



CLASS:BCA5thSem
Batch: 2018-19

JAVA

Notes as per IKGPTU Syllabus

Name of Faculty: Ms<Jatinderpal Kaur>
Faculty of IT Department, SBS College. Ludhiana

SECTION-A

FUNDAMENTALS OF OBJECT-ORIENTED PROGRAMMING: -Introduction; Object-Oriented Paradigm; Basic Concepts of Object-Oriented Programming; Benefits of OOP; Applications of OOP.

JAVA EVOLUTION: -Java History; Java Features; How Java Differs from C and C++; Java and Internet, Java and World Wide Web, Web Browsers; Hardware and Software Requirements; Java Support Systems, Java Environment

OVERVIEW OF JAVA LANGUAGE:-Introduction;SimpleJavaProgram;Commentsinjava; An application with Two Classes; Java Program Structure; Java Tokens; Java Statements; ImplementingaJavaProgram;JavaVirtualMachine; Command LineArguments;ProgrammingStyle.

CONSTANTS,VARIABLESANDDATATYPES:-Introduction;Constants; Variables;Data Types; Variables, Constants, Standard DefaultValues.

OPERATORSANDEXPRESSIONS:-Introduction toOperators,Expressions; OperatorPrecedence; MathematicalFunctions.

DECISION MAKING, BRANCHINGANDLOOPING: - Decision making and Branching Statements, LoopingStatements,Labeledloops,JumpingStatements

SECTION-B

CLASSES,OBJECTSANDMETHODS:-Introduction;DefiningaClass;Adding Variables; Adding Variables; Adding Methods; Creating Objects; Accessing ClassMembers;Constructors;MethodsOverloading;StaticMembers;NestingofMethods;

Inheritance: Extending a Class; Overriding Methods; Final Variables and Methods; Final Classes; FinalizerMethods;AbstractMethodsandClasses;VisibilityControl.

ARRAYS,STRINGSANDVECTORS:- Arrays;ZaggedArrays;;Strings; String functions;Vectors; WrapperClasses.

INTERFACES: Introduction;Defining Interfaces;ExtendingInterfaces;Implementing Interfaces; AccessingInterfaceVariables, ImplementingMultipleInheritanceusingInterfaces.

PACKAGES: Introduction;SystemPackages; Using System Packages; Naming Conventions; CreatingPackages;AccessingaPackage;UsingaPackage;AddingaClasstoPackage;HidingClasses.

SECTION-C

MANAGING ERRORS AND EXCEPTIONS:- Introduction; Types of Errors; Exceptions; Exception Handling using Try, Catch and Finally block; Throwing Our Own Exceptions; Using Exceptions for Debugging.

APPLET PROGRAMMING:- Introduction; How Applets Differ from Applications; Applet Life Cycle; Creating an Executable Applet; Passing Parameters to Applets; Aligning the Display; More about HTML Tags; Displaying Numerical Values; Getting Input from the User.

GRAPHICS PROGRAMMING:- Introduction; The Graphics Class; Lines and Rectangles; Circles and Ellipses; Drawing Arcs; Drawing Polygons; Line Graphs; Using Control Loops in Applets; Drawing Bar Charts.

SECTION-D

JAVA AWT: -Java AWT package Containers; Basic User Interface components; Layouts.

EVENT HANDLING: -Event delegation Approach; ActionListener; AdjustmentListener, MouseListener; MouseMotionListener; WindowListener; KeyListener; ItemListener

JAVA I/O HANDLING : I/O File Handling (InputStream & OutputStreams, FileInputStream & FileOutputStream, Data I/P and O/P Streams, File Class, Reader and Writer Streams, Random Access File).

SECTION-A

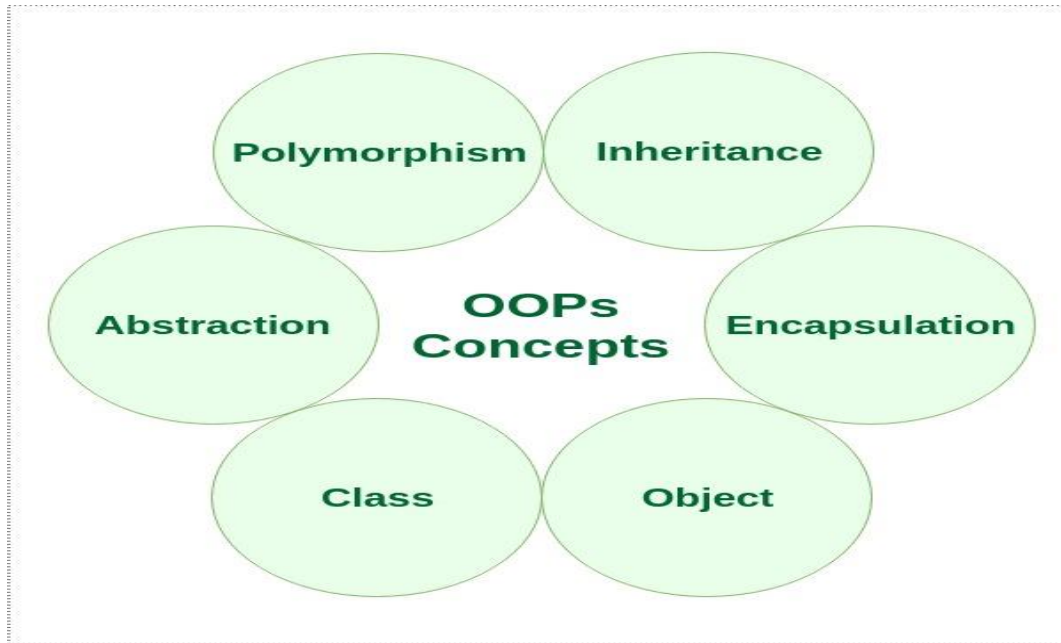
Introduction of Java

Java is a simple and yet powerful object oriented programming language and it is in many respects similar to C++. Java originated at Sun Microsystems, Inc. in 1991. It was conceived by James Gosling, It was developed to provide a platform-independent programming language. It is Pure Object Oriented Language Because Every Program must be written into Classes. Object Oriented Programming (OOPs) Concept in Java

Object-oriented programming Paradigm & Concepts: As the name suggests, Object-Oriented Programming or OOPs refers to languages that uses objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

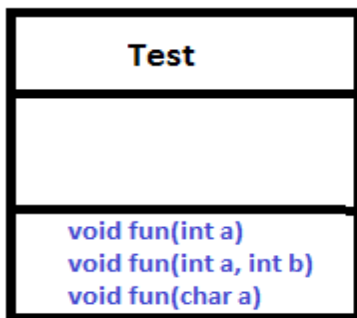
OOPs Concepts:

- Polymorphism
- Inheritance
- Encapsulation
- Abstraction
- Class
- Object
- Method
- Message Passing

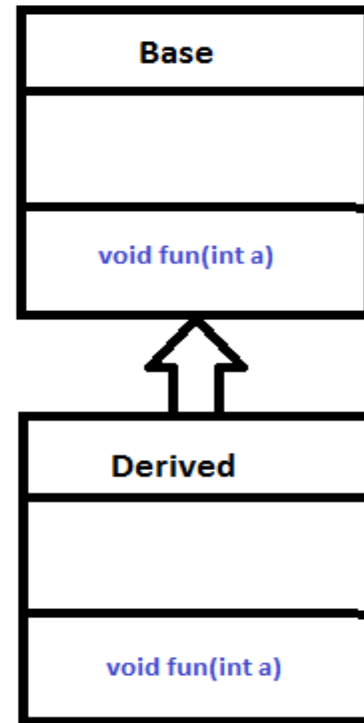


1. Polymorphism in Java

- The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.
- **Real life example of polymorphism:** A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So the same person possesses different behavior in different situations. This is called polymorphism.
- Polymorphism is considered as one of the important features of Object Oriented Programming. Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations. The word “poly” means many and “morphs” means forms, So it means many forms.
- **In Java polymorphism is mainly divided into two types:**
 - Compile time Polymorphism
 - Runtime Polymorphism
- **Compile time polymorphism:** It is also known as static polymorphism. This type of polymorphism is achieved by function overloading or operator overloading.
- **Runtime polymorphism:** It is also known as Dynamic Method Dispatch. It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding.



Overloading



Overriding

2. Inheritance in Java

Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in java by which one class is allow to inherit the features(fields and methods) of another class.

Important terminology:

Super Class: The class whose features are inherited is known as super class(or a base class or a parent class).

Sub Class: The class that inherits the other class is known as sub class(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.

Reusability: Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes

some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

3. Encapsulation in Java

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. Other way to think about encapsulation is, it is a protective shield that prevents the data from being accessed by the code outside this shield.

- Technically in encapsulation, the variables or data of a class is hidden from any other class and can be accessed only through any member function of own class in which they are declared.
- As in encapsulation, the data in a class is hidden from other classes, so it is also known as **data-hiding**.
- Encapsulation can be achieved by: Declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.



4. Abstraction in Java

Data Abstraction is the property by virtue of which only the essential details are displayed to the user. The trivial or the non-essentials units are not displayed to the user. Ex: A car is viewed as a car rather than its individual components.

Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details.

The properties and behaviors of an object differentiate it from other objects of similar type and also help in classifying/grouping the objects.

Classes and Objects in Java

Classes and Objects are basic concepts of Object Oriented Programming which revolve around the real life entities.

5. Class in java

A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:

- **Modifiers** : A class can be public or has default access (Refer this for details).
- **Class name**: The name should begin with a initial letter (capitalized by convention).
- **Superclass(if any)**: The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
- **Interfaces(if any)**: A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
- **Body**: The class body surrounded by braces, { }.

6. Object in java

It is a basic unit of Object Oriented Programming and represents the real life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of :

- **State** : It is represented by attributes of an object. It also reflects the properties of an object.
- **Behavior** : It is represented by methods of an object. It also reflects the response of an object with other objects.
- **Identity** : It gives a unique name to an object and enables one object to interact with other objects.

7. Methods in Java

A method is a collection of statements that perform some specific

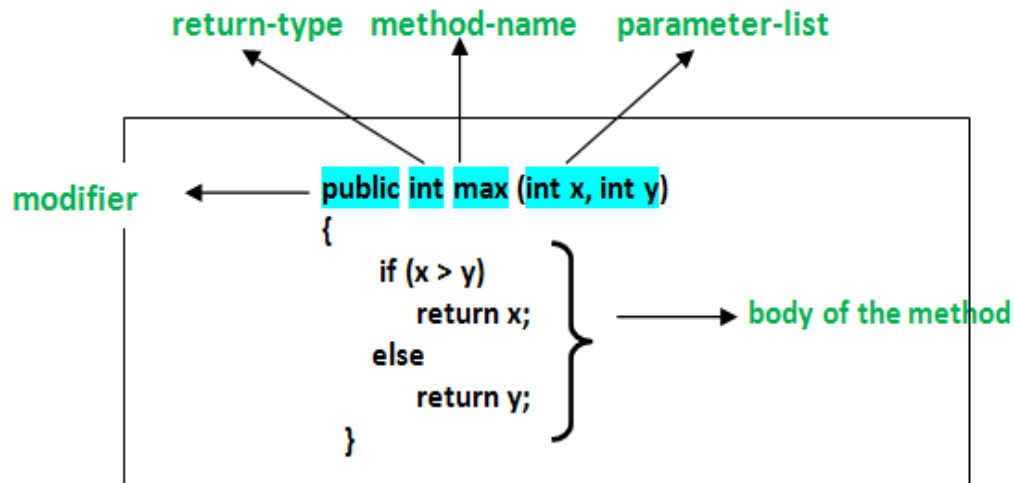
task and return the result to the caller. A method can perform some specific task without returning anything. Methods allow us to **reuse** the code without retyping the code. In Java, every method must be part of some class which is different from languages like C, C++, and Python.

Methods are **time savers** and help us to **reuse** the code without retyping the code.

