long and then writes that long value to the file
void writeFloat(float v)	It converts the float argument to an int and then writes that int value to the file
void write(int b)	It writes the specified byte to this file.
int read()	It reads a byte of data from this file.
long length()	It returns the length of this file.

Table 11.11

Example: Unit-11 (Program-12)

/* Write a program to store the data in the file with RandamAccessFile. */

```
import java.io.*;
import java.util.*;
public class unit11p12
{
    public static void main(String args[]) throws IOException, ClassNotFoundException
    {
        String str;
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter the string to store in the file: ");
        str=sc.nextLine();
        RandomAccessFile raf = new RandomAccessFile("data5.txt", "rw");
        raf.seek(0);
        raf.write(str.getBytes());
        raf.close();
```

```
            System.out.println("Data stored in the file successfully");
        }
    }
```

Result :

Enter the string to store in the file: I am the student of computer science

Data stored in the file successfully

Above program will store the data in the file.

Example: Unit-11 (Program-13)

/* Write a program to read the data from file with RandamAccessFile. */

```
import java.io.*;
import java.util.*;
public class unit11p13
{
    public static void main(String args[]) throws IOException, ClassNotFoundException
    {
        RandomAccessFile raf = new RandomAccessFile("data5.txt", "r");
        int len=(int)raf.length();
        byte[] bt=new byte[len];
        raf.seek(0);
        raf.read(bt);
        raf.close();
        System.out.println("Data from file: "+new String(bt));
    }
}
```

Result :

Data from file: I am the student of computer science

In the above example, object of the RandamAccessFilw will read the data from file and it will be displayed in the form of string.

Check Your Progress

1. What is RandamAccessFile in java?
2. What are the modes in RamdamAccessFile?
3. Write a program in java to store the data in the file using RandamAccessFile.
4. Write a program in java to search and element from file in java and display the previous element to searched element using RamdamAccessFile.

11.7 SUMMARY

Java has several IO classes in the java.io package. Java IO is based on two formats byte stream and character stream. Byte stream process 8 bits at a time while character stream process 16 bits at a time. Java provides several classes in both categories. Java also supports RamdamAccessFile for these operation. In java using the serializability we may write or read the entire object at a time. Java also provides the StreamTokenizer for parsing data in the form of tokens. Java also provides a helpful of library for managing files and directories in in the operating system. With java we may also set the properties of the file.

11.8 EXERCISE

Q. 1. What is File class? Describe the constructors used in File class.

Q. 2. Write a program in java to list the read only files of a directory.

Q. 3. Write a program in java to change the name of a file to a new name.

Q. 4. Write a program in java to list all html files inside the directory and count them.

Q. 5. Write a program in java to set all read only files of any directory to readable files.

Q. 6. Write a program in java to list all files those are create in the current year.

Q. 7. Write a program in java to store the 10 student's record in file.

Q. 8. Write a program in java to search the student record on the basis of his name.

Q. 9. Write a program in java to delete the record of a student from the file. It has to be searched on the basis of his roll number.

Q. 10. Write a program in java to count the number of words and numbers in file.

Q. 11. Write a program in java to count the number of lines and white spaces in the program with the help of StreamTokenizer class.

Uttar Pradesh Rajarshi Tandon
Open University

Master in Computer Applications

MCS-112

Java Programming

BLOCK

4

GRAPHICS AND USER INTERFACES

UNIT-12

Applets

UNIT-13

Graphics and user interfaces

UNIT-14

Networking Features

Course Design Committee

Prof. Ashutosh Gupta — **Chairman**
Director (In-charge)
School of Computer and Information Science, UPRTOU Allahabad

Prof. Suneeta Agarwal — **Member**
Department of CSE
MNNIT Allahabad, Prayagraj

Dr. Upendra Nath Tripathi — **Member**
Associate Professor, Department of Computer Science
Deen Dayal Upadhyaya Gorakhpur University, Gorakhpur

Dr. Ashish Khare — **Member**
Associate Professor, Department of Computer Science
University of Allahabad, Prayagraj

Dr. Marisha — **Member**
Assistant Professor (Computer Science),
School of Science, UPRTOU Allahabad

Mr. Manoj Kumar Balwant — **Member**
Assistant Professor (computer science),
School of Sciences, UPRTOU Allahabad

Course Preparation Committee

Er. Pooja Yadav — **Author (Unit 1 to 14)**
Asst Professor, Dept. of Computer Science & Information Technology, MJPRU, Bareilly

Prof. Abhay Saxena — **Editor**
Professor, DSVV, Shantikunj, Haridwar (UK)-249411

Prof. Ashutosh Gupta — **Director (In-charge)**
School of Computer and Information Science, UPRTOU Allahabad

Dr. Marisha, — **Coordinator**
Asstt. Professor (Computer Science), School of Science, UPRTOU, Prayagraj

Mr. Manoj Kumar Balwant — **Coordinator**
Asstt. Professor (Computer Science), School of Science, UPRTOU, Prayagraj

ISBN :

Printed and Published by Dr. Arun Kumar Gupta Registrar, Uttar Pradesh Rajarshi Tandon Open University, 2020.
Printed By : Chandrakala Universal Pvt. 42/7 Jawahar Lal Neharu Road, Prayagraj.

UNIT-12 APPLETS

Structure :

12.1 OBJECTS

Applet as a remote accessing code will be covered in this unit. This will explain the life cycle of the applet and its architecture. How java managed the events, will be covered in this unit. AWT in applets will be discussed. This will also compare various technologies of java. Events management with applet will be implemented.

12.2 INTRODUCTION

Java is one of the most widely used technology these days. Reason behind is that it supports all 5 types of application development. Those are-

1. Console application : These application have common entry and exit point. Programs made in java with main methods are the example of this type.
2. Window application : These applications are graphical application without fixed entry and exit point. Any graphical application software is the example of this type.
3. Web application : These are web based applications which don't need installation on client side. All web applications like google, gmail, yahoo etc are the example of this type.

4. Mobile application : These applications are those which are meant for smart devices. Mobile apps or tablet apps are the example of this type.

5. Web services : These are not the application but these are services and used when two different organizations wants to access some functionality of each other. Payment gateways, SMS gateways are the example of this type.

We have gone through the console application till now. In case of java, we may also design graphical applications. Applet is one of the graphical application designed in java.

12.3 THE APPLET CLASS

Java provides three types of codes in programming-

1. Application code : This code in installed on the client machine and executes there. All window applications are the example of this type.

2. Applet code : Applet code is stored on the remote machine. Whenever user request/call for the code, it is downloaded on the client machine and executes there. But it is never installed on the client machine. These codes were broadly used for thick client application.

3. Servlet code : Servlet code is the server side code. It is stored on the web servers. When user request for this code, it gets executed on the server and in the response, result is returned to the client. In this type code is never transferred to the client machine. Client machine will get the result only in response.

Java applet is a special code that is embedded in the HTML code and it generates the dynamic contents. It is initially stored on the web server, at the request of user, it is downloaded onto the client machine via browser and it executes there till browser is in the memory. As soon as browser is closed, it is unloaded from the memory. It is not permanently stored in the client machine. It is also not executed on the server machine.

Applet vs. Application of java	
APPLICATION CODE	**APPLET CODE**
This contains main() method	This does not contain main() method
It requires JRE for execution	It requires a browser and JRE

It may be executed from command prompt.	It needs a browser for execution
It can easily access the resources of the system	It rarely access the resources except browser
It does not need security	Requires highest security for the system as they are untrusted
It may be graphical or non-graphical	It is graphical only
It is coded to perform some task of the user.	It is coded to perform any small task of part of task.
It is installed on client machine.	It can't be installed on machine.

Table 12.1

Benefits of applet :

- They are very secure.
- It works at client side so less response time.
- Applets can be executed by browsers running under different platforms.

One disadvantage of Applets is that plugins are required at the client browser for executing applets.

Applet is implemented in the form of class in java. Here are two packages which are used in Applet implementation in java-

- java.applet
- java.awt

java.applet package contains the Applet class. Applet class is to be inherited in the class, which we want to use as an applet. The class for applet must be a public class.

Here is the syntax for applet class-

```
public class MyApplet extends Applet
{
      :
}
```

Once we have inherited the Applet class in any public class, then this new class will not be an application code. It will be an applet code. It indicates the compiler that this class will be used for the applet execution. This will not be executed from

command prompt, this will be called from html code. Although this code will be compiled but not executed. JDK gives us a flexibility to test the applet. For this JDK give a utility known as Appletviewer. Appletviewer is a used to test the applet temporarily. It will show how the applet will look like and we may interact with the applet as well. But keep in mind that this appletviewer is not meant for applet execution.

Now moving towards coding part of the applet, we have to override the methods in the applet class. paint() is the method which draws the objects which are to be displayed on applet.

Here is the sample program of applet in java-

Example: Unit-12 (Program-01)

/* Write a program display the message in applet. */

```
import java.awt.*;
import java.applet.*;
/*<applet code="unit12p01" width=200 height=200></applet> */
public class unit12p01 extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("Ankur Mittal", 20,40);
    }
}
```

Result :

As we have to run the applet program. There are two steps in running the program.

Step1: Compile the java file with javac as we do earlier.

Setp2: For testing java applet, add the applet tag with source code in source code like:

```
/* <applet code="appletClassName" height=200 width=200></applet> */
```

This tag may be added anywhere in code inside the comment.

Step3: Test the applet with appletviewer like:

```
C:\> appletviewer filename.java
```

Methods used in Graphics class in Applet class:

Methods used in Graphics class	
Method name	**Purpose**
public abstract void drawstring (String str, int x, int y)	is used to draw the specified string.
public void drawRect (int x, int y, int width, int height)	draws a rectangle with the specified width and height.
public abstract void fillRect (int x, int y, int width, int height)	is used to fill rectangle with the default color and specified width and height.
public abstract void drawOval (int x, int y, int width, int height)	is used to draw oval with the specified width and height.
public abstract void fillOval (int x, int y, int width, int height)	is used to fill oval with the default color and specified width and height.
public abstract void drawLine (int x1, int y1, int x2, int y2)	is used to draw line between the points(x1, y1) and (x2, y2).
public abstract boolean drawImage (Image img, int x, int y, ImageObserver observer)	is used draw the specified image.
public abstract void drawArc (int x, int y, int width, int height, int startAngle, int arcAngle)	is used draw a circular or elliptical arc.
public abstract void fillArc (int x, int y, int width, int height, int startAngle, int arcAngle)	is used to fill a circular or elliptical arc.
public abstract void setColor(Color c)	is used to set the graphics current color

	to the specified color.
public abstract void setFont(Font font)	is used to set the graphics current font to the specified font.

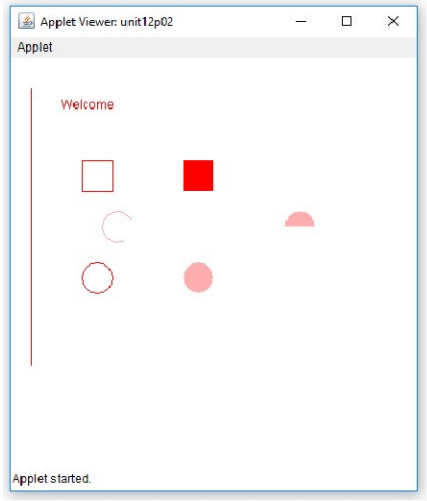
Table 12.2

Example: Unit-12 (Program-02)

/* Write a program display the various shapes in applet. */

```
import java.awt.*;
import java.applet.*;
/*<applet code="unit12p02" width=400 height=400></applet> */
public class unit12p02 extends Applet
{
    public void paint(Graphics g)
    {
        g.setColor(Color.red);
        g.drawString("Welcome",50, 50);
        g.drawLine(20,30,20,300);
        g.drawRect(70,100,30,30);
        g.fillRect(170,100,30,30);
        g.drawOval(70,200,30,30);
        g.setColor(Color.pink);
        g.fillOval(170,200,30,30);
        g.drawArc(90,150,30,30,30,270);
        g.fillArc(270,150,30,30,0,180);
    }
}
```

Result :



Above program will show various shapes on the applet area.

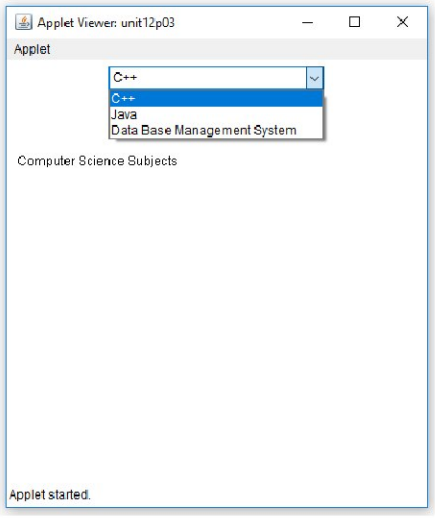
Example: Unit-12 (Program-03)

/* Write a program display the combo box in applet. */

```
import java.awt.*;
import java.applet.*;
/*<applet code="unit12p03" width=400 height=400></applet> */
public class unit12p03 extends Applet
{
 Choice sub;
 public void init()
 {
   sub=new Choice();
   sub.add("C++");
   sub.add("Java");
   sub.add("Data Base Management System");
   add(sub);
 }
 public void paint(Graphics g)
 {
   g.drawString("Computer Science Subjects", 10, 100);
 }
```

```
}
```

Result:



Above program will add three items in the combo box but it will not do anything after selection as we have just designed them and we have not assigned any coding for it.

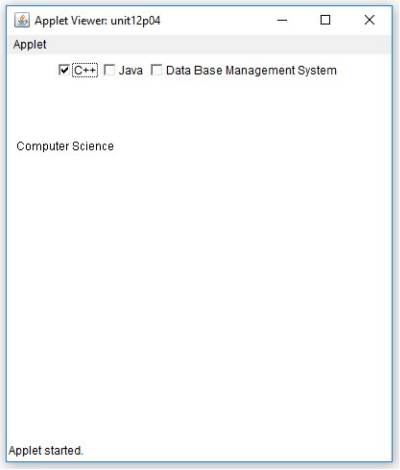
Example : Unit-12 (Program-04)

/* Write a program display the items with check box in applet. */

```
import java.awt.*;
import java.applet.*;
/*<applet code="unit12p04" width=400 height=400></applet> */
public class unit12p04 extends Applet
{
 Checkbox cpp,java,dbms;
 public void init()
 {
  cpp=new Checkbox("C++");
  java=new Checkbox("Java");
  dbms=new Checkbox("Data Base Management System");
  add(cpp);
  add(java);
```

```
    add(dbms);
  }
  public void paint(Graphics g)
  {
    g.drawString("Computer Science", 10, 100);
  }
}
```

Result:



Above program will display the 3 items in the check box. Check box may be selected but it will not take any action as there is no coding associated with it.

Check Your Progress

1. What is applet? Explain its benefits
2. Compare applet and standalone application.
3. Design an applet in java to display some metropolitian cities name in the combo box.
4. Design an applet to show the subjects of the current semester with checkboxes
5. Design an applet to show the smiley face.

12.4 APPLET ARCHITECTURE

This applet begins with two import statements. The first import statement is for the Abstract Window Toolkit (AWT) classes. Applets interact with the user through the AWT and not through the console-based I/O classes. An applet must be a subclass of the java.applet.Applet class. The Applet class provides the

standard interface between the applet and the browser environment. Swing provides a special subclass of the Applet class called javax.swing.JApplet. The JApplet class should be used for all applets that use Swing components to construct their graphical user interfaces (GUIs). The browser's Java Plug-in software manages the lifecycle of an Applet. The applet in Java can appear in a frame of the web page, a new application window, Sun's AppletViewer, or a stand-alone tool for testing them.

Applet is different from console application of java. So the architecture of applet is different from console application. As java applet is a graphical application, which runs inside the web browser in the web page. Applet class and are enabled by reference with HTML page. You can observed that when applet are combined with HTML, that can make an interface more dynamic and powerful than with HTML alone. While some Applet do not more than scroll text or play movements, but by incorporating theses basic features in webpage you can make them dynamic. These dynamic web page can be used in an enterprise application to view or manipulate data coming from some source on the server. The Applet and there class files are distribute through standard HTTP request and therefore can be sent across firewall with the web page data. Applet code is referenced automatically each time the user revisit the hosting website. Therefore keeps full application up to date on each client desktop on which it is running.

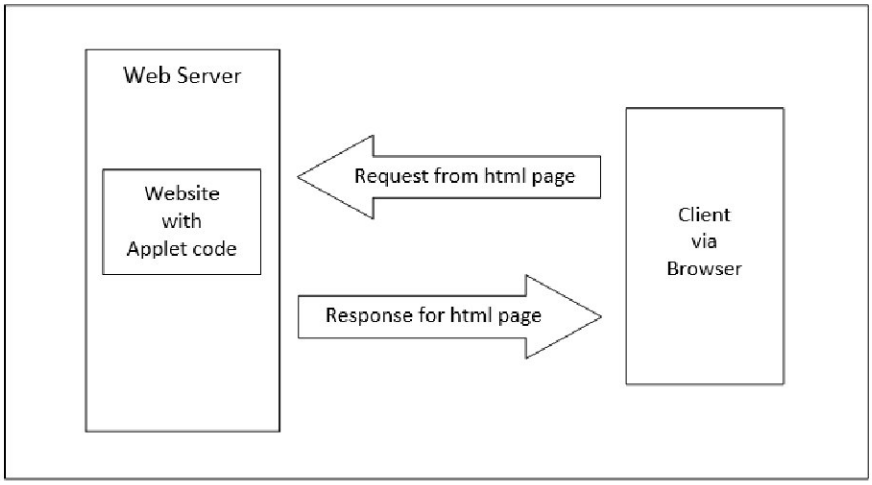


Figure 12.1

In the above diagram, it is shown that client request for the html page to web server. At the web server, applet code is stored embedded in the html code. Applet code is stored on server in the form of class file. After getting the request from client, server return the html code along with applet code. Applet code is returned into memory at client and executes there. Applet is not installed in client machine. Applet is just executed there. Applet code can't access system recourses other than browser.

If we are using remote applet code, we may also use database on the web server may be accessed there.

Here are the 5 methods which represent 5 states of the applet-

1. **init**() method is called at the time of starting the execution. This is called only once in the life cycle.
2. **start**() method is called by the init() method. This method is called a number of times in the life cycle; whenever the applet is deiconifed , to make the applet active.
3. **paint**() method is called by the start() method. This is called number of times in the execution.
4. **stop**() method is called whenever the applet window is iconified to inactivate the applet. This method is called number of times in the execution.
5. **destroy**() method is called when the applet is closed. This method is called only once in the life cycle.

Check Your Progress

1. Explain the architecture of applet.
2. Explain the request and response model in regard of applet.
3. Compare Applet and JApplet.

12.5 AN APPLET SKELETON : INITIALIZATION AND TERMINATION

Applet gone through the various states from the loading time into memory to removal from memory. These states are called the lifecycle of the applet. Each state has its own method. These methods are called call back methods as they are automatically called by the browser for the smooth execution of the applet in browser. We just have to write the code of these methods but we need not to write the code for calling the methods.

There are 5 states so there are 5 methods in applet lifecycle-

1. **init**(): The applet's journey starts from here. In this method, browser creates the the applet object. This method can be used to initialize variables, setting background and foreground colors in GUI etc. as this method is called before all the other methods. It acts like a constructor in an application. It is also equivalent to born state of a thread.
2. **start**(): init() method initialize the applet object but applet is not active. It is in the inactive state. start() method make the applet active because inactive applet can't access microprocessor even if microprocessor is free. To make the applet active, the init() method calls start() method. After calling of start() method, applet becomes active and thereby eligible for accessing processor time.
3. **paint**(): java.awt.Graphics is accepted as a parameter in this method. This Graphics class has the capacity of drawing which is necessary to draw on

applet window. This is the place where the programmer can write his code of what he expects from applet like animation etc. This pain() method is equivalent to runnable state of thread.

4. **stop()**: To make the applet temporarily inactive, we may use this method. An applet may be made active and inactive any number of times. Making applet inactive, doesn't unload applet from the memory. Stopping applet just made applet inactive, later on with start() method, applet may be again made active. It is equivalent to the blocked state of the thread.

5. **destroy()** : Just before the garbage collection for applet, this method is called. This methods shows the end of the applet. It is also the best place for cleanup code. destroy() method is equivalent to the dead state of the thread.

Here is the diagram of the Applet lifecycle-

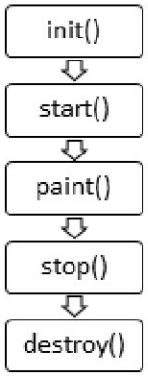


Figure 12.2

From the above diagram it is clear that, the init() and destroy() methods are called only once in the life cycle. But start(), paint() and stop() methods are called a number of times.

Check Your Progress

1. Explain the lifecycle of applet.
2. Compare init() and start method of applet.
3. How init() in applet and constructor in java class are similar? Explain.
4. Describe the similarity between paint() method of applet and main() of standalone application.

12.6 HANDLING EVENTS

Event is the any action taken. In computer events may be triggered by these three sources-

- Generated by user

- Generated by operating System
- Generated by other event

User takes the action as and when required. Operating system takes action as configured to do some tasks. But sometimes any event may cause other event to occur.

In java event is same as in our operating system. But as a programmer event is the change in the state of any object.

Event Sources :

A source is an object that generates an event. This occurs when the internal state of that object changes in some way. Sources may generate more than one type of event. A source must register listeners in order for the listeners to receive notifications about a specific type of event. Each type of event has its own registration method.

Event Handling :

There are two Event Handling Mechanisms in java-

- By The Delegation Event Model.
- By extending AWT component.

The Delegation Event Model :

This is the standard modern approach which defines standard and consistent mechanisms to generate and process events. Its concept is quite simple: a source generates an event and sends it to one or more listeners. In this scheme, the listener simply waits until it receives an event. Once received, the listener processes the event and then returns.

The advantage of this design is that the application logic that processes events is cleanly separated from the user interface logic that generates those events. A user interface element is able to "delegate" the processing of an event to a separate piece of code.

In the delegation event model, listeners must register with a source in order to receive an event notification. This provides an important benefit: notifications are sent only to listeners that want to receive them.

Event Sources :

A source is an object that generates an event. This occurs when the internal state of that object changes in some way. Sources may generate more than one type of event. A source must register listeners in order for the listeners to receive notifications about a specific type of event. Each type of event has its own registration method.

Event Listeners :

A listener is an object that is notified when an event occurs. It has two major requirements. First, it must have been registered with one or more sources to

receive notifications about specific types of events. Second, it must implement methods to receive and process these notifications. The methods that receive and process events are defined in a set of interfaces found in java.awt.event.

Event Classes :

Event classes represent events. They are at the core of Java's event handling mechanism. At the root of the Java event class hierarchy is EventObject, which is in java.util. It is the superclass for all events

Here is the list of the classes of the Events-

List of Event Classes	
Event Class	**Purpose**
ActionEvent	Generated when a button is pressed, a list item is double-clicked, or a menu item is selected.
AdjustmentEvent	Generated when a scroll bar is manipulated.
ComponentEvent	Generated when a component is hidden, moved, resized, or becomes visible.
ContainerEvent	Generated when a component is added to or removed from a container.
FocusEvent	Generated when a component gains or loses keyboard focus.
InputEvent	Abstract super class for all component input event classes.
ItemEvent	Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected.
KeyEvent	Generated when input is received from the keyboard.
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.
MouseWheelEvent	Generated when the mouse wheel is moved.
TextEvent	Generated when the value of a text area or text field is changed.
WindowEvent	Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Table 12.3

Sources of Events :

When an event occurs, the event source invokes the appropriate method defined by the listener and provides an event object as its argument.

Here is the list of the object, they may be the event sources-

Objects as Event Source	
Event Source	**Description**
Button	Generates action events when the button is pressed.
Checkbox	Generates item events when the check box is selected or deselected.
Choice	Generates item events when the choice is changed.
List	Generates action events when an item is double-clicked; generates item events when an item is selected or deselected.
Menu Item	Generates action events when a menu item is selected; generates item events when a checkable menu item is selected or deselected.
Scrollbar	Generates adjustment events when the scroll bar is manipulated.
Text components	Generates text events when the user enters a character.
Window	Generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Table 12.4

Event Listener Interfaces :

The delegation event model has two parts: sources and listeners. Listeners are created by implementing one or more of the interfaces defined by the java.awt.event package. When an event occurs, the event source invokes the appropriate method defined by the listener and provides an event object as its argument.

Here is the list of the commonly used Event Listener interfaces of java-

Commonly Used Event Listener Interfaces

Interface	Description
ActionListener	Defines one method to receive action events.
AdjustmentListener	Defines one method to receive adjustment events.
ComponentListener	Defines four methods to recognize when a component is hidden, moved, resized, or shown.
ContainerListener	Defines two methods to recognize when a component is added to or removed from a container.
FocusListener	Defines two methods to recognize when a component gains or losses keyboard focus.
ItemListener	Defines one method to recognize when the state of an item changes.
KeyListener	Defines three methods to recognize when a key is pressed, released, or typed.
MouseListener	Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released.
MouseMotionListener	Defines two methods to recognize when the mouse is dragged or moved.
MouseWheelListener	Defines one method to recognize when the mouse wheel is moved.
TextListener	Defines one method to recognize when a text value changes.
WindowFocusListener	Defines two methods to recognize when a window gains or loses input focus.
WindowListener	Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Table 12.5

Using the Delegation Event Model :

Applet programming using the delegation event model is actually quite easy. Just follow these two steps:

1. **Implement the appropriate interface in the listener** so that it will receive the type of event desired.
2. **Implement code to register and unregister (if necessary) the listener** as a recipient for the event notifications.

Remember that a source may generate several types of events. Each event must be registered separately. Also, an object may register to receive several types of events, but it must implement all of the interfaces that are required to receive these events.

Adapter Classes :

Java provides a special feature, called an *adapter class* that can simplify the creation of event handlers in certain situations. An adapter class provides an empty implementation of all methods in an event listener interface. Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface. You can define a new class to act as an event listener by extending one of the adapter classes and implementing only those events in which you are interested.

Here is the list of Commonly Used Listener Interfaces Implemented by Adapter Classes-

Commonly Used Listener Interfaces Implemented by Adapter Classes	
Adapter Class	**Listener Interface**
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener

Table 12.6

12.6.1 HTML APPLET TAG

An applet may be invoked by embedding directives in an HTML file. Actually we use the applet in the html code. So appletviewer is just for testing. For running the applet in the browser use this html code like:

```
<html>
    <head>
        <title> Sample HTML with applet</title>
    </head>
    <body>
        <applet code="appletClassName" height=200 width=200></applet>
    </body>
</html>
```

In the above example appletClassName is the name of the class file that is generated after compiling the applet source code by java compiler. Height and width is given in pixels here. It may vary as per the need of the program. Location of applet in web page may vary as per the requirement of the web page.

Browser Responsibility for Applet

Applet life cycle methods are callback methods because they are called implicitly by the browser for the smooth execution of the applet. The browser should provide an environment known as a container for the execution of the applet. Following are the responsibilities of the browser.

- For the smooth execution, it should call the callback methods at appropriate times.
- It is responsible to maintain the Applet Life Cycle.
- It should have the capability to communicate between applets, applet to JavaScript and HTML, applet to browser, etc.