Method Declaration

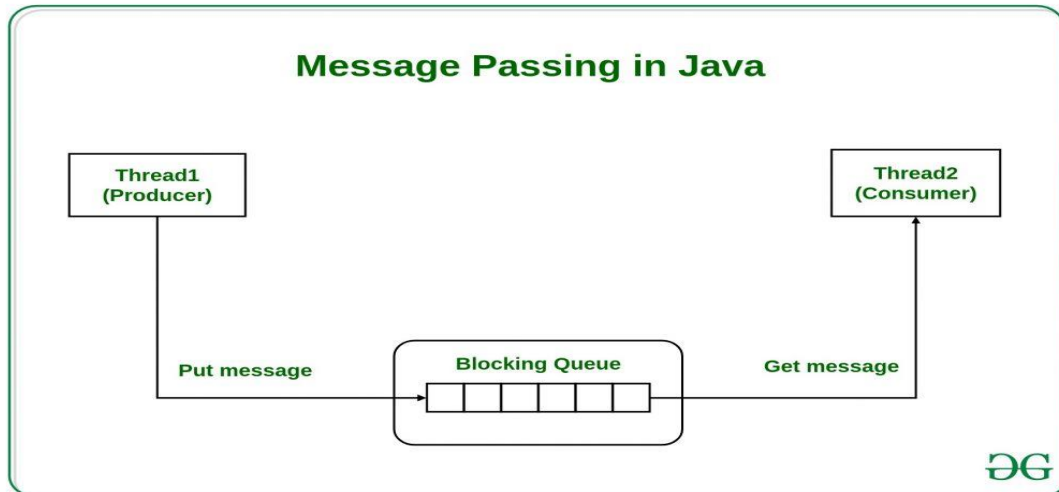
In general, method declarations has six components :

- **Modifier-**: Defines **access type** of the method i.e. from where it can be accessed in your application. In Java, there 4 type of the access specifiers.
 - **public**: accessible in all class in your application.
 - **protected**: accessible within the class in which it is defined and in its **subclass(es)**
 - **private**: accessible only within the class in which it is defined.
 - **default** (declared/defined without using any modifier) : accessible within same class and package within which its class is defined.
- **The return type** : The data type of the value returned by the method or void if does not return a value.
- **Method Name** : the rules for field names apply to method names as well, but the convention is a little different.
- **Parameter list** : Comma separated list of the input parameters are defined, preceded with their data type, within the enclosed parenthesis. If there are no parameters, you must use empty parentheses ().
- **Method body** : it is enclosed between braces. The code you need to be executed to perform your intended operations.



8. Message Passing in Java

Message Passing in terms of computers is communication between processes. It is a form of communication used in object-oriented programming as well as parallel programming. Message passing in Java is like sending an object i.e. message from one thread to another thread. It is used when threads do not have shared memory and are unable to share monitors or semaphores or any other shared variables to communicate. Suppose we consider an example of producer and consumer, likewise what produce will produce, the consumer will be able to consume that only. We mostly use **Queue** to implement communication between threads.



Applications of Object Oriented Programming

Main application areas of OOP are:

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

Benefits of OOP:

- It is easy to model a real system as real objects are represented by programming objects in OOP. The objects are processed by their member data and functions. It is easy to analyze the user requirements.
- With the help of inheritance, we can reuse the existing class to derive a new class such that the redundant code is eliminated and the use of existing class is extended. This saves time and cost of program.
- In OOP, data can be made private to a class such that only member functions of the class can access the data. This principle of data hiding helps the programmer to build a secure program that can not be invaded by code in other part of the program.

- With the help of polymorphism, the same function or same operator can be used for different purposes. This helps to manage software complexity easily.
- Large problems can be reduced to smaller and more manageable problems. It is easy to partition the work in a project based on objects.
- It is possible to have multiple instances of an object to co-exist without any interference i.e. each object has its own separate member data and function.

Java History

The history of Java is very interesting. Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time. The history of java starts with Green Team. Java team members (also known as **Green Team**), initiated this project to develop a language for digital devices such as set-top boxes, televisions, etc. However, it was suited for internet programming. Later, Java technology was incorporated by Netscape.

The principles for creating Java programming were "Simple, Robust, Portable, Platform-independent, Secured, High Performance, Multithreaded, Architecture Neutral, Object-Oriented, Interpreted and Dynamic". Currently, Java is used in internet programming, mobile devices, games, e-business solutions, etc. There are given the significant points that describe the history of Java.

1) **James Gosling, Mike Sheridan, and Patrick Naught** on initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.

- 2) Originally designed for small, embedded systems in electronic appliances like set-top boxes.
 - 3) Firstly, it was called "**Greentalk**" by James Gosling, and file extension was .gt.
 - 4) After that, it was called **Oak** and was developed as a part of the Green project.
- Why Java named "Oak"?
- 5) **Why Oak?** Oak is a symbol of strength and chosen as a national tree of many countries like U.S.A., France, Germany, Romania, etc.
 - 6) In 1995, Oak was renamed as "**Java**" because it was already a trademark by Oak Technologies.
 - 7) Initially developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.
 - 8) In 1995, Time magazine called **Java one of the Ten Best Products of 1995**.
 - 9) JDK 1.0 released in (January 23, 1996).

Java Features

1. Object

In Java, everything is an Object. Java can be easily extended since it is based on the Object model.

2. Platform Independent

Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform-independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.

3. Simple

Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.

4. Secure

With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.

5. Architecture-neutral

Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.

6. Portable

Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. The compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.

7. Robust

Java makes an effort to eliminate error-prone situations by emphasizing mainly on compile time error checking and runtime checking.

8. Multithreaded

With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.

9. Interpreted

Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.

10. High Performance

With the use of Just-In-Time compilers, Java enables high performance.

11. DistributedJava is designed for the distributed environment of the internet.

12. Dynamic

Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java

programs can carry an extensive amount of run-time information that can be used to verify and resolve accesses to objects at run-time.

Difference Between C ,C++,Java

C Programming	Java Programming
It does include the unique statement keywords sizeof and typedef.	It does not include the C unique statement keywords sizeof, and typedef.
It contains the data type struct and union.	It does not contain the data type struct and union.
It defines the type modifiers keywords auto, extern, register, signed, and unsigned.	It does not define the type modifiers keywords auto, extern, register, signed, and unsigned.
It supports an explicit pointer type.	It does not support an explicit pointer type.
It has a preprocessor and therefore we can use # define, # include, and # ifdef statements.	It does not have a preprocessor and therefore we cannot use # define, # include, and # ifdef statements.
It requires that the functions with no arguments, with the void keyword	It requires that the functions with no arguments must be declared with empty parenthesis, not with the void keyword
C has no operators such as instanceof and >>>.	Java adds new operators such as instanceof and >>>.

	C++	JAVA
1.	C++ was developed by Bjarne Stroustrup . Development began in 1979.	Java was developed by James Gosling and his team. Development began in 1991.
2.	C++ is a compiled language .	Java is both compiled and interpreted .
3.	C++ supports conditional compilation and inclusion.	Java does not support conditional compilation.
4.	C++ programs are platform dependent . They need to be compiled for a particular platform.	Java programs are platform independent. Java programs are written for Java Virtual Machine (JVM) and wherever a JVM is installed, Java program will run without needing recompilation.
5.	C++ does support operator overloading . Function overloading is also available.	Java does not support operator overloading. However, function overloading is possible.
6.	C++ fully support pointers .	Java has restricted support for pointers. Pointers are supported internally you can not write pointer programs.
7.	C++ supports structures .	Java does not support structures.
8.	C++ supports unions .	Java does not support unions.
9.	C++ does not have built-in support for threads .	Java fully supports threads.
10.	C++ supports manual object management through <i>new</i> and <i>delete</i> keywords.	Java relies on automatic garbage collection . It does not support destructors the way C++ does.

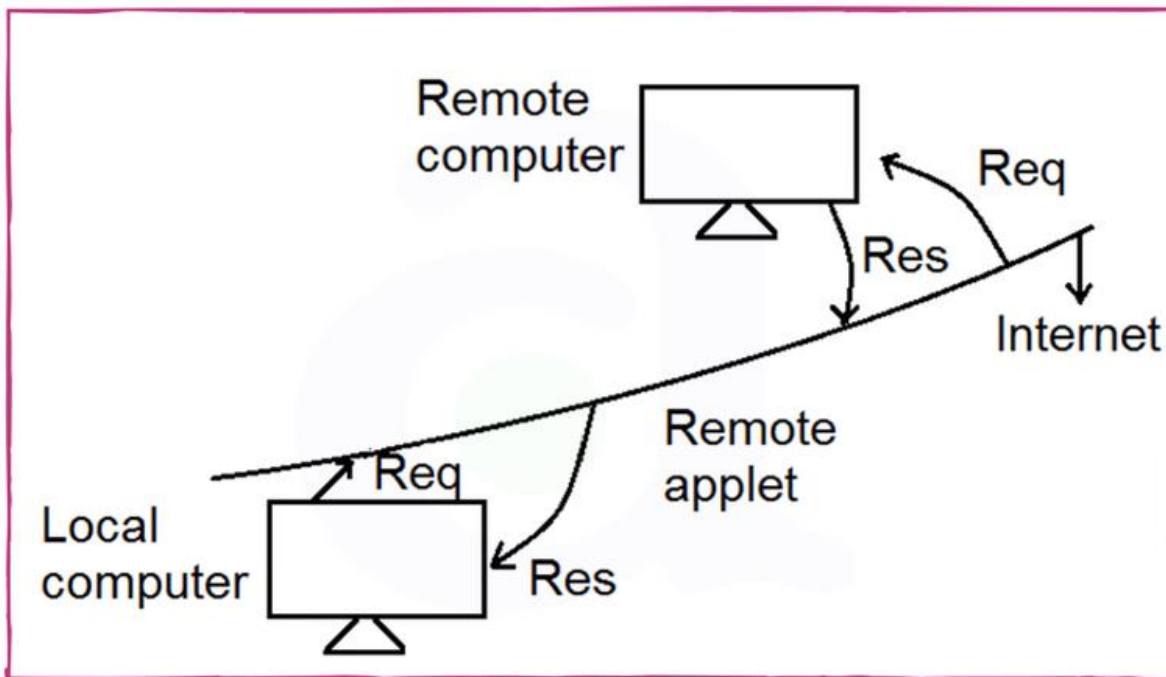
Java and Internet

Java is strongly associated with the internet because of the first application program is written in Java was hot Java.

Web browsers to run applets on the internet.

Internet users can use Java to create applet programs & run then locally using a Java-enabled browser such as hot Java.

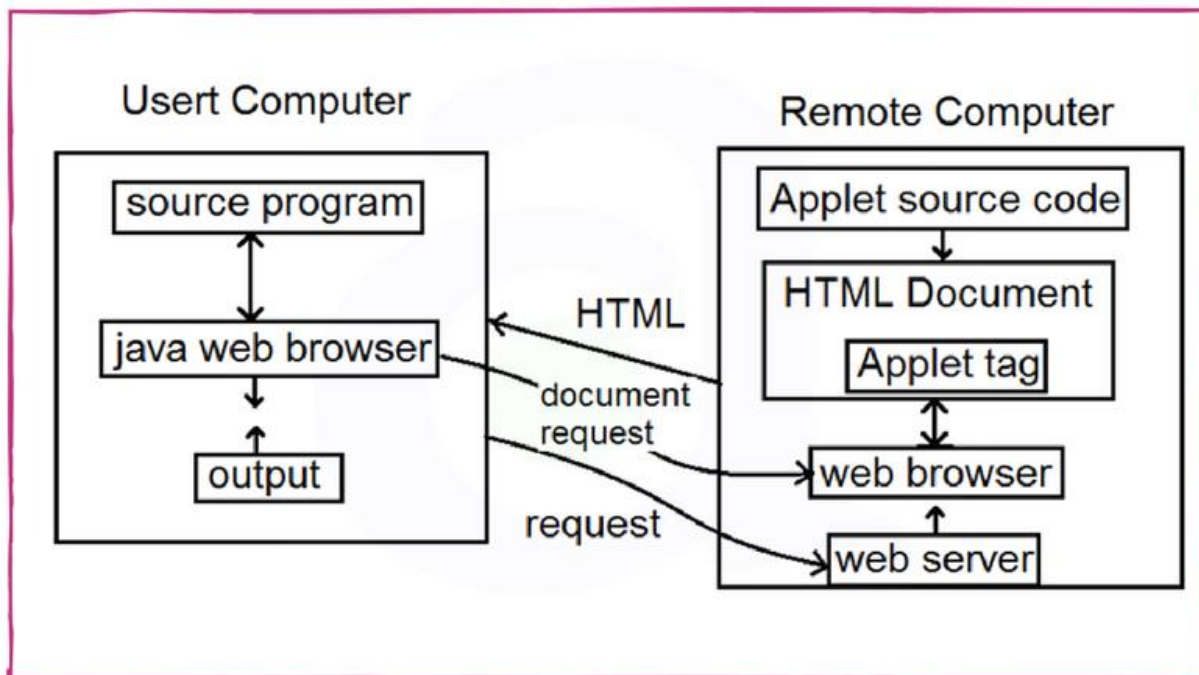
Java applets have made the internet a true extension of the storage system of the local computer.



Java and Internet

World wide web and internet

1. World wide web is a collection of information stored on internet computers.
2. World wide web is an information retrieval system designed to be used in the internet's distributed environment.
3. World wide web contains web pages that provide both information and controls.
4. Web pages contain HTML tags that enable us to find retrieve, manipulate and display documents world wide.
5. Before Java, the world wide web was limited to the display of still images & texts.
6. With the help of Java WWW is capable of supporting animation graphics, games and wide rage special effects.



Java and World Wide Web (www)

Web Browsers

The internet is a vast sea of information represented in many formats and stored on many computers. a browser is a software application used to locate, retrieve and display content on the World Wide Web, including Web pages, images, video and other files. As a client/server model, the browser is the client run on a computer that contacts the Web server and requests information. The Web server sends the information back to the Web browser which displays the results.

The browser application retrieves or fetches code, usually written in HTML (HyperText Markup Language) and/or another language, from a web server, interprets this code, and renders (displays) it as a Web page for you to view. on the computer or another Internet-enabled device that supports a browser.

An example of Web Browsers:

- Hot Java
- Netscape Navigator
- Internet Explorer
- Google Chrome

Java Support System

Web Browser

local computer should be connected to the internet

Web Server

A program that accepts a request from a user and gives output as per the requirement. Apache TomCat server is one of the major web servers.

Web Browser

The web browser is a software that will allow you to view web pages on the internet. it is a program that you use to access the Internet. It reads and knows how to display and download files that are put on servers for people to read. A program that provides the access of WWW and runs java applets. Chrome and Firefox are two major web browsers.

HTML

HTML is short for HyperText Markup Language. HTML is used to

create electronic documents (called pages) that are displayed on the World Wide Web. Each page contains a series of connections to other pages called hyperlinks. Every web page you see on the Internet is written using one version of HTML code or another.

Byte Code

Compiled java code that is referred to in the applet tag and transfers to the user computer.

Proxy Server

An intermediate server between the requesting client work station and the original server. It is typically implemented for ensuring security.

Mail Server

A mail server (also known as a mail transfer agent or MTA, a mail transport agent, a mail router or an Internet mailer) is an application that receives an incoming e-mail from local users (people within the same domain) and remote senders and forwards outgoing .

Hardware Requirement for Java

Minimum hardware requirement to download Java on your Windows operating system as follows:

- Minimum Windows 95 software
- IBM-compatible 486 system
- Hard Drive and Minimum of 8 MB memory
- A CD-ROM drive
- Mouse, keyboard and sound card, if required

Software requirement for Java

Nowadays, Java is supported by almost every operating systems. whether it is a Windows, Macintosh and Unix all supports the Java application development. So you can download any of the operating system on your personal computer. Here are the minimum requirement.

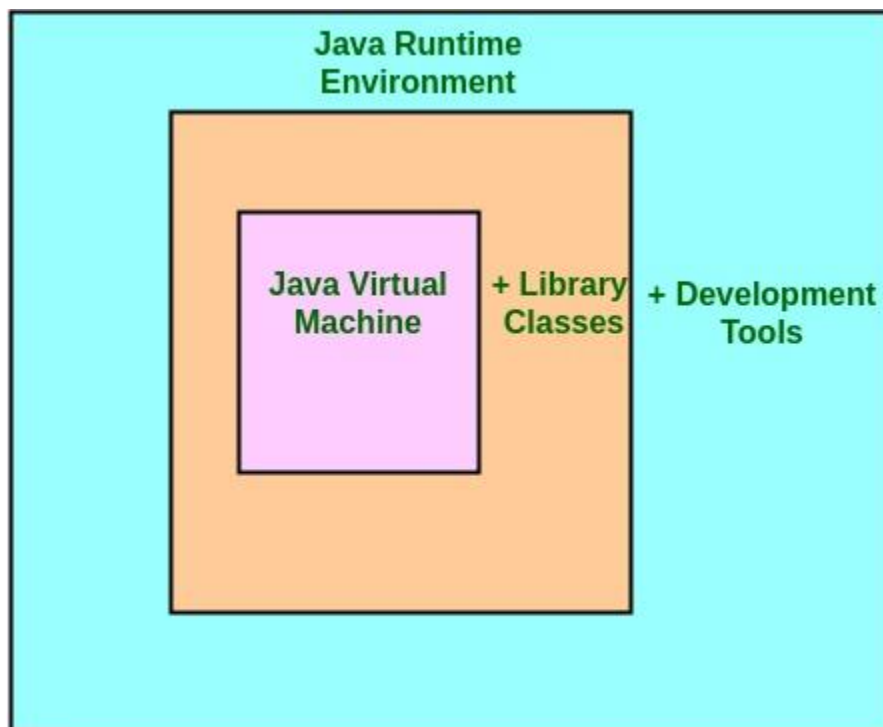
- Operating System
- Java SDK or JRE 1.6 or higher
- Java Servlet Container (Free Servlet Container available)
- Supported Database and library that supports the database connection with Java.

Setting up the environment in Java

- Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented etc.

Java applications are typically compiled to **bytecode** that can run on any Java virtual machine (JVM) regardless of computer architecture. The latest version is **Java 11**.

- Below are the environment settings for both Linux and Windows. JVM, JRE and JDK all three are platform dependent because configuration of each Operating System is different. But, Java is platform independent.
- There are few things which must be clear before setting up the environment
- **JDK**(Java Development Kit) : JDK is intended for software developers and includes development tools such as the Java compiler, Javadoc, Jar, and a debugger.
- **JRE**(Java Runtime Environment) : JRE contains the parts of the Java libraries required to run Java programs and is intended for end users. JRE can be view as a subset of JDK.
- **JVM**: JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed. JVMs are available for many hardware and software platforms.



JDK = JRE + Development Tool
JRE = JVM + Library Classes

Structure of Java program

Introduction and Simple Program in Java

- **class** keyword is used to declare a class in java.
- **public** keyword is an access modifier which represents visibility. It means it is visible to all.
- **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create an object to invoke the main method. So it saves memory.
- **void** is the return type of the method. It means it doesn't return any value.
- **main()** represents the starting point of the program.
- **String[] args** is used for command line argument. We will learn it later.
- **System.out.println()** is used to print statement. Here, System is a class, out is the object of PrintStream class, println() is the method of PrintStream class. We will learn about the internal working of System.out.println statement later.

Example of print message “Hello” in Java

```
class First
```

```
{  
    public static void main(String args[ ] )  
    {
```

```
        System.out.println(“hello guys”);
```

```
    }  
}
```

- Save it by name using class name i.e First.java
- To compile Program
- Go to dos window using key window+r=run
- Write Command javac First.java

- Then to Run program Write Command java First
- To run program java First

Output

Hello guys

Using multiple classes in a Java program

A Java program can contain any number of classes. Following Java program comprises of two classes: Computer and Laptop. Both classes have their own constructors and a method. In the main method, we create objects of two classes and call their methods.

Using two classes in Java program

```
class Computer
```

```
{
```

```
    void computer_method()
```

```
{
```

```
    System.out.println("Power gone! Shut down your PC soon...");
```

```
}
```

```
class Laptop
```

```
{
```

```
    void laptop_method()
```

```
{
```

```
    System.out.println("99% Battery available.");
```

```
}
```

```
}
```

```
public static void main(String[] args)
```

```
{  
  Computer my = new Computer();  
  Laptop your = new Laptop();  
  
  my.computer_method();  
  your.laptop_method();  
}
```

Structure of Java Program

A Java program involves the following sections:

- Documentation Section
- Package Statement
- Import Statements
- Interface Statement
- Class Definition
- Main Method Class
 - Main Method Definition

Documentation Section
Package Statement
Import Statement
Interface Statement
Class Definition
<pre> Main Method Class { //Main method defintion } </pre>

Detail

Section	Description
Documentation Section	You can write a comment in this section. Comments are beneficial for the programmer because they help them understand the code. These are optional, but we suggest you use them because they are useful to understand the

	operation of the program, so you must write comments within the program.
Package statement	<p>You can create a package with any name. A package is a group of classes that are defined by a name. That is, if you want to declare many classes within one element, then you can declare it within a package. It is an optional part of the program, i.e., if you do not want to declare any package, then there will be no problem with it, and you will not get any errors. Here, the package is a keyword that tells the compiler that package has been created.</p> <p>It is declared as:</p> <hr/> <pre>package package_name;</pre> <hr/>
Import statements	<p>This line indicates that if you want to use a class of another package, then you can do this by importing it directly into your program.</p> <p><u>Example:</u></p> <hr/> <pre>import calc.add;</pre> <hr/>
Interface statement	Interfaces are like a class that includes a group of method declarations. It's an optional section and can be used when programmers want to implement multiple inheritances within a program.
Class Definition	A Java program may contain several class definitions. Classes are the main and essential elements of any Java program.
Main Method Class	Every Java stand-alone program requires the main method as the starting point of the program. This is an essential part of a Java program. There may be many classes in a Java program, and only one class defines the main method . Methods contain data type declaration and executable statements.

Example

package details	→	import java.io.*
class className	→	class Sum
{		
Data members;	→	int a, b, c;
user_defined method;	→	void display();
public static void main(String args[])		
{		
Block of Statements;	→	System.out.println("Hello Java !");
}		
}		

Tutorial4us.com

Java Tokens

Java Tokens:- A [java](#) Program is made up of Classes and Methods and in the Methods are the Container of the various Statements And a Statement is made up of Variables, Constants, operators etc .

Tokens are the smallest unit of Program There is Five Types of Tokens

- **Reserve Word or Keywords:** Keywords are the pre-defined identifiers reserved by Java for a specific purpose and used only in a limited, specific manner.
- **Identifier:** A Java identifier is the symbolic name that a programmer gives to various programming elements such as a variables method, class, array, etc.

- **Literals:** A literal is a constant value that can be classified as integer literals, string literals, and boolean literals.
- **Operators:** An operator is a special symbol that tells the compiler to perform a specific mathematical or logical operation on one or more operands where an operand can be an expression.
- **Separators:** Separators are the lines that are used to virtual group related items together.

Java Statements

Statements are similar to sentences in the English language. A sentence forms a complete idea which can include one or more clauses. Likewise, a *statement* in Java forms a complete command to be executed and can include one or more expressions.

In simpler terms, a Java statement is just an instruction that explains what should happen.

Types of Java Statements

Java supports three different types of statements:

- **Expression statements** change values of variables, call methods, and create objects.
- **Declaration statements** declare variables.
- **Control-flow statements** determine the order that statements are executed. Typically, Java statements parse from the top to the bottom of the program. However, with control-flow statements, that order can be interrupted to implement branching or looping so that the Java program can run particular sections of code based on certain conditions.

JVM

JVM is a engine that provides runtime environment to drive the Java Code or applications. It converts Java bytecode into machines language. JVM is a part of **JRE(Java Run Environment)**. It stands for Java Virtual Machine

- In other programming languages, the compiler produces machine code for a particular system. However, Java compiler produces code for a Virtual Machine known as Java Virtual Machine.
- First, Java code is compiled into bytecode. This bytecode gets interpreted on different machines
- Between host system and Java source, Bytecode is an intermediary language.
- JVM is responsible for allocating memory space.