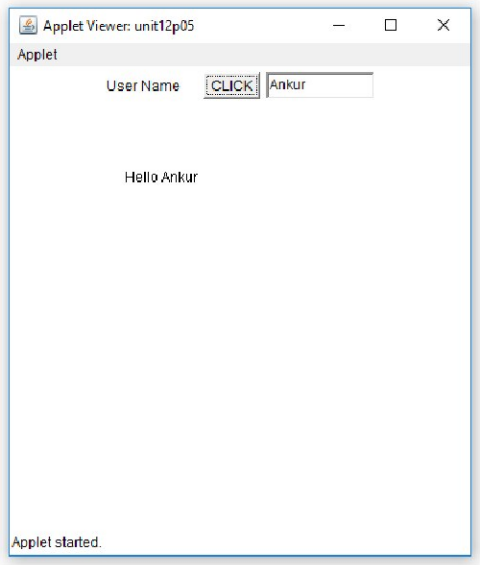
Example: Unit-12 (Program-05)

/* Write a program to display the user name on button click in applet. */

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*<applet code="unit12p05" width=400 height=400></applet> */
public class unit12p05 extends Applet implements ActionListener
{
 Label name;
 Button click;
```

```
TextField uname;
String msg;
public void init()
{
 name =new Label("User Name ");
 click= new Button("CLICK");
 uname = new TextField(10);
 add(name);
 add(click);
 add(uname);
 click.addActionListener(this);
}
public void actionPerformed(ActionEvent ae)
{
  String str = ae.getActionCommand();
  if(str.equals("CLICK"))
    msg = "Hello "+uname.getText();
  else
    msg = "unknown event";
  repaint();
}
public void paint(Graphics g)
{
 g.drawString(msg, 100, 100);
}
}
```

Result:



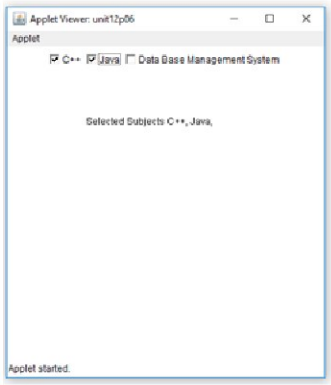
Example: Unit-12 (Program-06)

/* Write a program to add subject names in check box and display all selected subject names in applet. */

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*<applet code="unit12p06" width=400 height=400></applet> */
public class unit12p06 extends Applet implements ItemListener
{
 Checkbox cpp,java,dbms;
 String msg;
 public void init()
 {
  cpp=new Checkbox("C++");
  java=new Checkbox("Java");
  dbms=new Checkbox("Data Base Management System");
  add(cpp);
  add(java);
  add(dbms);
  cpp.addItemListener(this);
  java.addItemListener(this);
```

```
    dbms.addItemListener(this);
  }
  public void itemStateChanged(ItemEvent ie)
  {
     msg = "Selected Subjects";
     if(cpp.getState()==true)
       msg=msg+ " C++,";
     if(java.getState()==true)
       msg=msg+ " Java,";
     if(dbms.getState()==true)
       msg=msg+ " DataBase management System";
     repaint();
  }
  public void paint(Graphics g)
  {
    g.drawString(msg, 100, 100);
  }
}
```

Result:



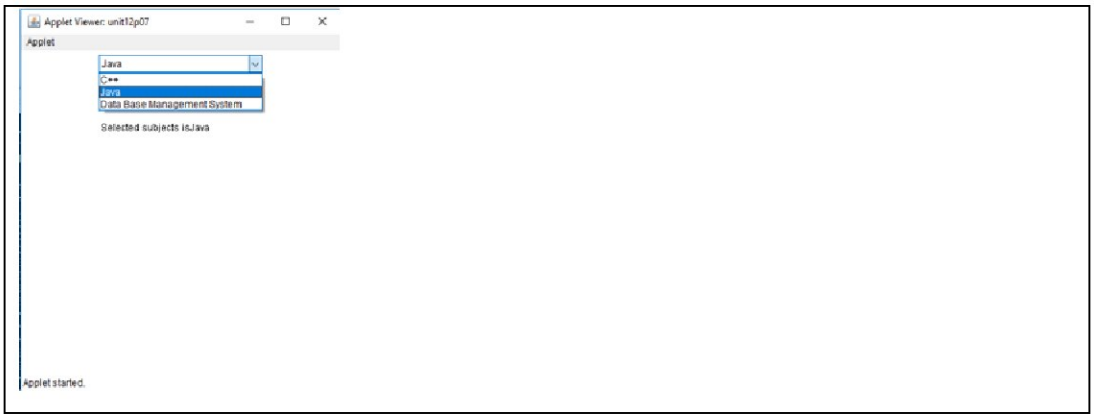
Example: Unit-12 (Program-07)

/* Write a program to add subjects in combo box and display selected subject in applet. */

```
import java.awt.*;
import java.awt.event.*;
```

```
import java.applet.*;
/*<applet code="unit12p07" width=400 height=400></applet> */
public class unit12p07 extends Applet implements ItemListener
{
 Choice sub;
 String msg;
 public void init()
 {
  sub=new Choice();
  sub.add("C++");
  sub.add("Java");
  sub.add("Data Base Management System");
  add(sub);
  sub.addItemListener(this);
 }
 public void itemStateChanged(ItemEvent ie)
 {
  msg="Selected subjects is"+sub.getSelectedItem();
  repaint();
 }
 public void paint(Graphics g)
 {
  g.drawString(msg, 100, 100);
 }
}
```

Result :



Example: Unit-12 (Program-08)

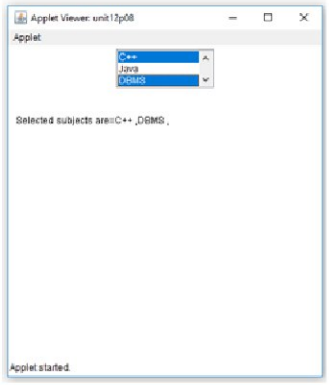
/* Write a program to add subjects in list box and display selected subjects in applet. */

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

/*<applet code="unit12p08" width=400 height=400></applet> */
public class unit12p08 extends Applet implements ItemListener
{
 List sub;
 String msg;
 public void init()
 {
   sub=new List(3,true);
   sub.add("C++");
   sub.add("Java");
   sub.add("DBMS");
   sub.add("SQL");
   add(sub);
   sub.addItemListener(this);
 }
```

```
public void itemStateChanged(ItemEvent ie)
{
  String[] str=sub.getSelectedItems();
  msg="Selected subjects are=";
  for(int i=0;i<str.length;i++)
    msg+=str[i]+" ,";
  repaint();
}
public void paint(Graphics g)
{
  g.drawString(msg, 10, 100);
}
}
```

Result:



Example: Unit-12 (Program-09)

/* Write a program to change the colour of rectangle with scrollbars in applet. */

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*<applet code="unit12p09" width=400 height=400></applet> */
```

```
public class unit12p09 extends Applet implements AdjustmentListener
{
 Scrollbar c1,c2,c3;
 public void init()
 {
  c1=new Scrollbar(Scrollbar.HORIZONTAL,0,1,0,255);
  c2=new Scrollbar(Scrollbar.HORIZONTAL,0,1,0,255);
  c3=new Scrollbar(Scrollbar.HORIZONTAL,0,1,0,255);
        //style,initialValue,thumbSize,min,max
  add(c1);
  add(c2);
  add(c3);
  c1.addAdjustmentListener(this);
  c2.addAdjustmentListener(this);
  c3.addAdjustmentListener(this);
 }
 public void adjustmentValueChanged(AdjustmentEvent ae)
 {
  repaint();
 }
 public void paint(Graphics g)
 {
  g.setColor(new Color(c1.getValue(),c2.getValue(),c3.getValue()));
  g.fillRect(10,10,50,60);
 }
}
```

Result:

Check Your Progress

1. What is event delegation model?
2. Compare adapter classes with interfaces in event handling.
3. What is action listener? Explain any 2 listeners.
4. Design an applet in java to calculate the sum of two numbers.

12.7 SUMMARY

Applet was one of the widely used code in java for remote accessing. But now a days due to the advancements in java script, applet codes are rarely used.

Applet is the code that is embedded with the html code and remotely accessed on HTTP Request. It's execution is done at client end but it is not installed at client end. It resides at client temporarily. Even it can't access the system resources completely. They run inside the browser with java runtime plugin.

There are 5 states of applet during its execution, and they are known as the methods of applet lifecycle. They are- init(), start(), paint(), stop() and destroy(). init() and destroy() methods are like constructor and destructor of java standalone programs respectively. Paint() methid is like main() method of standalone application. Once loaded in memory, we may start and stop the applet any number of time. There is event delegation in applets for event handling.

12.8 EXERCISE

Q.1 Explain the applet lifecycle and its architecture.

Q.2 What is event delegation model?

Q.3 Design an applet in java to find the greatest of three numbers.

Q.4 Design an applet in java to check for the prime number.

Q.5 Design an applet to display the hut shape via graphics in applet.

Q.6 Design an applet to draw the rectangle of specific size as specified by user at run time in textboxes.

Q.7 Design an applet to perform the arithmetic operation (+,-,*,/) on two number given by the user on the basis of operation selected by user from combo box.

Q.8 Design an applet to find the reverse of the sting as specified by user on button click.

UNIT-13 GRAPHICS AND USER INTERFACES

Structure :

13.1 OBJECTIVE

Java programming language is widely used for graphical application development as well. Graphical applications are the need of the hour for end users as console application are not meant for end users. So java provides a lots of support for this. Graphical applications are very easy to learn by end users as graphics are easier to understand and remember. Besides this, graphical application give a sense of easiness to end user for their functionality. It gives end user an easy zone to work with.

13.2 INTRODUCTION

Java provides the Applet for graphical working. Besides that java also provides the individual application development in the form of AWT (abstract window toolkit). There is also one more addition other than these two, and that is swing development. So java provides three facility in the form of applet, awt and swing for graphical application development.

13.3 GRAPHICS CONTESTS AND GRAPHICS OBJECTS

In java, Graphics class the given. Graphics class is the abstract super class for all graphics context. It allow an application to draw onto components that can be realized on various devices, or onto off-screen images as well.

All state information that is required basic rendering operation for java, is encapsulated in Graphics class. Some of the State information that is contained by graphics class is-

- The Component object on which to draw.
- The current font.
- The current clip.
- The current colour
- The current logical pixel operation function.
- A translation origin for rendering and clipping coordinates.
- The current XOR alternation colour

There are several methods which are available in Graphics class-

Method of Graphic Class

Method name	Purpose
abstract void drawLine (int x1, int y1, int x2, int y2)	Draws a line, using the current colour, between the points (x1, y1) and (x2, y2)
abstract void drawOval (int x, int y, int width, int height)	Draws the outline of an oval.
abstract void drawPolygon (int[] xPoints, int[] yPoints, int nPoints)	Draws a closed polygon defined by arrays of x and y coordinates.
void drawPolygon (Polygon p)	Draws the outline of a polygon defined by the specified Polygon object.
abstract void drawPolyline (int[] xPoints, int[] yPoints, int nPoints)	Draws a sequence of connected lines defined by arrays of x and y coordinates.
void drawRect (int x, int y, int width, int height)	Draws the outline of the specified rectangle.
abstract void drawRoundRec t(int x, int y, int width, int height, int arcWidth, int arcHeight)	Draws an outlined round-cornered rectangle using this graphics context's current colour.
abstract void drawstring (String str, int x, int y)	Draws the text given by the specified string, using this graphics context's current font and colour.
void draw3DRect (int x, int y, int width, int height, boolean raised)	Draws a 3-D highlighted outline of the specified rectangle.

abstract void drawArc (int x, int y, int width, int height, int startAngle, int arcAngle)	Draws the outline of a circular or elliptical arc covering the specified rectangle.
void drawBytes (byte[] data, int offset, int length, int x, int y)	Draws the text given by the specified byte array, using this graphics context's current font and colour.
void drawChars (char[] data, int offset, int length, int x, int y)	Draws the text given by the specified character array, using this graphics context's current font and colour.
abstract boolean drawImage(Image img, int x, int y, int width, int height, ImageObserver observer)	Draws as much of the specified image as has already been scaled to fit inside the specified rectangle.
abstract void fillArc (int x, int y, int width, int height, int startAngle, int arcAngle)	Fills a circular or elliptical arc covering the specified rectangle.
abstract void fillOval (int x, int y, int width, int height)	Fills an oval bounded by the specified rectangle with the current colour.
abstract void fillPolygon (int[] xPoints, int[] yPoints, int nPoints)	Fills a closed polygon defined by arrays of x and y coordinates.
void fillPolygon (Polygon p)	Fills the polygon defined by the specified Polygon object with the graphics context's current colour.
abstract void fillRect (int x, int y, int width, int height)	Fills the specified rectangle.
abstract void fillRoundRect (int x, int y, int width, int height, int arcWidth, int arcHeight)	Fills the specified rounded corner rectangle with the current colour.
void fill3DRect (int x, int y, int width, int height, boolean raised)	Paints a 3-D highlighted rectangle filled with the current colour.
abstract void clearRect (int x, int y, int width, int height)	Clears the specified rectangle by filling it with the background colour of the current drawing surface.
abstract Color getColor ()	Gets this graphics context's current colour.
abstract Font getFont ()	Gets the current font.

FontMetrics getFontMetrics ()	Gets the font metrics of the current font.
abstract void setColor (Color c)	Sets this graphics context's current color to the specified colour.
abstract void setFont (Font font)	Sets this graphics context's font to the specified font.

Table 13.1

General Syntax for using AWT application is-

```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;       // package for 2D Geometry

public class AwtClass extends Frame
{
  public AwtClass ()          // constructor
  {
    super("Text ");
    prepareGUI();
  }
  public static void main(String[] args)
  {
    AwtClass obj = new AwtClass( );        // object of graphical class
    obj.setVisible(true);
  }
  private void prepareGUI()
  {
       setSize(400,400);
       addWindowListener ( new WindowAdapter()
       {
         public void windowClosing(WindowEvent windowEvent)
           {
              System.exit(0);
           }
       });
```

```
        }
        @Override
        public void paint(Graphics g)
        {
          g.setColor(Color.GRAY);
          Font font = new Font("Serif", Font.PLAIN, 24);
          g.setFont(font);
          g.drawString("Text", 50, 150);
        }
      }
```

Above is the general syntax for AWT application.

Example: Unit-13 (Program-01)

/* Write a program to display the message in AWT program. */

```
import java.awt.*;
import java.awt.event.*;
public class unit13p01 extends Frame
{
  public unit13p01(){
    super("AWT Testing");
    prepareGUI();
  }
  public static void main(String[] args)
  {
    unit13p01 obj = new unit13p01();
    obj.setVisible(true);
  }
  private void prepareGUI()
  {
    setSize(400,400);
    addWindowListener(new WindowAdapter()
    {
      public void windowClosing(WindowEvent windowEvent)
      {
```

```
            System.exit(0);
        }
      });
    }
    @Override
    public void paint(Graphics g)
    {
      g.setColor(Color.GRAY);
      Font font = new Font("Serif", Font.PLAIN, 24);
      g.setFont(font);
      g.drawString("Ankur Mittal", 50, 150);
    }
}
```

Result:

Above will be compiled and executed in the same way like other java programs are done.