Command line argument in Java

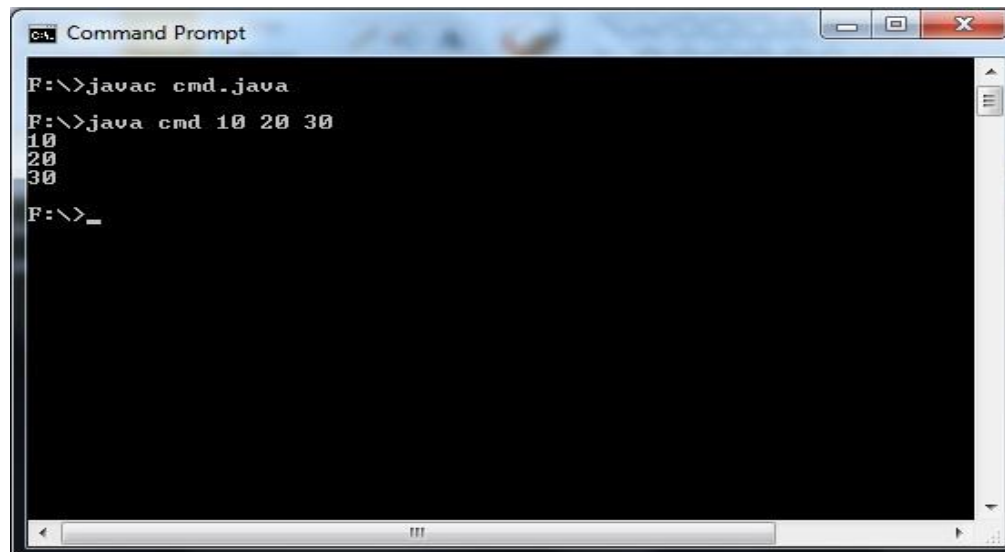
The command line argument is the argument passed to a program at the time when you run it. To access the command-line argument inside a java program is quite easy, they are stored as string in **String** array passed to the args parameter of main() method.

Example

```
class cmd
{
    public static void main(String[] args)
    {
        for(int i=0;i<args.length;i++)
        {
            System.out.println(args[i]);
        }
    }
}
```

Execute this program as java cmd 10 20 30

10
20
30



```
cmd Command Prompt
F:\>javac cmd.java
F:\>java cmd 10 20 30
10
20
30
F:\>_
```

Constants

Constants in java are fixed values those are not changed during the Execution of program java supports several types of Constants those are:

Integer Constants

Integer Constants refers to a Sequence of digits which Includes only negative or positive Values and many other things those are as follows:-

1. An Integer Constant must have at Least one Digit.
2. it must not have a Decimal value.
3. it could be either positive or Negative.
4. if no sign is Specified then it should be treated as Positive.
5. No Spaces and Commas are allowed in Name.

Real Constants

1. A Real Constant must have at Least one Digit.
2. it must have a Decimal value.
3. it could be either positive or Negative.
4. if no sign is Specified then it should be treated as Positive.

5. No Spaces and Commas are allowed in Name.
6. Like 251, 234.890 etc are Real Constants.

In The Exponential Form of Representation the Real Constant is Represented in the two Parts The part before appearing e is called mantissa whereas the part following e is called Exponent.

7. In Real Constant The Mantissa and Exponent Part should be Separated by letter e.
8. The Mantissa Part have may have either positive or Negative Sign.
9. Default Sign is Positive.

Single Character Constants

A Character is Single Alphabet a single digit or a Single Symbol that is enclosed within Single inverted commas.

Like 'S' , '1' etc are Single Character Constants.

String Constants

String is a Sequence of Characters Enclosed between double Quotes These Characters may be digits ,Alphabets Like "Hello" , "1234" etc.

Backslash Character Constants

Java Also Supports Backslash Constants those are used in output methods For Example \n is used for new line Character These are also Called as escape Sequence or backslash character Constants For Ex:

\t For Tab (Five Spaces in one Time)
\b Back Space etc.

Variables

A variable is used for storing a value either a number or a character and a variable also vary its value means it may change his value Variables are used for given names to locations in the Memory of Computer where the different constants are stored. These locations contain Integer ,Real or Character Constants.

Data Types

Every variable has a data type which denotes the type of data which a variable will hold. There are many built-in data types in Java. Those are called as Primitives data types or built-in data types and there are also some data types those are defined by user-defined types which are also called as Non-Primitives data types.

Data types are means to identify the type of data and associated operations for handling it. Java data types are of two types:

1. Primitive (Intrinsic)
2. Non-Primitive (Derived)

Integer types

These types can hold whole numbers such as 123, -90 etc. The size of the values depends on the Integer data type or range of the Integer data type that Java supports. But always remember Java doesn't support signed, unsigned data types. But the range of the Integer data type is increased from 2 bytes to four bytes.

Integer Types, Size and Range of Values.

Reserved Word	Data Types	Size	Range of Values
byte	Byte length Integer	1 byte	-2^8 to $2^7 - 1$
short	Short Integer	2 bytes	-2^{16} to $2^{15} - 1$
int	Integer	4 bytes	-2^{32} to $2^{31} - 1$
long	Long Integers	8 bytes	-2^{64} to $2^{63} - 1$

Floating data types

The Floating data types contain whole numbers and also decimal values. Those are also called as the 3 Real Constants or Fractional Numbers. These support special types of values known as NAN or not a number which is used for representing the result when we divide a number by zero or when the actual result is not produced.

Floating-point Types, Size and Range of Values.

Reserved Word	Data Types	Size	Range of Values
float	Single Precision	4 byte	-2^{32} to $2^{31} - 1$
double	Real number with double precision	8 bytes	-2^{34} to $2^{63} - 1$

Character type

Java provides a character data type called Char For Storing a Character value and in java char has 2 bytes for holding a Single Character.

Boolean data type

This is Also Special data type used when the Execution is depend on Some Conditions Either they are true or False Boolean is a Special data type Which Returns Result only in true or False.

Default Value of Data Types in Java :

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null

boolean	false
---------	-------

Live Example : Default value of Data Type

Sample Program that will illustrate Default Value Stored in Each Primitive Data Type Variable

```
public class DefaultValue {  
    static boolean bool;  
    static byte by;  
    static char ch;  
    static double d;  
    static float f;  
    static int i;  
    static long l;  
    static short sh;  
    static String str;  
  
    public static void main(String[] args) {  
        System.out.println("Bool :" + bool);  
        System.out.println("Byte :" + by);  
        System.out.println("Character:" + ch);  
        System.out.println("Double :" + d);  
        System.out.println("Float :" + f);  
        System.out.println("Integer :" + i);  
        System.out.println("Long :" + l);  
        System.out.println("Short :" + sh);  
        System.out.println("String :" + str);  
    }  
}
```

Output :

```
[468x60]  
Bool   :false  
Byte   :0  
Character:  
Double :0.0  
Float  :0.0
```

Integer :0
Long :0
Short :0
String :null

Java Operators

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups –

- Arithmetic Operators(+,-,*,/,%)
- Relational Operators (<,>,>=,<=,==,!=)
- Bitwise Operators(>,<)
- Logical Operators(&&,||,!)
- Assignment Operators(=)

The Arithmetic Operators

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators –

Assume integer variable A holds 10 and variable B holds 20, then –

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	A + B will give 30
- (Subtraction)	Subtracts right-hand operand from left-hand operand.	A - B will give -10

* (Multiplication)	Multiplies values on either side of the operator.	A * B will give 200
/ (Division)	Divides left-hand operand by right-hand operand.	B / A will give 2
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	B % A will give 0
++ (Increment)	Increases the value of operand by 1.	B++ gives 21
-- (Decrement)	Decreases the value of operand by 1.	B-- gives 19

The Relational Operators

There are following relational operators supported by Java language.

Operator	Description	Example
== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
> (greater than)	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then	(A < B) is true.

	condition becomes true.	
>= (greater than or equal to)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

-The Bitwise Operators

Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte.

Bitwise operator works on bits and performs bit-by-bit operation.

Assume if a = 60 and b = 13; now in binary format they will be as follows –

a = 0011 1100

b = 0000 1101

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a = 1100 0011

The following table lists the bitwise operators

Operator	Description	Example
& (bitwise and)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
(bitwise or)	Binary OR Operator copies a bit if it exists in either operand.	(A B) will give 61 which is 0011 1101
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~ (bitwise)	Binary Ones Complement Operator is unary and	(~A) will give -61 which is 1100

compliment)	has the effect of 'flipping' bits.	0011 in 2's complement form due to a signed binary number.
<< (left shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>> (right shift)	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>> (zero fill right shift)	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

The Logical Operators

The following table lists the logical operators

Operator	Description	Example
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false
(logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A B) is true
! (logical not)	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true

The Assignment Operators

Following are the assignment operators supported by Java language

Operator	Description	Example
----------	-------------	---------

		$C = C \ll 2$
$\gg=$	Right shift AND assignment operator.	$C \gg= 2$ is same as $C = C \gg 2$
$\&=$	Bitwise AND assignment operator.	$C \&= 2$ is same as C $= C \& 2$
$\wedge=$	bitwise exclusive OR and assignment operator.	$C \wedge= 2$ is same as C $= C \wedge 2$
$ =$	bitwise inclusive OR and assignment operator.	$C = 2$ is same as C $= C 2$

Java Math Functions

The `java.lang.Math` class provides us access to many of these functions. The table below lists some of the more common among these. (*There are more than just these in `java.lang.Math`, however -- one should consult the java API for the whole list.*)

Method	Description
Math.abs()	It will return the Absolute value of the given value.
Math.max()	It returns the Largest of two values.
Math.min()	It is used to return the Smallest of two values.
Math.round()	It is used to round of the decimal numbers to the nearest value.
Math.sqrt()	It is used to return the square root of a number.

Math.cbrt()	It is used to return the cube root of a number.
Math.pow()	It returns the value of first argument raised to the power to second argument.
Math.signum()	It is used to find the sign of a given value.
Math.ceil()	It is used to find the smallest integer value that is greater than or equal to the argument or mathematical integer.

Decision Making

Decision Making in Java (if, if-else, switch, break, continue, jump)

A programming language uses control statements to control the flow of execution of program based on certain conditions. These are used to cause the flow of execution to advance and branch based on changes to the state of a program.

Java's Selection statements:

- if
- if-else
- nested-if
- if-else-if ladder
- switch-case

These statements allow you to control the flow of your program's execution based upon conditions known only during run time.

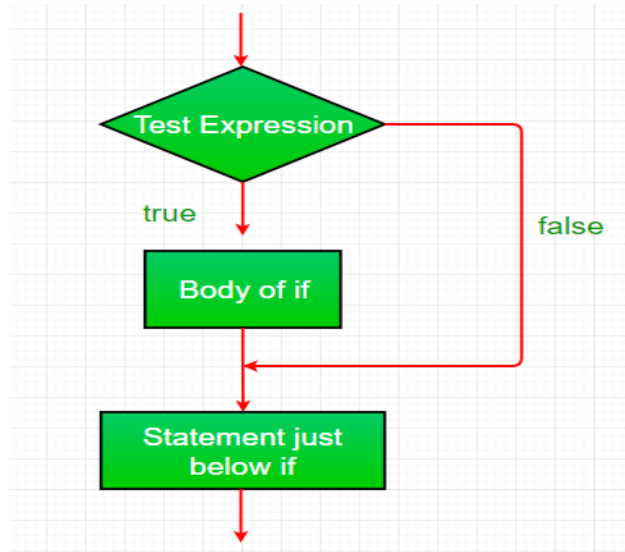
1) **if**: if statement is the most simple decision making statement.

It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.

Syntax:

```
if(condition)
{
    // Statements to execute if
    // condition is true
}
```

Flow Chart



```
// Java program to illustrate If statement
class IfDemo
{
    public static void main(String args[])
    {
        inti = 10;

        if (i> 0)
        {
            System.out.println("number is +ve");
        }

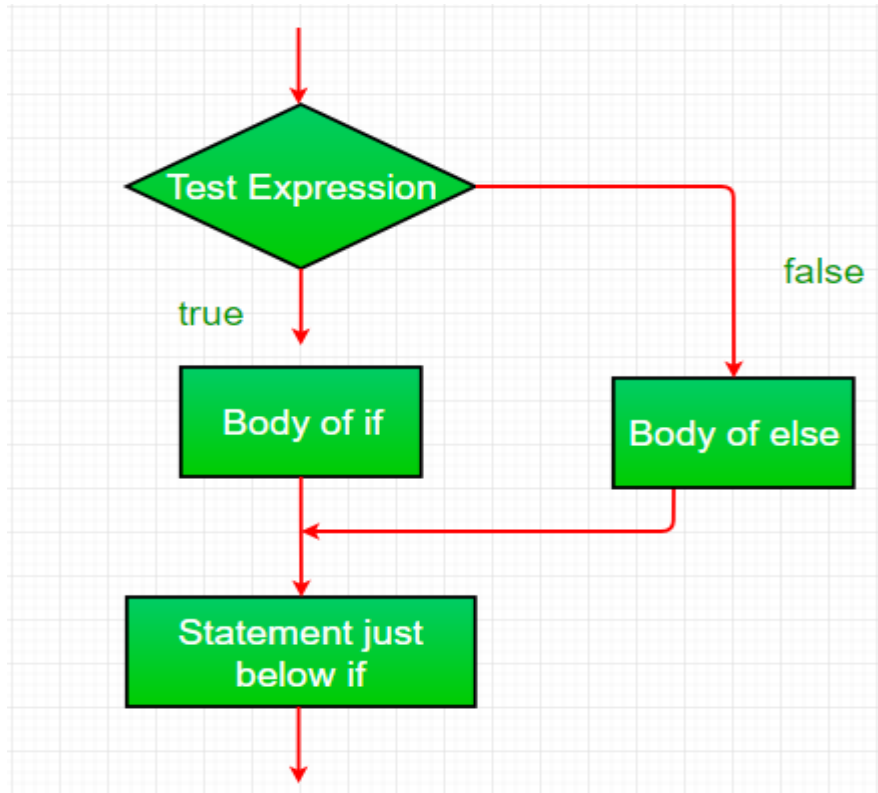
    }
}
```

Output:
Number is +ve

2)**if-else**: The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't. But what if we want to do something else if the condition is false. Here comes the else statement. We can use the else statement with if statement to execute a block of code when the condition is false.