Example: Unit-13 (Program-02)

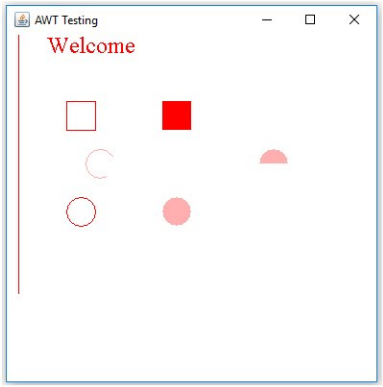
/* Write a program to display some shapes in AWT frame. */

```
import java.awt.*;
import java.awt.event.*;
public class unit13p02 extends Frame
{
```

```
public unit13p02()
{
        super("AWT Testing");
        prepareGUI();
}
public static void main(String[] args)
{
        unit13p02 obj = new unit13p02();
        obj.setVisible(true);
}
private void prepareGUI()
{
        setSize(400,400);
        addWindowListener (new WindowAdapter()
        {
                public void windowClosing(WindowEvent windowEvent)
                {
                System.exit(0);
                }
        } );
}
@Override
public void paint(Graphics g)
{
        g.setColor(Color.GRAY);
        Font font = new Font("Serif", Font.PLAIN, 24);
        g.setFont(font);
        g.setColor(Color.red);
        g.drawString("Welcome",50, 50);
        g.drawLine(20,30,20,300);
        g.drawRect(70,100,30,30);
```

```
                g.fillRect(170,100,30,30);
                g.drawOval(70,200,30,30);
                g.setColor(Color.pink);
                g.fillOval(170,200,30,30);
                g.drawArc(90,150,30,30,30,270);
                g.fillArc(270,150,30,30,0,180);
        }
    }
```

Result:



There are some classes of Graphics Controls-

Control	Purpose
Graphics	It is the top level abstract class for all graphics contexts.
Graphics2D	It is a subclass of Graphics class and provides more sophisticated control over geometry, coordinate transformations, color management, and text layout.
Arc2D	Arc2D is the abstract superclass for all objects that store a 2D arc defined by a framing rectangle, start angle, angular extent (length of the arc), and a closure type (OPEN, CHORD, or PIE).
CubicCurve2D	The CubicCurve2D class is the abstract superclass fpr all objects which store a 2D cubic curve segment and it defines a cubic parametric curve segment in (x,y) coordinate space.
Ellipse2D	The Ellipse2D is the abstract superclass for all objects which store a 2D ellipse and it describes an ellipse that is defined by

	a framing rectangle.
Rectangle2D	The Rectangle2D class is an abstract superclass for all objects that store a 2D rectangle and it describes a rectangle defined by a location (x,y) and dimension (w x h).
QuadCurve2D	The QuadCurve2D class is an abstract superclass for all objects that store a 2D quadratic curve segment and it describes a quadratic parametric curve segment in (x,y) coordinate space.
Line2D	This Line2D represents a line segment in (x,y) coordinate space.
Font	The Font class represents fonts, which are used to render text in a visible way.
Color	The Color class is used encapsulate colors in the default sRGB color space or colors in arbitrary color spaces identified by a ColorSpace.
BasicStroke	The BasicStroke class defines a basic set of rendering attributes for the outlines of graphics primitives, which are rendered with a Graphics2D object that has its Stroke attribute set to this BasicStroke.

Table 13.2

Check Your Progress

1. What is AWT in java? Compare SWT program with Applet.
2. List the methods of Graphics class. Explain any 5 methods of Graphics class
3. Design an AWT program to show the hut of AWT Frame.
4. Design an AWT program to smiley in AWT Frame.
5. Compare AWT with Applet.

13.4 USER INTERFACE COMPONENTS

User interface is the window on computer screen which is used by end user to use the java application. There are several components which are used to develop the graphical interface for end user.

Java includes libraries to provide multi-platform support for Graphic User Interface objects. The "multi-platform" aspect of this is that we can write a

program on a Macintosh and have the same graphic objects show up when the program is run under UNIX or Windows. Java's GUI components include labels, text fields, text areas, buttons, etc. The Abstract Windowing Toolkit (AWT) also includes containers which can include these components. Containers include frames (windows), canvases (which are used to draw on), and panels (which are used to group components). Panels and canvases are contained in frames (windows) while buttons and other components can be placed either directly on frames or in panels inside the frames. These GUI components are automatically drawn whenever the window they are in is drawn. These GUI components are handled using Java's event model.

There are current three sets of Java APIs for graphics programming: AWT (Abstract Windowing Toolkit), Swing and JavaFX-

- AWT API was introduced in JDK 1.0. Most of the AWT components have become obsolete and should be replaced by newer Swing components.
- Swing API, a much more comprehensive set of graphics libraries that enhances the AWT, was introduced as part of Java Foundation Classes (JFC).
- The latest JavaFX, which was integrated into JDK 8, is meant to replace Swing.

Hierarchy of AWT controls in java-

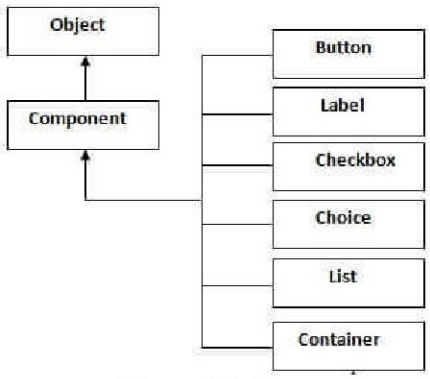


Figure 13.1

COMPONENTS OF AWT	
Class	**Purpose**
Label	A Label object is a component for placing text in a container.
Button	This class creates a labeled button.
CheckBox	A check box is a graphical component that can be in either an on (true) or off (false) state.

CheckBoxGroup	The CheckboxGroup class is used to group the set of checkbox.
List	The List component presents the user with a scrolling list of text items.
TextField	A TextField object is a text component that allows for the editing of a single line of text.
TextArea	A TextArea object is a text component that allows for the editing of a multiple lines of text.
Choice	A Choice control is used to show pop up menu of choices. Selected choice is shown on the top of the menu.
Canvas	A Canvas control represents a rectangular area where application can draw something or can receive inputs created by user.
Image	An Image control is superclass for all image classes representing graphical images.
ScrollBar	A Scrollbar control represents a scroll bar component in order to enable user to select from range of values.
Dialog	A Dialog control represents a top-level window with a title and a border used to take some form of input from the user.
FileDialog	A FileDialog control represents a dialog window from which the user can select a file.
Container	It can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

Table 13.3

13.4.1 BUILDING USER INTERFACE WITH AWT

For creating graphical applications, we have to use AWT package. AWT is huge! It consists of 12 packages of 370 classes.

Fortunately, only 2 packages - java.awt and java.awt.event - are commonly-used.

1. The java.awt package contains the *core* AWT graphics classes:
   - GUI Component classes, such as Button, TextField, and Label.
   - GUI Container classes, such as Frame and Panel.
   - Layout managers, such as FlowLayout, BorderLayout and GridLayout.

- Custom graphics classes, such as Graphics, Color and Font.

2. The java.awt.event package supports event handling:
   - Event classes, such as Action Event, Mouse Event, Key Event and Window Event,
   - Event Listener Interfaces, such as ActionListener, MouseListener, MouseMotionListener, KeyListener and WindowListener,
   - Event Listener Adapter classes, such as MouseAdapter, KeyAdapter, and WindowAdapter.

   AWT provides a *platform-independent* and *device-independent* interface to develop graphic programs that runs on all platforms, including Windows, Mac OS X, and Unixes.

   There are two types of GUI elements:

- Component: Components are elementary GUI entities, such as Button, Label, and TextField.
- Container: Containers, such as Frame and Panel, are used to hold components in a specific layout. A container can also hold sub-containers.

13.4.2 CONTAINER

The most basic GUI elements you will be dealing with in Java include what are known as containers and components. A container is used to store the items the user will see on display, such as a panel or a frame (main window). These are the very core of what holds everything together and are the primary skeletal system of the GUI as a whole.

Components can be described as the add-ons or extensions to the containers that help to fill in the content and make up the rest of the user interface. Examples can include things such as a textbox, a button, a checkbox, or even a simple label. When you combine both containers and components, you begin to understand the bigger picture of what the GUI is and why it represents the core of the user experience for any program or application.

The hierarchy of the AWT Container classes is-

CONTAINER OF AWT	
Class	**Purpose**
Container	It can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.
Window	It is the container that have no borders and menu bars. You must

	use frame, dialog or another window for creating a window.
Panel	It is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.
Frame	It is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.
ScrollPane	It provides automatic horizontal and/or vertical scrolling for a single child component.

Table 13.4

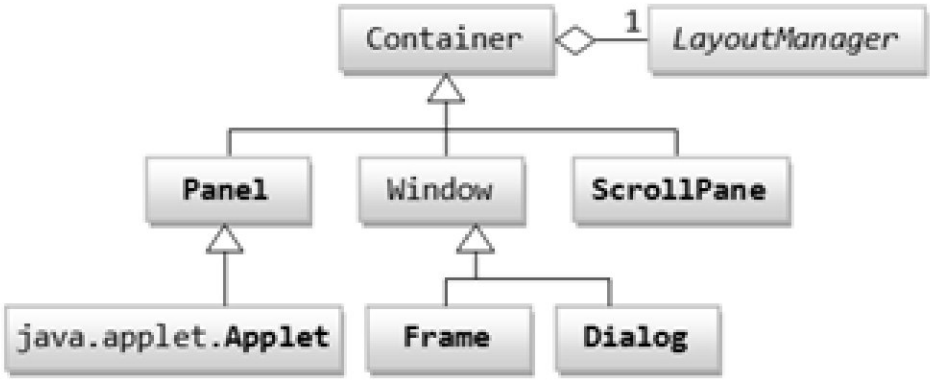


Figure 13.2

There are two ways to create a frame in AWT.

- By extending Frame class (inheritance)
- By creating the object of Frame class (association)

Here are some components with their constructor and methods-

TextField:	**Button:**

public TextField(String initialText, int columns);
public TextField(String initialText);
public TextField(int columns);
public String getText();
public void setText(String strText);
public void setEditable(boolean editable);
	public Button(String btnLabel);
	public Button();
	public String getLabel();
	public void setLabel(String btnLabel);
	public void setEnable(boolean enable);

Label:	**Frame:**

public Label(String strLabel, int alignment); public Label(String strLabel);
public Label();
public String getText();
public void setText(String strLabel);
public int getAlignment();
public void setAlignment(int alignment);
	Frame( )
	Frame(String title)
	void setSize(int newWidth, int newHeight)
	void setSize(Dimension newSize)
	Dimension getSize( )
	void setVisible(boolean visibleFlag)
	void setTitle(String newTitle)

Color: Color(int red, int green, int blue) Color(int rgbValue) Color(float red, float green, float blue) int getRed( ) int getGreen( ) int getBlue( ) void setColor(Color newColor)	**Font:** Font(String fName, int fStyle, int pointSize) String getFontName() int getSize( ) int getStyle( ) boolean isBold( ) boolean isItalic( ) boolean isPlain( ) Font getFont( ) void setFont(Font fontObj)
CheckBox: Checkbox( ) Checkbox(String str) Checkbox(String str, boolean on) Checkbox(String str, boolean on, CheckboxGroup cbGroup) Checkbox(String str, CheckboxGroup cbG, boolean on) boolean getState( ) void setState(boolean on) String getLabel( )	**List:** List( ) List(int numRows) List(int numRows, boolean multipleSelect) void add(String name) void add(String name, int index) String getSelectedItem( ) int getSelectedIndex( ) String[ ] getSelectedItems( )

void setLabel(String str)	int[ ] getSelectedIndexes( ) int getItemCount( ) void select(int index) String getItem(int index)
Scrollbar : Scrollbar( ) Scrollbar(int style) Scrollbar(int style, int initialValue, int thumbSize, int min, int max) int getValue( ) void setValue(int newValue) int getMinimum( ) int getMaximum( ) void setUnitIncrement(int newIncr) void setBlockIncrement(int newIncr)	**TextArea:** TextArea( ) TextArea(int numLines, int numChars) TextArea(String str) TextArea(String str, int numLines, int numChars) TextArea(String str, int numLines, int numChars, int sBars) void append(String str) void insert(String str, int index) void replaceRange(String str, int startIndex, int endIndex)
Choice: void addItem(String name) void add(String name) String getSelectedItem( ) int getSelectedIndex( ) int getItemCount( ) void select(int index) void select(String name) String getItem(int index) **Image:** Image createImage(ImageProducer imgProd) Image createImage(int width, int height) Image getImage(URL url) Image getImage(URL url, String	**Menu:** Menu( ) Menu(String optionName) Menu(String optionName, boolean removable) MenuItem( ) MenuItem(String itemName) MenuItem(String itemName, MenuShortcut keyAccel) void setEnabled(boolean enabledFlag) boolean isEnabled( ) void setLabel(String newName) String getLabel( ) MenuItem add(MenuItem item) Menu add(Menu menu)

imageName) boolean drawImage(Image imgObj, int left, int top, ImageObserver imgOb)	Object getItem( )
Dialog: Dialog(Frame parentWindow, boolean mode) Dialog(Frame parentWindow, String title, boolean mode)	**FileDialog :** FileDialog(Frame parent, String boxName) FileDialog(Frame parent, String boxName, int how) FileDialog(Frame parent) String getDirectory( ) String getFile( )

Table 13.5

Example: Unit-13 (Program-03)

/* Write a program to display some message in text box by button click in AWT frame. */

```
import java.awt.*;
import java.awt.event.*;
public class unit13p03 extends Frame implements ActionListener
{
        TextField tf;
        unit13p03 ()
        {
                tf=new TextField();
                tf.setBounds(60,50,170,20);
                Button b=new Button("click me");
                b.setBounds(100,120,80,30);
                b.addActionListener(this);//passing current instance
                add(b);
                add(tf);
                setSize(300,300);
```

```
                setLayout(null);
                addWindowListener (new WindowAdapter()
                {
                        public void windowClosing(WindowEvent windowEvent)
                        {
                        System.exit(0);
                        }
                } );
                setVisible(true);
        }
        public void actionPerformed(ActionEvent e)
        {
                tf.setText("Welcome Ankur Mittal");
        }
        public static void main(String args[])
        {
                new unit13p03 ();
        }
}
```

Result:

Example: Unit-13 (Program-04)

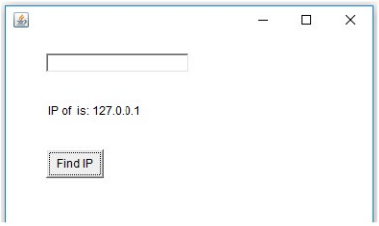
/* Write a program to display IP Address of machine in AWT frame. */

```
import java.awt.*;
import java.awt.event.*;
```

```
public class unit13p04 extends Frame implements ActionListener
{
    TextField tf; Label l; Button b;
     unit13p04 ()
     {
     tf=new TextField();
     tf.setBounds(50,50, 150,20);
              l=new Label();
              l.setBounds(50,100, 250,20);
              b=new Button("Find IP");
              b.setBounds(50,150,60,30);
              b.addActionListener(this);
              add(b);add(tf);add(l);
              setSize(400,400);
              setLayout(null);
              setVisible(true);
     addWindowListener (new WindowAdapter()
     {
              public void windowClosing(WindowEvent windowEvent)
              {
                System.exit(0);
              }
     } );
   }
   public void actionPerformed(ActionEvent e)
   {
     try
     {
              String host=tf.getText();
              String
ip=java.net.InetAddress.getByName(host).getHostAddress();
```

```
        l.setText("IP of "+host+" is: "+ip);
        }
        catch(Exception ex)
        {
        System.out.println(ex);
        }
    }
    public static void main(String[] args)
    {
        new unit13p04 ();
    }
}
```