Syntax:

```
if (condition)
{
    // Executes this block if
    // condition is true
}
else
{
    // Executes this block if
    // condition is false
}
```



```

// Java program to illustrate if-else statement
class IfElseDemo
{
    public static void main(String args[])
    {
        int i = 10;

        if (i>0)
            System.out.println("Number is +ve");
        else
            System.out.println("Number is -ve");
    }
}

```

Output:

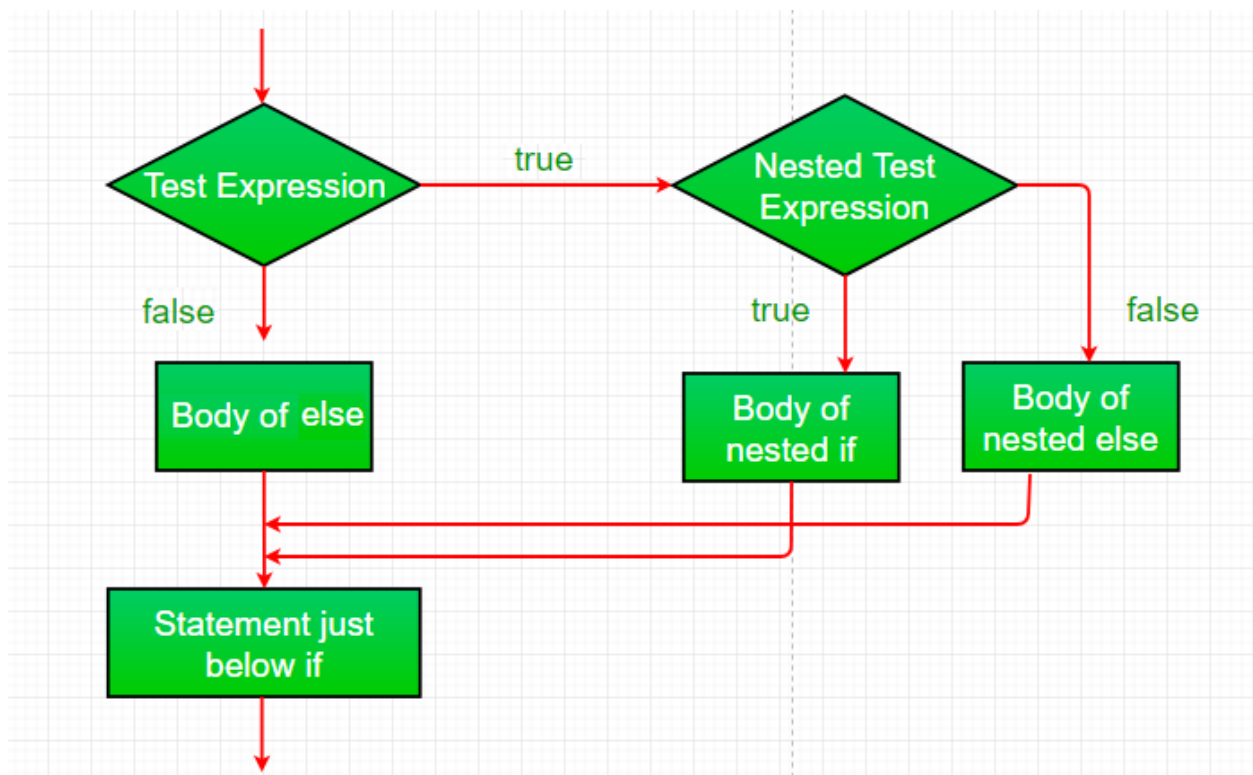
Number is +ve

3)nested-if: A nested if is an if statement that is the target of another if

or else. Nested if statements means an if statement inside an if statement. Yes, java allows us to nest if statements within if statements. i.e, we can place an if statement inside another if statement.

Syntax:

```
if (condition1)
{
    // Executes when condition1 is true
    if (condition2)
    {
        // Executes when condition2 is true
    }
}
```



```
// Java program to illustrate nested-if statement
Write program to display days of week according to input 1-7
Class a
```

```
{
    Public static void main(String args[])
    {
        int a=1;
        if(a==1)
        {
            System.out.print("today is Sunday");
        }
        If(a==2)
        {
            System.out.print("today is Monday");
        }
        If(a==3)
        {
            System.out.print("today is Tuesday");
        }
        If(a==4)
        {
            System.out.print("today is Wednesday");
        }
        If(a==5)
        {
            System.out.print("today is Thursday");
        }
        If(a==6)
        {
            System.out.print("today is Friday");
        }
        If(a==7)
        {
            System.out.print("today is Saturday");
        }

    }
}
```


Output:

Today is sunday

4)if-else-if ladder: Here, a user can decide among multiple options. The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.

```
if (condition)
```

```
    statement;
```

```
else if (condition)
```

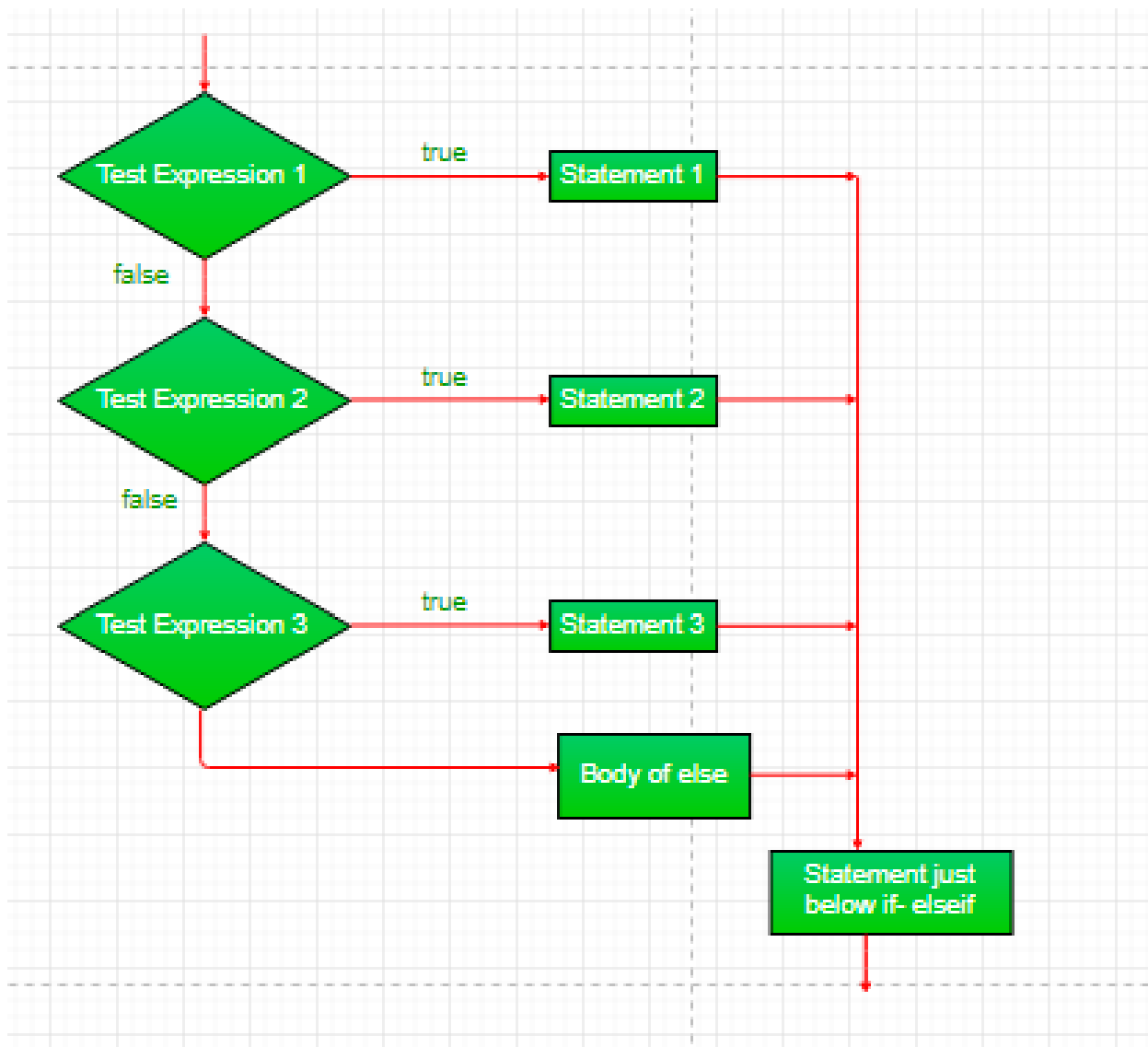
```
    statement;
```

```
·
```

```
·
```

```
else
```

```
    statement;
```



```
// Java program to illustrate if-else-if ladder
Write program to display days of week according to input 1-7
Class a
{
    Public static void main(String args[])
    {
        int a=1;
        if(a==1)
```

```
    {
        System.out.print("today is Sunday");
    }
    If(a==2)
    {
        System.out.print("today is Monday");
    }
    else If(a==3)
    {
        System.out.print("today is Tuesday");
    }
    else If(a==4)
    {
        System.out.print("today is Wednesday");
    }
    else If(a==5)
    {
        System.out.print("today is Thursday");
    }
    else If(a==6)
    {
        System.out.print("today is Friday");
    }
    else If(a==7)
    {
        System.out.print("today is Saturday");
    }
    Else
    {
        System.out.print("wrong key");
    }

}
}
```

Output:

Today is sunday

5)switch-case The switch statement is a multiway branch statement.

It provides an easy way to dispatch execution to different parts of code based on the value of the expression.

Syntax:

```
switch (expression)
```

```
{
```

```
  case value1:
```

```
    statement1;
```

```
    break;
```

```
  case value2:
```

```
    statement2;
```

```
    break;
```

```
  .
```

```
  .
```

```
  case valueN:
```

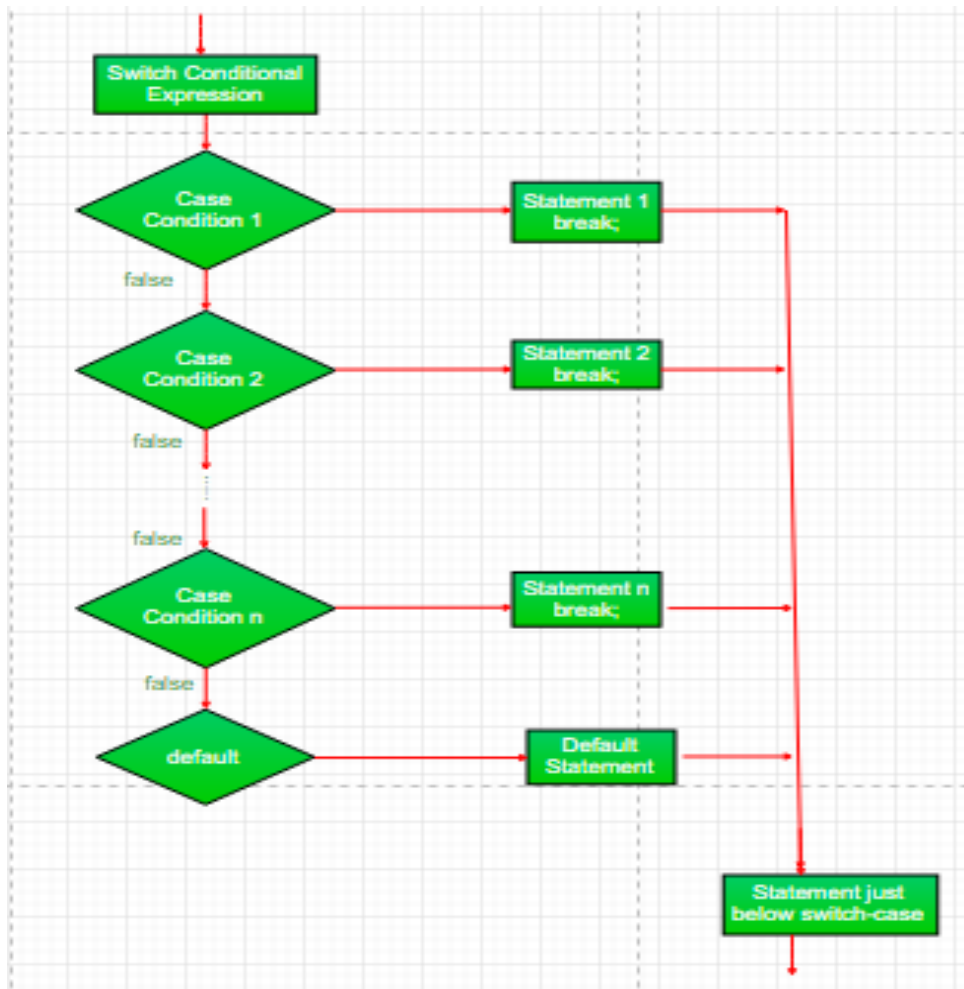
```
statementN;
```

```
  break;
```

```
  default:
```

```
statementDefault;
```

```
}
```



- // Java program to illustrate switch-case
class SwitchCaseDemo
{
public static void main(String args[])
{
inti = 9;
switch (i)
{
case 0:
System.out.println("i is zero.");

```

        break;
    case 1:
        System.out.println("i is one.");
        break;
    case 2:
        System.out.println("i is two.");
        break;
    default:
        System.out.println("i is greater than 2.");
    }
}
}

```

- Output:
- i is greater than 2.

6)jump: Java supports three jump statement: **break**, **continue** and **return**. These three statements transfer control to other part of the program.

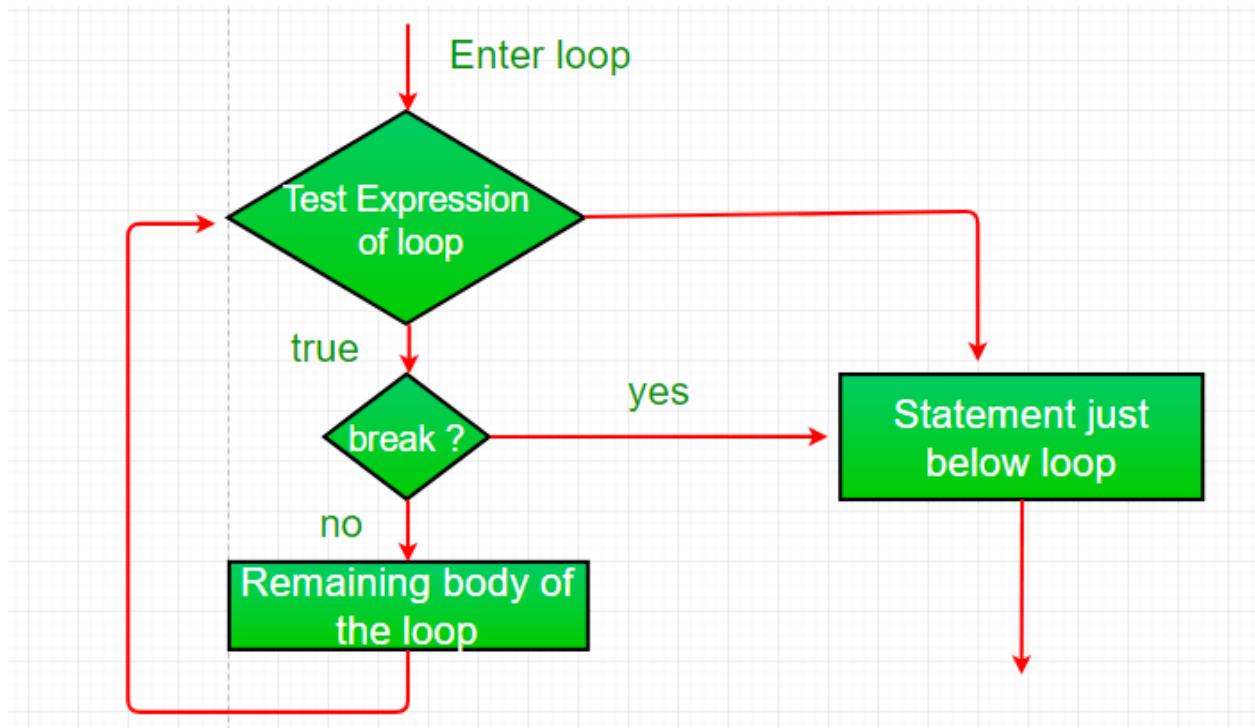
1. **Break:** In Java, break is majorly used for:

- Terminate a sequence in a switch statement (discussed above).
- To exit a loop.
- Used as a “civilized” form of goto.

Using break to exit a Loop

Using break, we can force immediate termination of a loop, bypassing the conditional expression and any remaining code in the body of the loop.

Note: Break, when used inside a set of nested loops, will only break out of the innermost loop.



Loops in Java

Looping in programming languages is a feature which facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true.

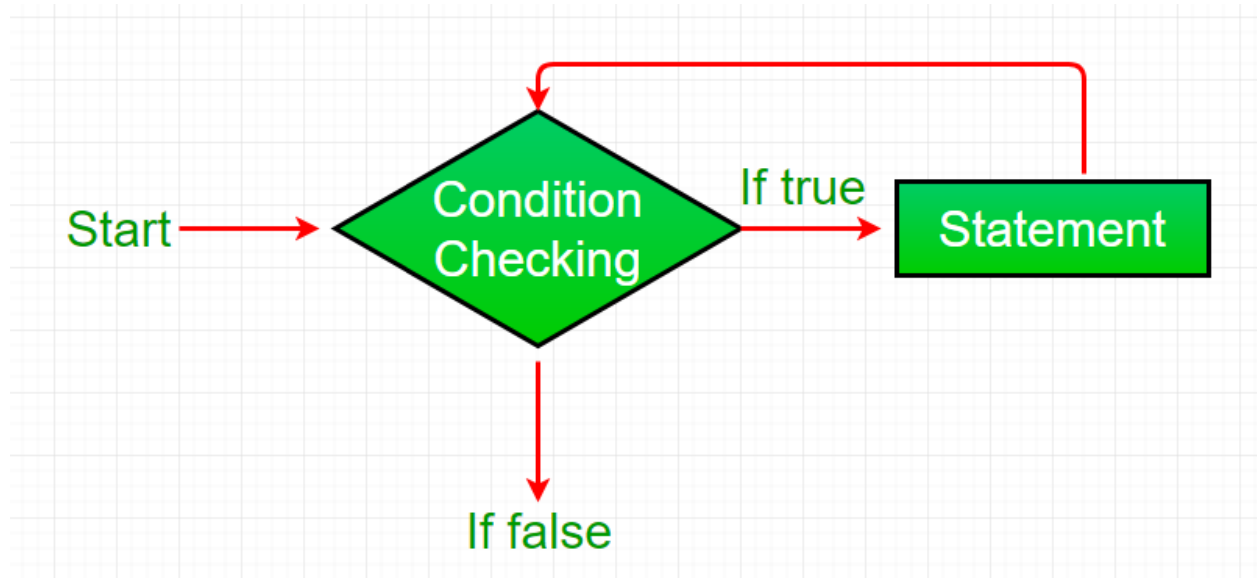
Java provides three ways for executing the loops. While all the ways provide similar basic functionality, they differ in their syntax and condition checking time.

1. **while loop:** A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

Syntax :

2. while (boolean condition)
3. {
4. loop statements...
5. }

Flowchart:



1.

```
// Java program to illustrate while loop
class whileLoopDemo
{
    public static void main(String args[])
    {
        int i;
        i=1;
        while(i<=10)
        {
            System.out.println(i);
            i++;
        }
    }
}
```

2. **Output:**

3. Value of x:1
4. Value of x:2
5. Value of x:3

6. Value of x:4

7. **for loop:** for loop provides a concise way of writing the loop structure. Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.

Syntax:

8. for (initialization condition; testing condition;

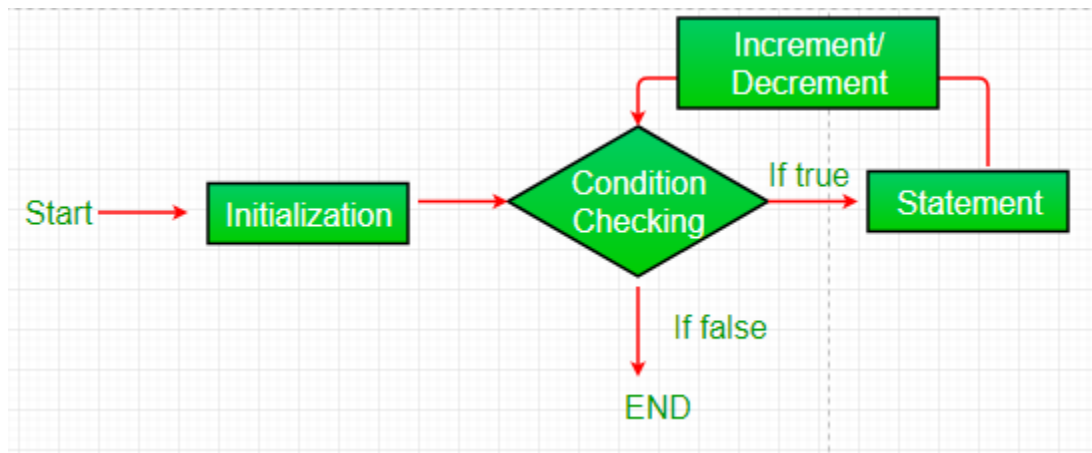
9. increment/decrement)

10. {

11. statement(s)

}

Flowchart:



// Java program to illustrate for loop.

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

Output:

Value of x:2

Value of x:3

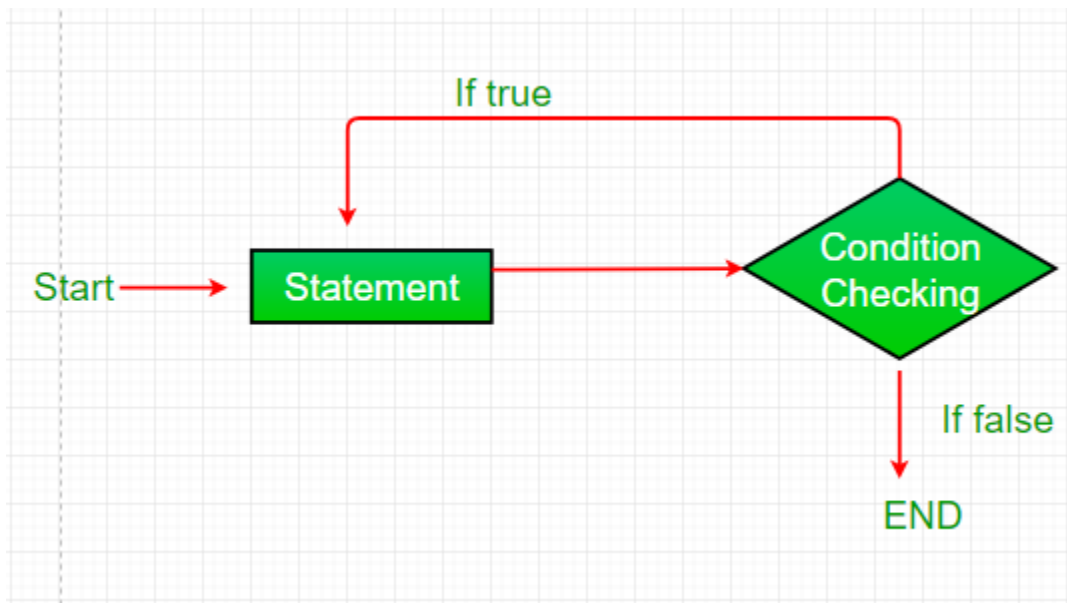
Value of x:4

3)do while: do while loop is similar to while loop with only difference that it checks for condition after executing the statements, and therefore is an example of **Exit Control Loop**.