Result:



Example: Unit-13 (Program-05)

/* Write a program to perform addition and subtraction in java AWT controls. */

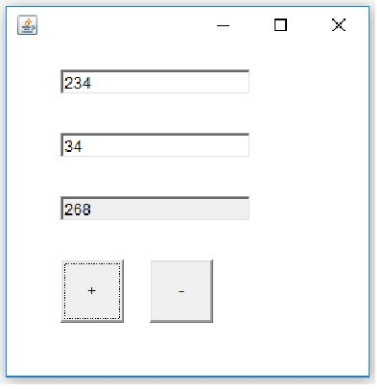
```
import java.awt.*;
import java.awt.event.*;
public class unit13p05 extends Frame implements ActionListener
{
        TextField tf1,tf2,tf3;
        Button b1,b2;
        unit13p05 ()
        {
                tf1=new TextField();
```

```
        tf1.setBounds(50,50,150,20);
        tf2=new TextField();
        tf2.setBounds(50,100,150,20);
        tf3=new TextField();
        tf3.setBounds(50,150,150,20);
        tf3.setEditable(false);
        b1=new Button("+");
        b1.setBounds(50,200,50,50);
        b2=new Button("-");
        b2.setBounds(120,200,50,50);
        b1.addActionListener(this);
        b2.addActionListener(this);
        add(tf1);
        add(tf2);
        add(tf3);
        add(b1);
        add(b2);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
        addWindowListener (new WindowAdapter()
        {
          public void windowClosing(WindowEvent windowEvent)
          {
              System.exit(0);
          }
        } );
    }
    public void actionPerformed(ActionEvent e)
    {
```

```
        String s1=tf1.getText();
        String s2=tf2.getText();
        int a=Integer.parseInt(s1);
        int b=Integer.parseInt(s2);
        int c=0;
        if(e.getSource()==b1)
        {
           c=a+b;
        }
       else if(e.getSource()==b2)
       {
           c=a-b;
        }
        String result=String.valueOf(c);
        tf3.setText(result);
    }

   public static void main(String[] args)
   {
      new unit13p05 ();
    }
}
```

Result:



Example: Unit-13 (Program-06)

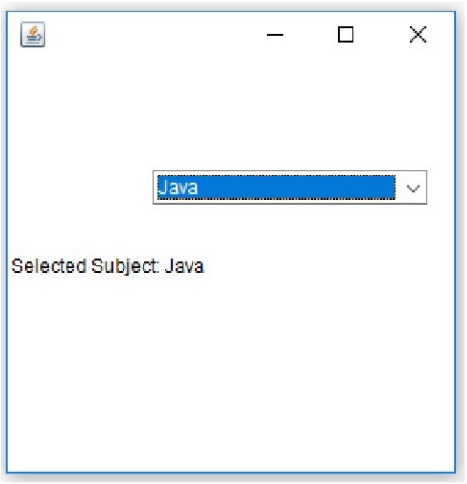
/* Write a program to show subject list and display selected subject. */

```
import java.awt.*;
import java.awt.event.*;

public class unit13p06 extends Frame implements ItemListener
{
        Choice sub;
        Label lbl;
        String msg;
        unit13p06 ()
        {
       sub=new Choice();
       sub.setBounds(100,100, 175,75);
       lbl=new Label(" Choice");
       lbl.setSize(200,150);
       sub.add("C++");
       sub.add("Java");
       sub.add("Data Base Management System");
       add(sub);
       add(lbl);
       sub.addItemListener(this);
       setSize(300,300);
       setVisible(true);
       addWindowListener (new WindowAdapter()
       {
                public void windowClosing(WindowEvent windowEvent)
                {
                  System.exit(0);
                }
       } );
     }
        public void itemStateChanged(ItemEvent ie)
```

```
    {
        String str="Selected Subject: "+sub.getItem(sub.getSelectedIndex());
        lbl.setText(str);
    }
    public static void main(String[] args)
    {
        new unit13p06 ();
    }
}
```

Result:



Check Your Progress

1. Describe all container of java AWT.
2. Describe all component in java AWT.
3. Design a program in java to calculate the simple interest.
4. Design a program in java to find the greatest of three numbers.
5. Design a program in java for login with user name and password. Accept the user name and password and validate it with any specific value.
6. Design a calculator in java and perform the operations.

13.5 SWING - BASED GUI

Swing can be described as the newest form of the two, doing the same thing as AWT, but as extensions to the AWT itself. These are known as the lightweight

components and are entirely independent of the platform or operating system, which give them much more freedom and flexibility than the AWT.

Hierarchy of Java Swing Components-

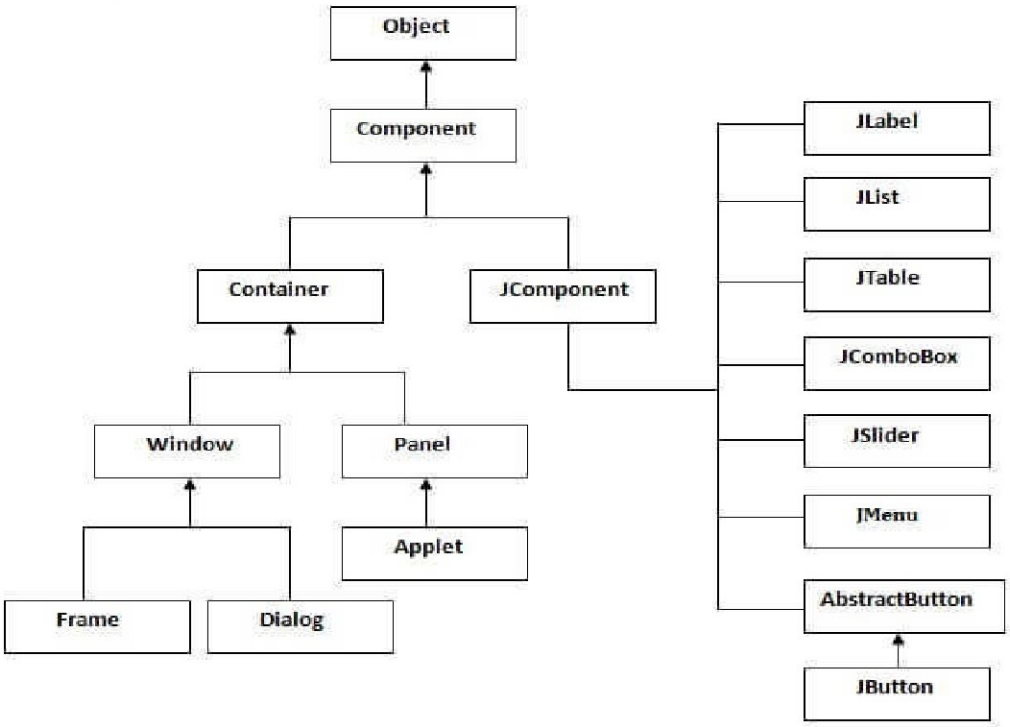


Figure 13.3

Java AWT vs. Java Swing	
Java AWT	**Java Swing**
AWT components are platform-dependent.	Java swing components are platform-independent.
AWT components are heavyweight.	Swing components are lightweight.
AWT doesn't support pluggable look and feel.	Swing supports pluggable look and feel.
AWT provides less components than Swing.	Swing provides more powerful components such as tables, lists, colorchooser, tabbedpane etc.
AWT doesn't follows MVC (Model View Controller).	Swing follows MVC.

Table 13.6

There are some popular swing controls-

Class	Purpose
JLabel	A JLabel object is a component for placing text in a container.
JButton	This class creates a labeled button.
JColorChooser	A JColorChooser provides a pane of controls designed to allow a user to manipulate and select a color.
JCheck Box	A JCheckBox is a graphical component that can be in either an on (true) or off (false) state.
JRadioButton	The JRadioButton class is a graphical component that can be in either an on (true) or off (false) state. in a group.
JList	A JList component presents the user with a scrolling list of text items.
JComboBox	A JComboBox component presents the user with a to show up menu of choices.
JTextField	A JTextField object is a text component that allows for the editing of a single line of text.
JPasswordField	A JPasswordField object is a text component specialized for password entry.
JTextArea	A JTextArea object is a text component that allows editing of a multiple lines of text.
ImageIcon	A ImageIcon control is an implementation of the Icon interface that paints Icons from Images
JScrollbar	A Scrollbar control represents a scroll bar component in order to enable the user to select from range of values.
JOptionPane	JOptionPane provides set of standard dialog boxes that prompt users for a value or informs them of something.
JFileChooser	A JFileChooser control represents a dialog window from which the user can select a file.
JProgressBar	As the task progresses towards completion, the progress bar

	displays the task's percentage of completion.
JSlider	A JSlider lets the user graphically select a value by sliding a knob within a bounded interval.
JSpinner	A JSpinner is a single line input field that lets the user select a number or an object value from an ordered sequence.

Table 13.7

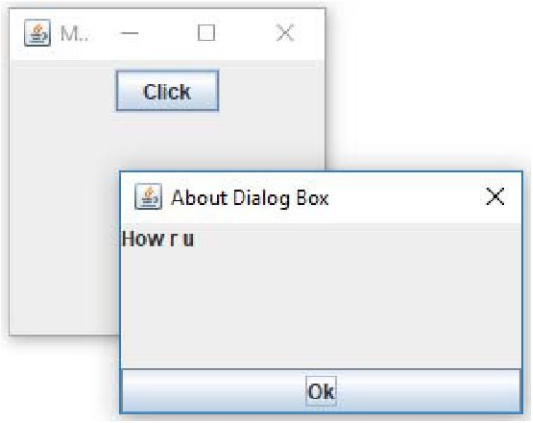
Example: Unit-13 (Program-07)

/* Write a program to show dialogue box by click on button in swing window. */

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class unit13p07
{
 public static void main(String[] args)
 {
   JFrame f=new JFrame("My Frame");
   Container contentPane =f.getContentPane();
   f.setSize(200,200);
   f.setLayout(new FlowLayout());
   JButton jb1=new JButton("Click");
   jb1.setActionCommand("Click");
   jb1.addActionListener(new ActionListener()
   { public void actionPerformed(ActionEvent ae)
    { String str=(String)ae.getActionCommand();
    if(str.equals("Click"))
    {
      MyDialog dia=new MyDialog(new JFrame("Dialog Box"));
      dia.setVisible(true);
    }} });
   contentPane.add(jb1);
   f.setVisible(true);
```

```
    }
  }
  class MyDialog extends JDialog
  {
     public MyDialog (JFrame parent)
     {
        super(parent, "About Dialog Box", true);
        setSize(250, 150);
        JLabel l1=new JLabel("How r u");
        JButton ok = new JButton("Ok");
        getContentPane().add(l1, "North");
        getContentPane().add(ok, "South");
        ok.addActionListener(new ActionListener() { public void
actionPerformed(ActionEvent evt) { setVisible(false);} });
     }
  }
```

Result:



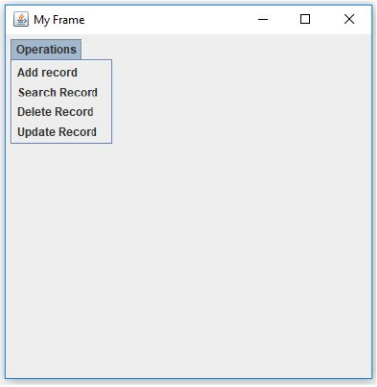
Example: Unit-13 (Program-08)

/* Write a program to add menus in swing window. */

```
  import java.awt.*;
  import javax.swing.*;
  public class unit13p08
  {
```

```
    public static void main(String[] str)
    {
     JFrame f=new JFrame("My Frame");
     f.setSize(400,400);
     f.setLayout(new FlowLayout(FlowLayout.LEFT));
     Container contentPane = f.getContentPane();
     JMenuItem i1=new JMenuItem("Add record");
     JMenuItem i2=new JMenuItem("Search Record");
     JMenuItem i3=new JMenuItem("Delete Record");
     JMenuItem i4=new JMenuItem("Update Record");
     JMenu m1=new JMenu("Operations");
     m1.add(i1);
     m1.add(i2);
     m1.add(i3);
     m1.add(i4);
     JMenuBar jm=new JMenuBar();
     jm.add(m1);
     f.add(jm);
     f.setVisible(true);
    }
  }
```

Result:



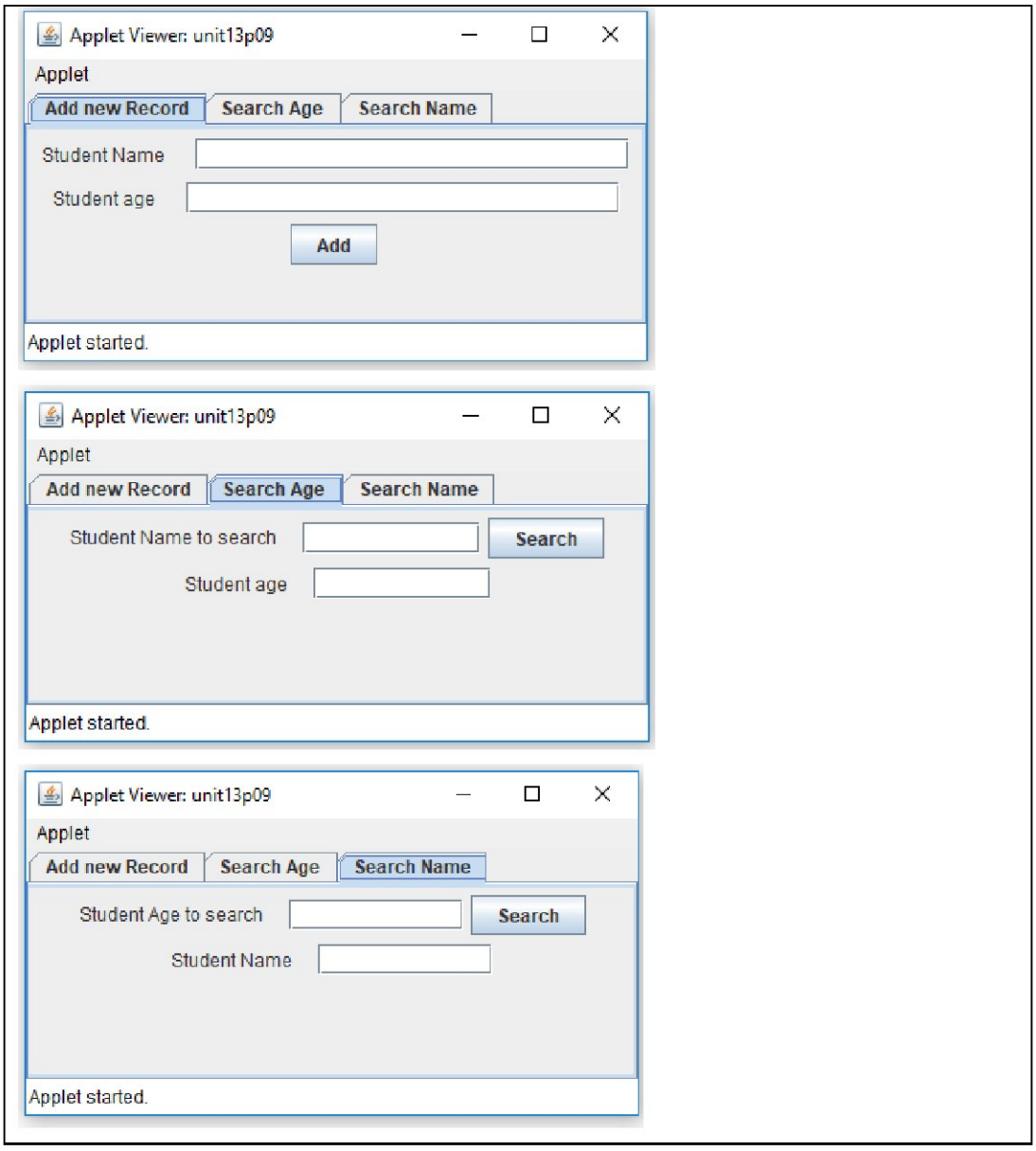
Example: Unit-13 (Program-09)

/* Write a program to add tabbed pane in swing. */

```
import java.awt.*;
import javax.swing.*;
/* <applet code="unit13p09" width=400 height=150> </applet>*/
public class unit13p09 extends JApplet
{
public void init()
{
 JTabbedPane jtp = new JTabbedPane();
 jtp.addTab("Add new Record", new addstudent());
 jtp.addTab("Search Age", new searchAge());
 jtp.addTab("Search Name", new searchName());
 getContentPane().add(jtp);
 }
}
class addstudent extends JPanel
{
  public addstudent()
  {
    Label l1=new Label("Student Name ");
    JTextField t1=new JTextField(25);
    Label l2=new Label("Student age ");
    JTextField t2=new JTextField(25);
    JButton b1=new JButton("Add");
    add(l1);
    add(t1);
    add(l2);
    add(t2);
    add(b1);
  }
}
class searchAge extends JPanel
{
  public searchAge()
```

```
    {
     Label l1=new Label("Student Name to search");
     JTextField t3=new JTextField(10);
     JButton b2=new JButton("Search");
     Label l2=new Label("Student age");
     JTextField t4=new JTextField(10);
     add(l1);
     add(t3);
     add(b2);
     add(l2);
     add(t4);
    }
  }
  class searchName extends JPanel
  {
   public searchName()
   {
     Label l1=new Label("Student Age to search");
     JTextField t5=new JTextField(10);
     JButton b3=new JButton("Search");
     Label l2=new Label("Student Name");
     JTextField t6=new JTextField(10);
     add(l1);
     add(t5);
     add(b3);
     add(l2);
     add(t6);
   }
  }
```

Result:



Check Your Progress

1. What is swing in java?
2. Why we need swing? Explain advantages of swing.
3. Design a program in java swing to find the greater of two numbers.
4. Design a program in java swing to implement all four arithmetic operation in different 4 panes in the tabbed panes.

13.6 LAYOUTS AND LAYOUT MANAGER

Layout defines the way components should be arranged in the container. We define the layout before adding the component in the container. If we don't assign any layout then the default layout will automatically be used by java.

For managing the layout there is a layout manager. It automatically positions all the components within the container.

If we have too many components then managing the component is difficult. Besides that we also have to manage the information height and width of components. So due to these two reasons we have to use layout manager.

Java provide us with various layout manager to position the controls. The properties like size, shape and arrangement varies from one layout manager to other layout manager.

On changing the size of container window, layout manager also adjust accordingly.

The layout manager is associated with every Container object. Each layout manager is an object of the class that implements the LayoutManager interface.

Types of layout:

There are 5 types of layout in java:

- Border Layout: The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window.
- FlowLayout: The FlowLayout is the default layout. It layouts the components in a directional flow.
- GridLayout: The GridLayout manages the components in form of a rectangular grid.
- CardLayout: The CardLayout object treats each component in the container as a card. Only one card is visible at a time.
- GridBagLayout: This is the most flexible layout manager class. The object of GridBagLayout aligns the component vertically, horizontally or along their baseline without requiring the components of same size.

Example: Unit-13 (Program-10)

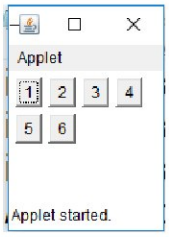
/* Write a program to implement flow layout. */

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
```

```
/* <applet code="unit13p10" width=100 height=100> </applet> */
public class unit13p10 extends Applet implements ActionListener
{
 Button b1,b2,b3,b4,b5,b6;
 String msg;
 public void init()
 {
  setLayout(new FlowLayout(FlowLayout.LEFT));
        //FlowLayout.CENTER or FlowLayout.RIGHT
  b1= new Button("1");
  b2= new Button("2");
  b3= new Button("3");
  b4= new Button("4");
  b5= new Button("5");
  b6= new Button("6");
  add(b1);
  add(b2);
  add(b3);
  add(b4);
  add(b5);
  add(b6);
  b1.addActionListener(this);
  b2.addActionListener(this);
  b3.addActionListener(this);
  b4.addActionListener(this);
  b5.addActionListener(this);
  b6.addActionListener(this);
 }
 public void actionPerformed(ActionEvent ae)
 {
   String str = ae.getActionCommand();
```

```
        if(str.equals("1"))
          msg = "1";
        else if(str.equals("2"))
          msg = "2";
        else if(str.equals("3"))
          msg = "3";
        else if(str.equals("4"))
          msg = "4";
        else if(str.equals("5"))
          msg = "5";
        else if(str.equals("6"))
          msg = "6";
        repaint();
    }
    public void paint(Graphics g)
    {
      g.drawString(msg, 10, 80);
    }
    }
```

Result:



Example: Unit-13 (Program-11)

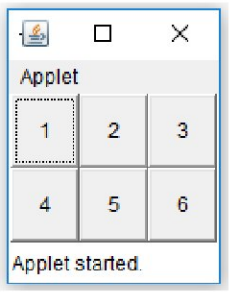
/* Write a program to implement grid layout. */

```
import java.awt.*;
import java.applet.*;
```

```
import java.awt.event.*;
/* <applet code="unit13p11" width=100 height=100> </applet> */
public class unit13p11 extends Applet implements ActionListener
{
 Button b1,b2,b3,b4,b5,b6;
 String msg;
 public void init()
 {
  setLayout(new GridLayout(2,3));
  b1= new Button("1");
  b2= new Button("2");
  b3= new Button("3");
  b4= new Button("4");
  b5= new Button("5");
  b6= new Button("6");
  add(b1);
  add(b2);
  add(b3);
  add(b4);
  add(b5);
  add(b6);
  b1.addActionListener(this);
  b2.addActionListener(this);
  b3.addActionListener(this);
  b4.addActionListener(this);
  b5.addActionListener(this);
  b6.addActionListener(this);
 }
 public void actionPerformed(ActionEvent ae)
 {
   String str = ae.getActionCommand();
```

```
    if(str.equals("1"))
      msg = "1";
    else if(str.equals("2"))
      msg = "2";
    else if(str.equals("3"))
      msg = "3";
    else if(str.equals("4"))
      msg = "4";
    else if(str.equals("5"))
      msg = "5";
    else if(str.equals("6"))
      msg = "6";
    repaint();
  }
  public void paint(Graphics g)
  {
   g.drawString(msg, 10, 80);
  }
 }
```

Result:



Example: Unit-13 (Program-12)

/* Write a program to implement border layout. */

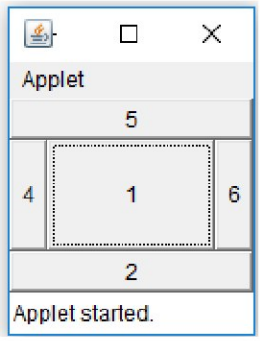
```
import java.awt.*;
```

```
import java.applet.*;
import java.awt.event.*;
/* <applet code="unit13p12" width=100 height=100> </applet> */

public class unit13p12 extends Applet implements ActionListener
{
 Button b1,b2,b3,b4,b5,b6;
 String msg;
 public void init()
 {
  setLayout(new BorderLayout());
  b1= new Button("1");
  b2= new Button("2");
  b3= new Button("3");
  b4= new Button("4");
  b5= new Button("5");
  b6= new Button("6");
  add(b1,BorderLayout.CENTER);
  add(b2,BorderLayout.SOUTH);
  add(b3,BorderLayout.EAST);
  add(b4,BorderLayout.WEST);
  add(b5,BorderLayout.NORTH);
  add(b6,BorderLayout.EAST);
  b1.addActionListener(this);
  b2.addActionListener(this);
  b3.addActionListener(this);
  b4.addActionListener(this);
  b5.addActionListener(this);
  b6.addActionListener(this);
 }
 public void actionPerformed(ActionEvent ae)
```

```
    {
      String str = ae.getActionCommand();
      if(str.equals("1"))
        msg = "1";
      else if(str.equals("2"))
        msg = "2";
      else if(str.equals("3"))
        msg = "3";
      else if(str.equals("4"))
        msg = "4";
      else if(str.equals("5"))
        msg = "5";
      else if(str.equals("6"))
        msg = "6";
      repaint();
    }
    public void paint(Graphics g)
    {
      g.drawString(msg, 10, 80);
    }
  }
```

Result:



Example: Unit-13 (Program-13)

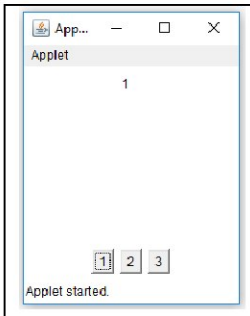
/* Write a program to implement card layout. */

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/* <applet code="unit13p13" width=200 height=200> </applet> */
public class unit13p13 extends Applet implements ActionListener
{
 Button b1,b2,b3;
 CardLayout cl;
 Panel p1,p2,p3,bp,lp;
 Label l1,l2,l3;
 public void init()
 {
  p1=new Panel();  //create panels
  p2=new Panel();
  p3=new Panel();
  bp=new Panel();  //button panel
  lp=new Panel();   //label panel

  l1=new Label("1");
  l2=new Label("2");
  l3=new Label("3");
  p1.add(l1);       // add label in different panels
  p2.add(l2);
  p3.add(l3);
  b1= new Button("1");
  b2= new Button("2");
  b3= new Button("3");
  b1.addActionListener(this);
```