Syntax:

```
do
{
    statements..
}
while (condition);
```

Flowchart:



```
// Java program to illustrate do-while loop
class dowhileloopDemo
{
    public static void main(String args[])
```

```
{
    int x = 21;
    do
    { do
    {
        System.out.println(i);
        i++;
    } while(i<=10);
    }
    while (x < 20);
    }
}
```

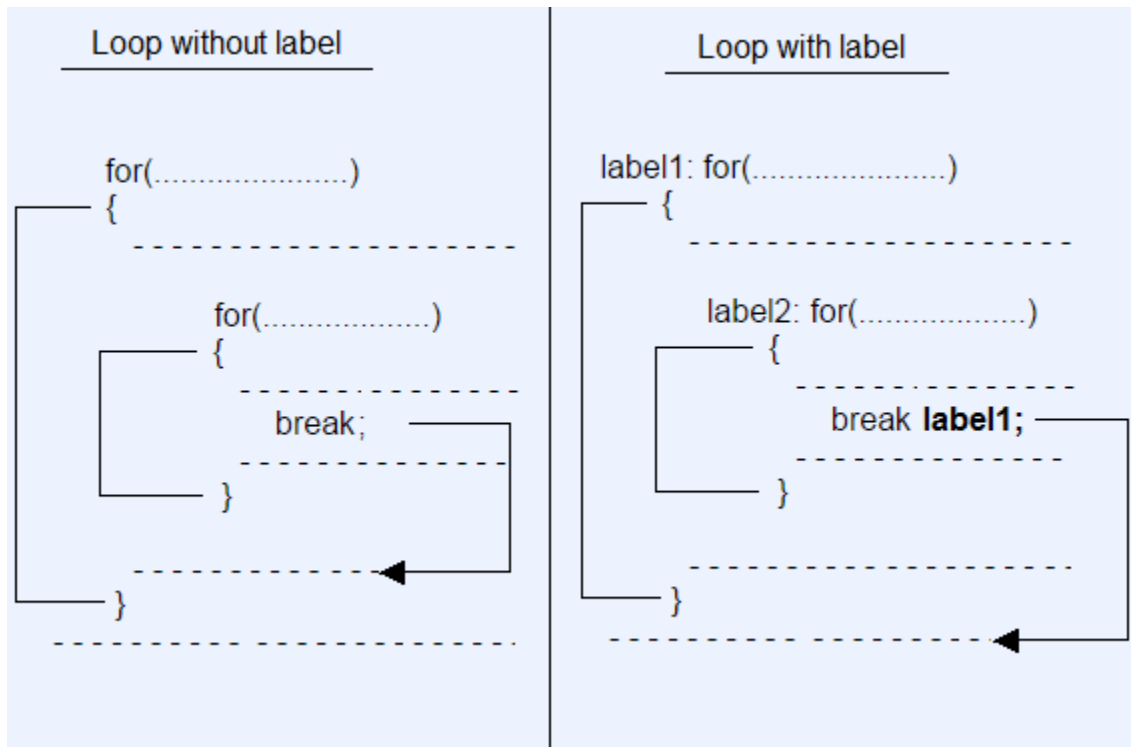
Output:

Value of x: 21

4)Labelled Loop(2 marks)

According to nested loop, if we put break statement in inner loop, compiler will jump out from inner loop and continue the outer loop again. What if we need to jump out from the outer loop using break statement given inside inner loop? The answer is, we should define **lable** along with colon(:) sign before loop.

Syntax of Labelled loop



Example with labelled loop

```
//WithLabelledLoop.java

class WithLabelledLoop
{
    public static void main(String args[])
    {
        inti,j;

        loop1: for(i=1;i<=10;i++)
        {
            System.out.println();

        }
    }
}
```

Output :

1 2 3 4 5

Comments in Java

Java Comments

Comments can be used to explain Java code, and to make it more readable. It can also be used to prevent execution when testing alternative code.

Single-line comments start with two forward slashes (//).

Any text between // and the end of the line is ignored by Java (will not be executed).

```
/*comments */
```

```
// This is a comment
```

```
System.out.println("Hello World");
```

Using multiple classes in a Java program

A Java program may contain any number of classes. The following program comprises of two classes: Computer and Laptop, both the classes have their constructors and a method. In the main method, we create objects of two classes and call their methods.

Using two classes in Java program

```
class Computer {
```

```
    Computer() {
```

```
        System.out.println("Constructor of Computer class.");
```

```
    }
```

```
    void computer_method() {
```

```
        System.out.println("Power gone! Shut down your PC soon...");
```

```
    }
```

```
public static void main(String[] args) {
```

```
    Computer my = new Computer();
```

```
    Laptop your = new Laptop();
```

```
    my.computer_method();
```

```
    your.laptop_method();
```

```
}
```

```

}
class Laptop {
    Laptop() {
        System.out.println("Constructor of Laptop class.");
    }

    void laptop_method() {
        System.out.println("99% Battery available.");
    }
}

```

Section-B

CLASSES, OBJECTS AND METHODS: - Introduction; Defining a Class; Adding Variables; Adding Methods; Creating Objects; Accessing Class Members; Constructors; Methods Overloading; Static Members; Nesting of Methods;

Inheritance: Extending a Class; Overriding Methods; Final Variables and Methods; Final Classes; Finalizer Methods; Abstract Methods and Classes; Visibility Control.

Classes and objects in Java

1. A **class** is nothing but a blueprint or a template for creating different objects which defines its properties and behaviors. Java class objects exhibit the properties and behaviors defined by its class. A class can contain fields and methods to describe the behavior of an object.

Syntax

```

Class classname
{
    Methods(functions)
    variables
}

```

2.Methods are nothing but members of a class that provide a service for an object or perform some business logic. Java fields and member functions names are case sensitive. Current states of a class's corresponding object are stored in the object's instance variables. Methods define the operations that can be performed in java programming.

```
Returntype method name (parameterlist)
{
    Block of code
}
```

3.Java object

object is an instance of a class created using a **new** operator. The new operator returns a reference to a new instance of a class. This reference can be assigned to a reference variable of the class. The process of creating objects from a class is called instantiation. An object encapsulates state and behavior.

Syntax

```
Classname objectname=new classname();
```

4.Accessing class methods and variables

```
Object.methodname();
Object.variablename;
```

Example to add two number

```
class MyClass
{
    int x,y,z;
    void get()
    {
        x = 5;
        y=2;
    }

    void disp()
    {
        z=x+y;
    }
}
```

```
System.out.print("sum is "+z);  
}}
```

```
Class mainmethod  
{  
    Public static void main(String args[])  
    {  
        Myclass ob=Myclass();  
        ob.get();  
        ob.disp();  
    }  
}}
```

Output : sum is 7

Java Constructors

A constructor in Java is a **special method** that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes:

Types of Constructor

1.Default Constructor

2.Parameterized Constructor

1.Default Constructor:A constructor without Parameters .it have same class name and have no return type

Example

```
class MyClass
```

```
{  
    Int x,y,z;
```

```
MyClass()
```

```
{  
    x = 5;
```

```
    y=2;
```

```
}
```

```
Void disp()
```



```
{
int z=x+y;

System.out.print(z);
}}
```

```
Class mainmethod
{
    Public static void main(String args[])
    {
        Myclass ob=Myclass();
        Ob.disp();
    }
}
```

2.Parameterized Constructor:A constructor have Parameters .it have same class name and have no return type

```
class MyClass
{
    Int x,y,z;

    MyClass()
    {
        X1=x;

        Y1=y;
    }
    Void disp()
    {
        int z=x+y;

        System.out.print(z);
    }
}
```

```
Class mainmethod
{
    Public static void main(String args[])
    {
        Myclass ob=Myclass(5,4);
    }
}
```

```
Ob.disp();  
}}
```

Method Overloading

With **method overloading**, multiple methods can have the same name with different parameters:

```
int myMethod(int x)  
float myMethod(float x)  
double myMethod(double x, double y)
```

Example

```
class DisplayOverloading2  
{  
    public void disp(char c)  
    {  
        System.out.println(c);  
    }  
    public void disp(int c)  
    {  
        System.out.println(c );  
    }  
}  
  
class Sample2  
{  
    public static void main(String args[])  
    {  
        DisplayOverloading2 obj = new DisplayOverloading2();  
        obj.disp('a');  
        obj.disp(5);  
    }  
}
```

Output:

a

Java static method

If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

Example of static method

```
static void add(int a,int b)
{
    int c;
    c=a+b; 5+4
    System.out.println("sum is"+c);
}
Class test
{
    Public static void main(String args[])
    {
        add(5,4);
    }
}
```

Output

Sum is 9

Nesting of Methods in java

A method of a class can be called only by an object of that class using the dot operator. So, there is an exception to this. A method can be called by using only its name by another method of the same class that is called Nesting of Methods.

Program:

Class compare

```
{
    int a,b;
    void get()
{
    a=2;
    b=3;
```

```

}
int comp()
{
    if(a>b)
    {
        return(a);
    }
else
{
    return(b);
}}
void disp()
{
System.out.println(comp( )); //nesting of method
}
Class test
{
    Public static void main(String args[])
    {

        compare ob=new compare();
ob.get();
ob.disp();
}
Output
3

```

Inheritance: Extending a Class; Overriding Methods; Final Variables and Methods; Final Classes; FinalizerMethods;AbstractMethodsandClasses;VisibilityControl.

Inheritance in Java

It is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of [OOps](#) (Object Oriented programming system).

create new [classes](#) that are built upon existing classes. When you inherit from an

existing class, you can reuse methods and fields of the parent class.

Syntax

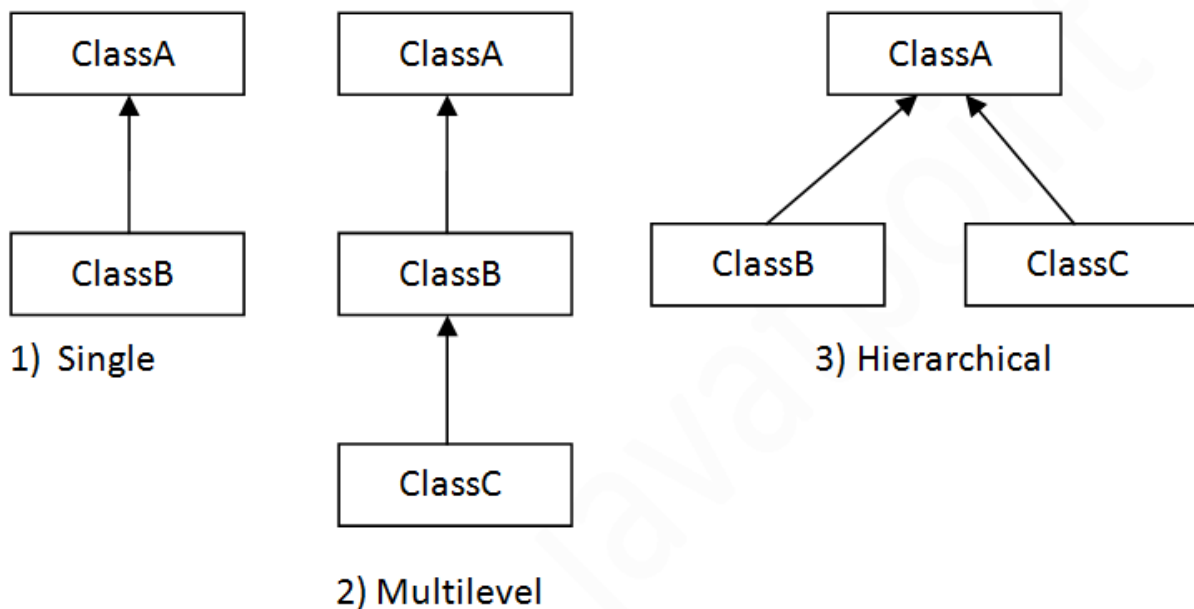
1. **class** Subclass-name **extends** Superclass-name
2. {
3. //methods and fields
4. }

The **extends** keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: **single**, **multilevel** and **hierarchical**.

In java programming, **multiple and hybrid inheritance** is supported through interface only. We will learn about interfaces later.