```
    b2.addActionListener(this);
    b3.addActionListener(this);
    bp.add(b1);   //add buttons in 1 panel
    bp.add(b2);
    bp.add(b3);
    cl=new CardLayout();
    lp.setLayout(cl);    //set card layout for label panel
    lp.add(p1,"p1");   //add all panel in card
    lp.add(p2,"p2");
    lp.add(p3,"p3");
    setLayout(new BorderLayout());
    add(lp,BorderLayout.CENTER);   //show card panel in center
    add(bp,BorderLayout.SOUTH);   //show button panel in bottom
  }
  public void actionPerformed(ActionEvent ae)
  {
    String str=ae.getActionCommand();
    if(str.equals("1"))
      cl.show(lp,"p1");        // showing selected panel
    else if(str.equals("2"))
      cl.show(lp,"p2");
    else if(str.equals("3"))
      cl.show(lp,"p3");
  }
  public void paint(Graphics g)  { }
 }
```

Result:



Check Your Progress

1. What is layout? Explain layouts of java.
2. Write a program in java to implement all four arithmetic operations with card layout.
3. Write a program in java to design calculator.
4. Write a program in java to design notepad with border layout.

13.7 SUMMARY

Graphical development is achieved via AWT and swing in java. AWT is the earlier technique in java but its development is heavy in terms of memory consumption. Later on java introduces swing controls which are actually light weight components. So now-a-days swing is preferred over AWT controls. Both may be implemented in applet or standalone programming. Although applets are obsolete these days, still they may be implemented.

If we have several components in our design, than we may use layout in our designing. It arrange the components in our program. There are 5 types of layouts, and as per our need we may choose any of the layout.

13.8 EXERCISE

Q. 1. What is the difference between graphical design with applet and frame?

Q. 2. What is Graphics class in AWT? Explain its 5 methods.

Q. 3. Compare Swing and AWT in java.

Q. 4. What is layout manager? Explain the types of layout available in java.

Q. 5. Design a program in java of notepad with some functions.

Q. 6. Design a program in java for the game of tic-tac-toe.

Q. 7. Design the EMI calculator in java for loan.

Q. 8. Design the compound interest calculator in java.

UNIT-14 NETWORKING FEATURES

Structure :

14.1 OBJECTIVE

Objective of this unit is to explore some concept of networking in regard of programming. This unit will elaborate the socket programming and its implementation in java. How communication between devices is done, will be shown in this unit.

14.2 INTRODUCTION

Term network programming is related to the programming that is executed over a number of devices in which devices are connected in a network. In java java.net package is given to provide classes and interfaces to achieve this type of programming.

14.3 SOCKET OVERVIEW

Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server. When the connection is made, the server creates a socket object on its end of the communication. The client and the server can now communicate by writing to and reading from the socket.

Java socket programming provides facility to share data between different computing devices. There are few advantages of java socket programming like sharing resources and centralize software management.

There are few term used in java socket programming-

- IP Address: IP address is a unique number assigned to a node of a network e.g. 192.168.0.1 . It is composed of octets that range from 0 to 255. It is a logical address that can be changed.
- Protocol: A protocol is a set of rules basically that is followed for communication. For example: TCP, FTP, Telnet, SMTP, POP etc.
- Port Number: The port number is used to uniquely identify different applications. It acts as a communication endpoint between applications. The port number is associated with the IP address for communication between two applications.
- MAC Address: MAC (Media Access Control) Address is a unique identifier of NIC (Network Interface Controller). A network node can have multiple NIC but each with unique MAC.
- Connection-oriented and connection-less protocol: In connection-oriented protocol, acknowledgement is sent by the receiver. So it is reliable but slow. The example of connection-oriented protocol is TCP. But, in connection-less protocol, acknowledgement is not sent by the receiver. So it is not reliable but fast. The example of connection-less protocol is UDP.
- Socket: A socket is an endpoint between two way communications.

The following steps occur when establishing a TCP connection between two computers using sockets –

- The server instantiates a ServerSocket object, denoting which port number communication is to occur on.
- The server invokes the accept() method of the ServerSocket class. This method waits until a client connects to the server on the given port.
- After the server is waiting, a client instantiates a Socket object, specifying the server name and the port number to connect to.
- The constructor of the Socket class attempts to connect the client to the specified server and the port number. If communication is established, the client now has a Socket object capable of communicating with the server.
- On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket.

After the connections are established, communication can occur using I/O streams. Each socket has both an OutputStream and an InputStream. The client's OutputStream is connected to the server's InputStream, and the client's InputStream is connected to the server's OutputStream.

TCP is a two-way communication protocol, hence data can be sent across both streams at the same time. Following are the useful classes providing complete set of methods to implement sockets.

ServerSocket :

The java.net.ServerSocket class is used by server applications to obtain a port and listen for client requests. There are few constructor and methods in this class-

Constructor / Method	Purpose
public ServerSocket(int port) throws IOException	Attempts to create a server socket bound to the specified port.
public ServerSocket(int port, int backlog) throws IOException	With the backlog parameter specifies how many incoming clients to store in a wait queue.
public ServerSocket(int port, int backlog, InetAddress address) throws IOException	With the InetAddress parameter specifies the local IP address to bind to. The InetAddress is used for servers that may have multiple IP addresses, allowing the server to specify which of its IP addresses to accept client requests on.
public ServerSocket() throws IOException	Creates an unbound server socket.
public int getLocalPort()	Returns the port that the server socket is listening on.
public Socket accept() throws IOException	Waits for an incoming client.
public void setSoTimeout (int timeout)	Sets the time-out value for how long the server socket waits for a client during the accept().
public void bind (SocketAddress host, int backlog)	Binds the socket to the specified server and port in the SocketAddress object.

Table 14.1

Socket Class :

The java.net.Socket class represents the socket that both the client and the server use to communicate with each other. The client obtains a Socket object by

instantiating one, whereas the server obtains a Socket object from the return value of the accept() method. There are few constructor and methods in this class-

Method	Description
public Socket(String host, int port) throws UnknownHostException, IOException.	This method attempts to connect to the specified server at the specified port.
public Socket (InetAddress host, int port) throws IOException	This method is identical to the previous constructor, except that the host is denoted by an InetAddress object.
public Socket(String host, int port, InetAddress localAddress, int localPort) throws IOException.	Connects to the specified host and port, creating a socket on the local host at the specified address and port.
public Socket(InetAddress host, int port, InetAddress localAddress, int localPort) throws IOException.	This method is identical to the previous constructor, except that the host is denoted by an InetAddress object instead of a String.
public Socket()	Creates an unconnected socket. Use the connect() method to connect this socket to a server.
public void connect (SocketAddress host, int timeout) throws IOException	This method connects the socket to the specified host.
public InetAddress getInetAddress()	This method returns the address of the other computer that this socket is connected to.
public int getPort()	Returns the port the socket is bound to on the remote machine.
public int getLocalPort()	Returns the port the socket is bound to on the local machine.
public SocketAddress getRemoteSocketAddress()	Returns the address of the remote socket.
public InputStream getInputStream() throws	Returns the input stream of the socket. The input stream is connected to the output stream of the

IOException	remote socket.
public OutputStream getOutputStream() throws IOException	Returns the output stream of the socket. The output stream is connected to the input stream of the remote socket.
public void close() throws IOException	Closes the socket, which makes this Socket object no longer capable of connecting again to any server.

Table 14.2

InetAddress Class :

This class represents an Internet Protocol (IP) address. Here are following usefull methods which you would need while doing socket programming –

static InetAddress getByAddress (byte[] addr)	Returns an InetAddress object given the raw IP address.
static InetAddress getByAddress (String host, byte[] addr)	Creates an InetAddress based on the provided host name and IP address.
static InetAddress getByName (String host)	Determines the IP address of a host, given the host's name.
String getHostAddress()	Returns the IP address string in textual presentation.
String getHostName()	Gets the host name for this IP address.
static InetAddress InetAddress getLocalHost()	Returns the local host.
String toString()	Converts this IP address to a String.

Table 14.3

Check Your Progress

1. What is socket in java? How it may be implemented in java?
2. What is the InetAddress class in java? Describe its important methods.
3. Explain ServerSocket class of java.

14.4 RESERVED PARTS AND PROXY SERVERS

Computer networking based on connected-oriented communication, port is like a medium through which, one application establish a connection with another application by binding a socket by a port number. Addressing the information and the port no., accompanied the data transfer over the network. The Ports are used by TCP and UDP to deliver the data to the right application, are identified by a 16-bit number present in the header of a data packet. Ports are typically used to map data to a particular process running on a client. If we consider a letter (data packet) sent to a particular apartment (IP) with house no. (port no), at this time the port no. is the most important part for the delivery of the letter. In order for the delivery to work, the sender needs to include a house number along with the address to ensure the letter gets to the right destination. If we consider the client-server architecture, a server application binds a socket to a specific port number in connection-based communication. It registered the server with the system where all the data destined for that port.

Now we are aware of the importance of the port number. In the same order there are some ports which are predefine and called reserved ports.

Some of them are given in this table:-

Service	**Port no.**
echo	7
daytime	13
ftp	21
telnet	23
smtp	25
finger	79
http	80
pop3	110

Table 14.4

If we consider the range of the port numbers, there are 0 to 65,535 ports available. The port numbers ranging from 0 - 1023 are reserved ports or we can say that are restricted ports. All the 0 to 1023 ports are reserved for use by well-known services such as FTP, telnet and http and other system services. These ports are

called well-known ports. We can directly use port numbers from 1024 to 65535 if they are available.

We also have to take care of the port numbers which are consumed by other application which have taken port number for their execution. It means the port number, we are usung in our programming must be unique.

Proxy Server :

A proxy server is a server that sits between the client and the remote server in which the client wishes to retrieve files from. All traffic that originates from the client, is sent to the proxy server and the proxy server makes requests to the remote server on the client's behalf. Once the proxy server receives the required files, it then forwards them on to the client. This can be beneficial as it allows the proxy server administrator some control over what the machines on its network can do. For example, certain websites may be blocked by the proxy server, meaning clients will not be able to access them. It is also beneficial as frequently visited pages can be cached by the proxy server. This means that when the client (or other clients) make subsequent requests for any files that have been cached, the proxy can issue them the files straight away, without having to request them from the remote server which can be much quicker if both the proxy and the clients are on the same network. Although these files are known to be contained in the proxy's cache, it is worth noting that the clients have no knowledge of this and may be maintaining their own local caches. The benefit of the proxy cache is when multiple clients are using the proxy and thus pages cached due to one client can be accessed by another client.

Proxy server can be useful to you for a number of reasons:

- To capture traffic between a client and server.
- To limit the uploading/downloading the bandwidth to find out how your site is loaded with slow connections.
- To see how your system reacts when there is trouble in your network.
- To modify the content (sent of received) of a client/server "on the fly".
- To produce statistics about traffic.

The proxy was implemented using Java and made extensive use of TCP sockets. Firefox was set up to issue all of its traffic to the specified port and ip address which were then used in the proxy configuration. There are two main components to the implementation - the Proxy class and the RequestHandler class. The Proxy Class The Proxy class is responsible for creating a ServerSocket which can accept incoming socket connections from the client. However it is vital that the implementation be multithreaded as the server must be able to serve multiple clients simultaneously. Thus once a socket connection arrives, it is accepted and the Proxy creates a new thread which services the request (see the RequestHandler class). As the server does not need to wait for the request to be fully serviced before accepting a new socket connection, multiple clients may have their requests serviced asynchronously.

The Proxy class is also responsible for implementing caching and blocking functionality. That is, the proxy is able to cache sites that are requested by clients and dynamically block clients from visiting certain websites. As speed is of utmost importance for the proxy server, it is desirable to store references to currently blocked sites and sites that are contained in the cache in a data structure with an expected constant order lookup time. For this reason a HashMap was chosen. This results in extremely fast cache and blocked site lookup times. This results in only a small overhead if the file is not contained in the cache, and an increase in performance if the file was contained in the cache. Finally the proxy class is also responsible for the providing a dynamic console management system. This allows an administrator to add/remove files to and from the cache and websites to and from the blacklist, in real time.

Example: Unit-14 (Program-01)

/* Write a program in java to implement proxy server.*/

```
import java.io.*;
import java.net.*;

public class unit14p01
{
   public static void main(String[] args)
   {
      try {
         if (args.length != 3)
            throw new IllegalArgumentException("insuficient arguments");
         // and the local port that we listen for connections on
         String host = args[0];
         int remoteport = Integer.parseInt(args[1]);
         int localport = Integer.parseInt(args[2]);
         // Print a start-up message
         System.out.println("Starting proxy for " + host + ":" + remoteport
               + " on port " + localport);
         ServerSocket server = new ServerSocket(localport);
         while (true) {
            new ThreadProxy(server.accept(), host, remoteport);
         }
      } catch (Exception e) {
```

```
            System.err.println(e);
            System.err.println("Usage: java ProxyMultiThread "
                  + "<host> <remoteport> <localport>");
         }
      }
   }
   /* Handles a socket connection to the proxy server from the client and uses 2
      threads to proxy between server and client
   */
   class ThreadProxy extends Thread
   {
      private Socket sClient;
      private final String SERVER_URL;
      private final int SERVER_PORT;
      ThreadProxy(Socket sClient, String ServerUrl, int ServerPort)
      {
         this.SERVER_URL = ServerUrl;
         this.SERVER_PORT = ServerPort;
         this.sClient = sClient;
         this.start();
      }
      @Override
      public void run() {
         try {
            final byte[] request = new byte[1024];
            byte[] reply = new byte[4096];
            final InputStream inFromClient = sClient.getInputStream();
            final OutputStream outToClient = sClient.getOutputStream();
            Socket client = null, server = null;
            // connects a socket to the server
            try
            {
               server = new Socket(SERVER_URL, SERVER_PORT);
            }
```

```
        catch (IOException e)
        {
            PrintWriter out = new PrintWriter(new
OutputStreamWriter(outToClient));
            out.flush();
            throw new RuntimeException(e);
        }
        // a new thread to manage streams from server to client (DOWNLOAD)
        final InputStream inFromServer = server.getInputStream();
        final OutputStream outToServer = server.getOutputStream();
        // a new thread for uploading to the server
        new Thread()
        {
          public void run()
          {
            int bytes_read;
            try {
              while ((bytes_read = inFromClient.read(request)) != -1)
               {
                 outToServer.write(request, 0, bytes_read);
                 outToServer.flush();
                 //TODO CREATE YOUR LOGIC HERE
               }
            }
            catch (IOException e) {   }
            try
            {
              outToServer.close();
            }
            catch (IOException e)
            {
              e.printStackTrace();
            }
          }
```

```
        }.start();
        // current thread manages streams from server to client (DOWNLOAD)
        int bytes_read;
        try {
           while ((bytes_read = inFromServer.read(reply)) != -1)
           {
              outToClient.write(reply, 0, bytes_read);
              outToClient.flush();
              //TODO CREATE YOUR LOGIC HERE
           }
        }
        catch (IOException e)
        {
           e.printStackTrace();
        }
        finally
        {
           try
           {
              if (server != null)
                 server.close();
              if (client != null)
                 client.close();
           }
           catch (IOException e)
           {
              e.printStackTrace();
           }
        }
        outToClient.close();
        sClient.close();
     }
     catch (IOException e)
     {
```

```
            e.printStackTrace();
        }
    }
}
```

Result:

>java unit14p01 ankur 2023 2024

Starting proxy for ankur:2023 on port 2024

Please note that in the above program at runtime, 3 arguments are to be specified. First is the host name, second is remote post number and third is local port number.

Check Your Progress

1. What is reserve port in networking? Describe some reserve ports. What is the suitable range for specifying port number in java socket programming?
2. What is proxy server? Explain its uses.
3. Write a program in java to create the service which enables one server port and one local port for proxy implementation.

14.5 INTERNET ADDRESSING : DOMAIN NAMING SERVICES (DNS)

Devices connected to the Internet are called nodes. Nodes that are computers are called hosts. Each node or host is identified by at least one unique 32-bit number called an Internet address, an IP address, or a host address.

DNS is a host name to IP address translation service. DNS is a distributed database implemented in a hierarchy of name servers. It is an application layer protocol for message exchange between clients and servers. Every host is identified by the IP address but remembering numbers is very difficult for the people and also the IP addresses are not static therefore a mapping is required to change the domain name to IP address. So DNS is used to convert the domain name of the websites to their numerical IP address.

There are various kinds of DOMAIN:

- Generic domain: .com (commercial) .edu (educational) .mil(military) .org (non-profit organization) .net(similar to commercial) all these are generic domain.
- Country domain: .in (India) .us .uk

- Inverse domain: if we want to know what is the domain name of the website. Ip to domain name mapping. So DNS can provide both the mapping.

It is Very difficult to find out the ip address associated to a website because there are millions of websites and with all those websites we should be able to generate the ip address immediately.

Hierarchy of Name Servers

- *Root name servers*: It is contacted by name servers that can't resolve the name. It contacts authoritative name server if name mapping is not known. It then gets the mapping and return the IP address to the host.
- *Top level server*: It is responsible for com, org, edu etc and all top level country domains like uk, fr, ca, in etc. They have info about authoritative domain servers and know names and IP addresses of each authoritative name server for the second level domains.
- *Authoritative name servers*: This is organization's DNS server, providing authoritative hostName to IP mapping for organization servers. It can be maintained by organization or service provider.

Example: Unit-14 (Program-02)

/* Write a program to display the IP address of a website via its domain name. */

```
import java.net.*;
 class unit14p02
{
    public static void main(String args[]) throws Exception
    {
        try
        {
            InetAddress[]
address=InetAddress.getAllByName("www.google.com");
            for(int j=0;j<address.length;j++)
              System.out.println(address[j]);
        }
        catch(Exception e)
        {
            System.out.println("Error in accessing Internet :"+e);
```

```
            }
        }
    }
```

Result:

www.google.com/172.217.167.36

The IP address depends on the web server. So it may vary as per the website name.

Check Your Progress

1. What is DNS? Why we need it.
2. Write a program in a java to accept hostname from user and display its IP address.
3. Write a program in java to accept IP address from user and display its host name.

14.6 JAVA AND THE NET: URL

As many of you must be knowing that Uniform Resource Locator-URL is a string of text that identifies all the resources on Internet, telling us the address of the resource, how to communicate with it and retrieve something from it.

The URL class is the gateway to any of the resource available on the internet. A Class URL represents a Uniform Resource Locator, which is a pointer to a "resource" on the World Wide Web. A resource can point to a simple file or directory, or it can refer to a more complicated object, such as a query to a database or to a search engine.

Components of a URL:-

A URL can have many forms. The most general however follows three-components system-

- Protocol: HTTP is the protocol here
- Hostname: Name of the machine on which the resource lives.
- File Name: The path name to the file on the machine.
- Port Number: Port number to which to connect (typically optional).

Given are the constructors of URL class in java-

URL(String address)	It creates a URL object from the specified String.
URL(String protocol, String host, String file)	A URL object from the specified protocol, host, and file name.
URL(String protocol, String host, int port, String file)	Creates a URL object from protocol, host, port and file name.
URL(URL context, String spec)	Creates a URL object by parsing the given spec in the given context.
URL(String protocol, String host, int port, String file, URLStreamHandler handler)	Creates a URL object from the specified protocol, host, port number, file, and handler.
URL(URL context, String spec, URLStreamHandler handler)	Creates a URL by parsing the given spec with the specified handler within a specified context.

Table 14.5

Some methods of URL class are-

Methods name	**Purpose**
int compareTo(Object url)	Compares a URL to another URL.
boolean equals(Object obj)	Compares two URLs.
Object getContent()	Returns the contents of this URL. This method is a shorthand for: openConnection().getContent()
String getFile()	Returns the file name of this URL.
String getHost()	Returns the host name of this URL, if applicable.
int getPort()	Returns the port number of this URL.
String getProtocol()	Returns the protocol name this URL.
String getRef()	Returns the anchor (also known as the "reference") of this URL.

URLConnection openConnection()	Returns an URLConnection object that represents a connection to the remote object referred to by the URL.
InputStream openStream()	Opens a connection to this URL and returns an InputStream for reading from that connection.
boolean sameFile(URL other)	Compares two URLs, excluding the "ref" fields.
protected void set (String protocol, String host, int port, String file, String ref)	Sets the fields of the URL. This is not a public method so that only URLStreamHandlers can modify URL fields.
String toExternalForm()	Constructs a string representation of this URL.

Table 14.6

Example: Unit-14 (Program-03)

/* Write a program to create the URL object and extract information from it. */

```
import java.net.*;

public class unit14p03
{
   public static void main(String[] args) throws MalformedURLException
   {
      // creates a URL with string representation.
      URL url1 = new URL("https://www.google.com/");

      // creates a URL with a protocol,hostname,and path
      URL url2 = new URL("https", "www.google.co.in", "/imghp");

      URL url3 = new URL("https://www.google.co.in/imghp");

      // print the string representation of the URL.
      System.out.println(url1.toString());
```

```
        System.out.println(url2.toString());
        System.out.println();
        System.out.println("Different components of the URL3-");

        // retrieve the protocol for the URL
        System.out.println("Protocol:- " + url3.getProtocol());

        // retrieve the hostname of the url
        System.out.println("Hostname:- " + url3.getHost());

        // retrieve the defalut port
        System.out.println("Default port:- " + url3.getDefaultPort());

        // retrieve the query part of URL
        System.out.println("Query:- " + url3.getQuery());

        // retrive the path of URL
        System.out.println("Path:- " + url3.getPath());

        // retrive the file name
        System.out.println("File:- " + url3.getFile());

        // retrieve the reference
        System.out.println("Reference:- " + url3.getRef());
    }
}
```

Result:

https://www.google.com/

https://www.google.co.in/imghp

Different components of the URL3-

Protocol:- https

Hostname:- www.google.co.in

Default port:- 443

Query:- hl=en&tab=wi&ogbl

Path:- /imghp

File:- /imghp?hl=en&tab=wi&ogbl

Reference:- null

Check Your Progress

1. What is URL?
2. Write a program in java to accept the URL from user and display its information.
3. Write a program in java to accept the URL from user and display its reference information.

14.7 TCP/IP SOCKETS

The java.net package provides support for the two common network protocols –

- TCP – TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.
- UDP – UDP stands for User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between applications.

In java we have to do the server programming and client programming separately. Server code is executed on the server machine while client programming has to be done at client side. There is one server for one service but there may be multiple client simultaneously for same server at different client machines.

Server Programming:

To write a server socket, we have to use two classes-

- A ServerSocket: which waits for the client requests
- A plain old Socket: to use for communication with the client.

getOutputStream() method is used to send the output through the socket. After finishing, it is important to close the connection by closing the socket as well as input/output streams.

Example: Unit-14 (Program-04)

/* Write a program to create server socket to receive the request from client. */

```
import java.net.*;
import java.io.*;

public class unit14p04  // server class
{
   //initialize socket and input stream
   private Socket socket= null;
   private ServerSocket server= null;
   private DataInputStream in =  null;
    // constructor with port
   public unit14p04 (int port)
   {
      // starts server and waits for a connection
      try
      {
         server = new ServerSocket(port);
         System.out.println("Server started");

         System.out.println("Waiting for a client ...");
          socket = server.accept();
         System.out.println("Client accepted");

         // takes input from the client socket
         in = new DataInputStream(new
BufferedInputStream(socket.getInputStream()));
          String line = "";
```

```
            // reads message from client until "Over" is sent
          while (!line.equals("Over"))
          {
            try
            {
              line = in.readUTF();
              System.out.println(line);

            }
            catch(IOException i)
            {
              System.out.println(i);
            }
          }
          System.out.println("Closing connection");
           // close connection
          socket.close();
          in.close();
        }
        catch(IOException i)
        {
          System.out.println(i);
        }
    }
    public static void main(String args[])
    {
      unit14p04  server = new unit14p04 (5000);
    }
}
```

Result:

```
Server started
Waiting for a client ...
```

Client Programm :

As we have codes the server socket, client has to be coded. For client socket, we have to use following three classes-

- Socket class — to establish a connection
- DataInputStream class — to take input from terminal
- DataOutputStream class — to send output to the socket

To connect to other machine we need a socket connection. A socket connection means the two machines have information about each other's network location (IP Address) and TCP port.The java.net.Socket class represents a Socket.

To open a socket:

```
Socket socket = new Socket("127.0.0.1", 5000)
```

First argument – IP address of Server. (127.0.0.1 is the IP address of localhost, where code will run on single stand-alone machine).

Second argument – TCP Port. (Just a number representing which application to run on a server.

Example : Unit-14 (Program-05)

/* Write a program to create client socket to send request to server. */

```
import java.net.*;
import java.io.*;
 public class unit14p05  // client class
{
   // initialize socket and input output streams
   private Socket socket= null;
   private DataInputStream  input= null;
   private DataOutputStream out= null;

   // constructor to put ip address and port
   public unit14p05 (String address, int port)
   {
```

```
// establish a connection
try
{
   socket = new Socket(address, port);
   System.out.println("Connected");

   // takes input from terminal
   input  = new DataInputStream(System.in);
    // sends output to the socket
   out    = new DataOutputStream(socket.getOutputStream());
}
catch(UnknownHostException u)
{
   System.out.println(u);
}
catch(IOException i)
{
   System.out.println(i);
}
 // string to read message from input
String line = "";
 // keep reading until "Over" is input
while (!line.equals("Over"))
{
   try
   {
      line = input.readLine();
      out.writeUTF(line);
   }
   catch(IOException i)
```

```
            {
               System.out.println(i);
            }
         }
         // close the connection
         try
         {
            input.close();
            out.close();
            socket.close();
         }
         catch(IOException i)
         {
            System.out.println(i);
         }
      }
      public static void main(String args[])
      {
         unit14p05  client = new unit14p05 ("127.0.0.1", 5000);
      }
   }
```

Result:

Connected

Please note that when this client code is executed at client machine in the network, the previous server code must be running at that time. Once client is executed, server socket will give the result like:

```
Server started
Waiting for a client ...
Client accepted
```

One thing, we may also do for testing the above program. If we don't have network for testing the program, we may compile and execute the code on different command prompt at our machine.

But important thing to remember is that-

- port number must be the same and
- server socket must be executed first.

Check Your Progress

1. What is TCP socket in java?
2. Why we need InputStream and OutputStream classes in receiving data in java?
3. Write a program in java to send student record using TCP protocol to other machine.
4. Write a program in java to receive the student record using TCP protocol from other machine.

14.8 DATAGRAMS

TCP/IP-style networking provides a serialized, predictable, reliable stream of packet data. This is not without its cost, however. TCP includes algorithms for dealing with congestion control on crowded networks, as well as pessimistic expectations about packet loss. This leads to inefficient way to transport data.

Clients and servers that communicate via a reliable channel, such as a TCP socket, have a dedicated point-to-point channel between themselves. To communicate, they establish a connection, transmit the data, and then close the connection. All data sent over the channel is received in the same order in which it was sent. This is guaranteed by the channel.

In contrast, applications that communicate via datagrams send and receive completely independent packets of information. These clients and servers do not have and do not need a dedicated point-to-point channel. The delivery of datagrams to their destinations is not guaranteed. Nor is the order of their arrival.

A datagram is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed.

Java implements datagrams on top of the UDP (User Datagram Protocol) protocol by using two classes:

- **DatagramPacket class**: DataGramPacket class provides the facility for connectionless transfer of message from one system to other. Each message is routed only on the basis of information contained within the packet and it may be possible for different packets to route differently. There is no guarantee for the successful delivery of message. Besides that packets may arrive out of order. This DataGramPacket clas provides the facility to datagram packets for connectionless delivery through DataGramSocket class.

There are two constructors available for this class-

public DatagramPacket(byte[] buf, int offset, int length)
public DatagramPacket(byte[] buf , int length)

- **DataGramSocket class**: DatagramSocket class in java provides functionality for sending and receiving the packet. Every packet sent from a datagram socket is individually routed and delivered. It follow UDP Protocal. A datagram is basically an information but there is no guarantee of its content, arrival or arrival time.

DatagramSocket( ) throws SocketEeption
DatagramSocket(int port) throws SocketEeption
DatagramSocket(int port , InetAddress address) throws SocketEeption

Methods of DataGramPacket class	
Method	**Purpose**
public InetAddress getAddress()	Returns the IP address to which this packet is sent to or from which it was received
public int getPort()	Returns the port to which this packet is sent to or from which it was received. This method is specifically used on the server from getting the port of the client who sent the request.
public byte[] getData()	Returns the data contained in this packet as a byte array. The data starts from the offset specified and is of length specified.
public int getOffset()	Returns the offset specified.
public int getLength()	Returns the length of the data to send or receive
public void setData (byte[] buf, int offset, int length)	Used to set the data of this packet.

Table 14.7

Methods of DataGramSocket class	
Method	**Purpose**

public void send (DatagramPacket p)	Sends a datagram packet from this socket.
public void receive (DatagramPacket p)	It is used to receive the packet from a sender.
public void bind (SocketAddress addr)	Binds this socket to specified address and port number.

Table 14.8

Example: Unit-14 (Program-06)

/* Write a program in java to receive the DataGram packet via UDP protocol.*/

```
import java.net.*;

public class unit14p06
{
    public static void main(String[] args)
    {
        try {
            // create a empty packet to receive the data
            byte b[] = new byte[10000];
            DatagramPacket p = new DatagramPacket(b, b.length);

            DatagramSocket socket = new DatagramSocket(8888);
            System.out.println("waiting for the packet.....");

            //receiving packet
            socket.receive(p);
            System.out.println(new String(b));
        }
        catch (Exception e)
        {
```

```
                    // TODO: handle exception
                    e.printStackTrace();
                }
            }
        }
```

Result:

waiting for the packet.....

The above program will not terminate after execution. It will keep on waiting to receive the datagram packet from any other source.

Example: Unit-14 (Program-07)

/* Write a program in java to send the DataGram packet via UDP protocol.*/

```
import java.net.*;
import java.io.*;
public class unit14p07
{
    public static void main(String[] args)
    {
        try {
            BufferedReader br = new BufferedReader (new
                                InputStreamReader(System.in));
            System.out.println("Enter the content to be send: ");
            String content = br.readLine();

            // Create the packet to be send using DatagramPacket
            DatagramPacket packet= new
                                DatagramPacket(content.getBytes(),
                                content.getBytes().length,
            InetAddress.getLocalHost(), 8888);

            // now create socket using DatagramSocket to send the
data..
```

```
                    DatagramSocket socket = new DatagramSocket();
                    // now send the packet using socket
                    socket.send(packet);
                    System.out.println("packet is sent successfully...");
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
        }
    }
```

Result:

Enter the content to be send:

Ankur Mittal

packet is sent successfully...

Once this program is executed, the socket which was waiting for the data to be received, will also receive the data packet, processing it and gets completed.

We must make sure that, Earlier program is already running and waiting for the data to receive.

TCP vs UDP in java :

- TCP is reliable and guaranteed but UDP is not guaranteed. You are at risk of losing messages in UDP.
- TCP provides ordering guaranteed which means the message will be received in the same order they are sent, UDP protocol doesn't provide such guarantee.
- TCP is a connection-oriented protocol as compared to UDP which is connectionless.
- UDP is much faster than TCP protocol.
- UDP protocol can be used for multicasting e.g. TIBCO RV
- UDP also preserves data boundary because all data is sent in the same packet, but TCP doesn't as one message may be divided into several packages before sending.

Check Your Progress

1. What is datagram in java?
2. Compare TCP and UDP protocols in java.
3. Write a program in java to send the data from one device to other using UDP protocol.
4. Write a program in java to receive the data with UDP protocol in java

14.9 SUMMARY

Network programming is one of the important part in java programming. It provides java.net package to do the network programming. It supports both TCP and UDP protocol for doing the protocol. Java also provides the facility to manage the URL via programming. DNS services may be resolved using the library of java too.

14.10 EXERCISE

Q. 1. What is socket programming in java?

Q. 2. Explain the types of sockets in java.

Q. 3. Describe the proxy management in java.

Q. 4. Write a program in java to download the web page with the URL of the web page.

Q. 5. Write a program in java to accept the URL from user and display its information.

Q. 6. Write a program in java to create the server socket which accepts the faculty information from different clients.

Q. 7. What are the reserve ports in network programming in java? List the possible ports for programming.

Q. 8. Write a program in java to exchange the record of the college from one source to destination machine via UDP protocol.

Q. 9. Write a program in java to share the information of the employee using TCP protocol.

Q. 10. Write a protocol in java to extract the properties of a web page as input from user

ROUGH WORK