Single Level

When a class inherits another class, it is known as a *single inheritance*. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

```
Class a
{
    int a,b,c;
    void get()
    {
        a=2;
        b=3;
    }
    void disp()
    {
        c=a+b;
        System.out.print("sum is "+c);
    }
}
class b extends a
{
    int d
    void disp2()
    {
        d=a-b;
        System.out.println("Subtraction is "+d);
    }
}
class test
{
    public static void main(String args[])
    {
        b ob=new b();
        ob.get();
        ob.disp();
    }
}
```

```
ob.disp2();
}
Output
Sum is 5
Subtraction is -1
```

Method Overriding

If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.

Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

```
class Vehicle
{
    void run()
    {
        System.out.println("Vehicle is running");
    }
}

class Bike2 extends Vehicle
{
    void run()
    {
        System.out.println("Bike is running safely");
    }
}

Class test
{
    public static void main(String args[])
    {
        Bike2 obj = new Bike2();
    }
}
```

```
Vehicle ob=new Vehicle();
ob.run();
obj.run();
}
}
```

Output

Vehicle is running

Bike is running safely

Visibility Controls

Private :Not access by all classes

Public :Accessed by all classes

Protected :only 1 time inheritance

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

final variable

Variables defined in an interface are implicitly final. You can't change value of a final variable (is a constant).

Example

```
final int a=10;
```

Final method

. A final method can't be overridden when its class is inherited. Any attempt to override or hide a final method **will result in a compiler error**.

Example

```
final void get()
```



```
{  
  
}
```

final class

A final class can't be extended i.e., final class may not be subclassed. This is done for security reasons with basic classes like String and Integer.

final class classname

```
{  
  
}
```

Finalize method

Java Object finalize() Method

Finalize() is the method of Object class. This method is called just before an object is garbage collected. finalize() method overrides to dispose system resources, perform clean-up activities and minimize memory leaks.

Syntax finalize()

```
{  
}
```

Difference between Method Overloading and Method Overriding

No.	Method Overloading	Method Overriding
1)	Method overloading is used <i>to increase the readability</i> of the program.	Method overriding is used <i>to provide the specific implementation</i> of the method that is already provided by its super class.
2)	Method overloading is performed <i>within class</i> .	Method overriding occurs <i>in two classes</i> that have IS-A (inheritance) relationship.

3)	In case of method overloading, <i>parameter must be different.</i>	In case of method overriding, <i>parameter must be same.</i>
4)	Method overloading is the example of <i>compile time polymorphism.</i>	Method overriding is the example of <i>run time polymorphism.</i>
5)	In java, method overloading can't be performed by changing return type of the method only. <i>Return type can be same or different</i> in method overloading. But you must have to change the parameter.	<i>Return type must be same or covariant</i> in method overriding.

Java Abstract Classes and Methods(Dynamic Dispatch method)

Data **abstraction** is the process of hiding certain details and showing only essential information to the user.

Abstraction can be achieved with either **abstract classes** or **interfaces** (which you will learn more about in the next chapter).

The abstract keyword is a non-access modifier, used for classes and methods:

- **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
- **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

An abstract class can have both abstract and regular methods:

```
abstract class Animal
{
    public abstract void animalSound();
```

```
}
```

```
class Pig extends Animal  
{  
    public void animalSound()  
    {  
  
        System.out.println("This is pig class");  
    }  
}
```

```
class MyMainClass  
{  
    public static void main(String[] args)  
    {  
        Pig myPig = new Pig();  
        myPig.animalSound();  
  
    }  
}
```

Output

This is pig class

Visibility Controls in java

Modifier	Class	Interface	Enum	Inner class	Variable	Method	Constructor
public	✓	✓	✓	✓	✓	✓	✓
<default>	✓	✓	✓	✓	✓	✓	✓
protected	✗	✗	✗	✓	✓	✓	✓
private	✗	✗	✗	✓	✓	✓	✓
final	✓	✗	✗	✓	✓	✓	✗
abstract	✓	✓	✗	✓	✗	✓	✗
strictfp	✓	✓	✓	✓	✗	✓	✗
static	✗	✗	✗	✓	✓	✓	✗
synchronized	✗	✗	✗	✗	✗	✓	✗
native	✗	✗	✗	✗	✗	✓	✗
transient	✗	✗	✗	✗	✓	✗	✗
volatile	✗	✗	✗	✗	✓	✗	✗

by- Manish K Singh

Arrays in Java

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

Types of Arrays

- ✓ One-Dimensional =It have 1 subscript value
- ✓ Two-Dimensional =It have 2 subscript value
- ✓ Multi-Dimensional =It have more than one subscript value

Declare

```
Datatype arrayname[];
int a[]=new a{1,2,3,4,5}
```

Program store Elements in array using 1D array

```

class JavaProgram
{
    public static void main(String args[])
    {

        int arr[ ]={ 10,20,30,40,50};
        int i;

        System.out.print("Elements in Array is :\n");
        for(i=0; i<5; i++)
        {
            System.out.print(arr[i] + " ");
        }

    }
}

```

Output

Elements in Array is

10 20 30 40 50

/*Program 2d array*/

Class 2darray

```

{
    Public static void main(String args[])
    {
        int[] [] a={{ 34,45 },{ 67,78 }};
        int i,j;
        for(i=0;i<2;i++)
        {
            for(j=0;j<2;j++)
            {
                System.out.print(a[i][j]);
            }
            System.out.println();
        }
    }
}

```

```
}  
}
```

Output 2*2=4

34 45

67 78

Java Strings

Strings are used for storing text.

A String variable contains a collection of characters surrounded by double quotes:

Example

Create a variable of type String and assign it a value:

```
String s1 = "Hello guys ";
```

```
/*program to store string */  
import java.io.*;  
class a  
{  
    public static void main(String args[])  
    {  
        String a="hello guys how r u?";  
        System.out.print("String is "+a);  
  
    }  
}
```

Output

String is hello guys how r u?

String Methods

1. length()

A String in Java is actually an object, which contain methods that can perform

certain operations on strings. For example, the length of a string can be found with the `length()` method:

Example

```
String txt = "hello";
```

```
System.out.println("The length of the txt string is: " + txt.length());
```

Output

The length of the txt string is :5

2. **toUpperCase():it is convert a string into upper letters**

Example

```
String txt = "Hello World";
```

```
System.out.println(txt.toUpperCase());
```

```
// Outputs "HELLO WORLD"
```

3.**toLowerCase():it is convert a string into lower letters**

Example

```
String txt = " HELLO WORLD ";
```

```
System.out.println(txt.toLowerCase());
```

Output

hello world

4. **concat()**

The `+` operator can be used between strings to combine them. This is called **concatenation**:

Example

```
String firstName = "John";
```

```
String lastName = "Cina";
```

```
System.out.println(firstName+" " + concat() lastName);
```

Output

John Cina

5. **charAt(int index):** returns char value for the particular index

Example

```
String a = "John";  
System.out.println(charAt(1));
```

Output: o

6.int indexOf(int ch) :It returns the specified char value index.

```
String firstName = "John";  
System.out.println(indexOf(h));
```

Output: 2

Vector in java

Vector implements List Interface. Like ArrayList it also maintains insertion order but it is rarely used in non-thread environment as it is synchronized and due to which it gives poor performance in searching, adding, delete and update of its elements.

Vector methods

1.addElement ():this is used to add elements in array

2.clear():this is used to delete values in vector

3.size():used to find size of vector

4. nextElement():It is used to display next value of vector

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

```
public class VectorExample {
```

```
    public static void main(String args[])  
    {  
        Vector vec = new Vector (2);
```



```

        vec.addElement("Apple");
        vec.addElement(1);
        System.out.println("Size is:"+vec.size());
        System.out.print(vec.nextElement() + " ");
        vec.clear();
        System.out.println(vec.nextElement());

    }
}

```

Output

Size is 2

Apple 1

WrapperClasses

Wrapper classes are those whose objects wraps a primitive data type within them. In the **java.lang** package java provides a separate class for each of the primitive data types namely **Byte, Character, Double, Integer, Float, Long, Short**.

Syntax

```
Integer variable name=value;
```

```

import java.lang.*;
class MyClass
{
    public static void main(String[] args)
    {
        Integer myInt = 5;

        Double myDouble = 5.99;

        Character myChar = 'A';

        System.out.println(myInt);

        System.out.println(myDouble);
        System.out.println(myChar);
    }
}

```

```
}
}
5
5.99
A
```

INTERFACES(Multiple inheritance): Introduction;Defining Interfaces;ExtendingInterfaces;Implementing Interfaces; AccessingInterfaceVariables, ImplementingMultipleInheritanceusingInterfaces

Java Interface

Another way to achieve abstraction in Java, is with interfaces.

An interface is a completely "**abstract class**" that is used to group related methods with empty bodies:

To access the interface methods, the interface must be "**implemented**" (kinda like inherited) by another class with the implements keyword (instead of extends). The body of the interface method is provided by the "implement" class:

Syntax

```
interface interface name
{
    Final variables ;
    Public functions;
}

// Interface

interface Animal
{
    public void animalSound()

    public void sleep();
}
```

```

class Pig implements Animal
{
    public void animalSound()
    {

        System.out.println("class pig method animal sound");
    }
    public void sleep()
    {

        System.out.println("class pig method sleep");
    }
}

```

```

class MyMainClass
{
    public static void main(String[] args)
    {
        Pig myPig = new Pig();
        myPig.animalSound();
        myPig.sleep();
    }
}

```

Output:

```

class pig method animal sound
class pig method sleep

```

/*Example 2*/

```

interface add
{
    final int a=10;
    final int b=20;
    public void sum();
}
class A implements add
{

```

```

int c;
public void sum()
{
    c=a+b;
System.out.print("sum is"+c);
}
}
class test
{
    Public static void main(String args[])
    {
        A ob=new A();
        ob.sum();
    }
}
Sum is 30

```

Keywords

1. Interface keyword is used with interfaces
2. Variables must be final
3. Functions must be public
4. No object is used with interfaces
5. **Interfaces implementes**

Extends an interface

In below example, the interface B is extending another interface A. notice the syntax – “interface B extends A”

```

interface A {
    void fa();
}

```

```

interface B extends A {
    void fb();
}

```

```

/*

```

```

/* Interface extends another interface java example

```

```
*/
```

```
interface a1
```

```
{
```

```
public
```

```
    void show1();
```

```
}
```

```
interface b1 extends a1
```

```
{
```

```
public void show2();
```

```
}
```

```
class Test implements b1
```

```
{
```

```
    public void show1()
```

```
    {
```

```
        System.out.println("function show1");
```

```
    }
```

```
public void show2()
```

```
{
```

```
    System.out.println("function show2");
```

```
}
```

```
}
```

Class mainmethod

```
{  
  
public static void main(String args[])  
  
    {  
        Test ob=new Test();  
        ob.show1();  
        ob.how2();  
    }  
}
```

Output

```
function show1  
function show2
```

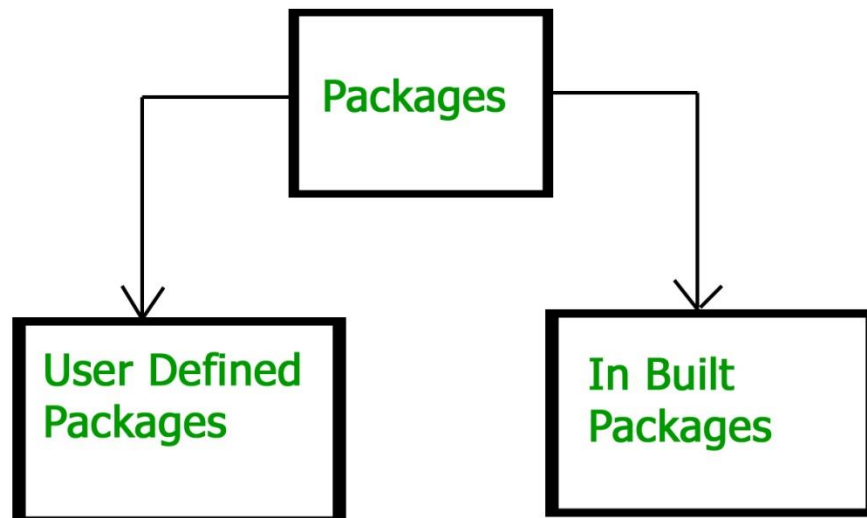
Sr. No.	Key	Class	Interface
1	Supported Methods	A class can have both an abstract as well as concrete methods.	Interface can have only abstract methods.
2	Multiple Inheritance	Multiple Inheritance is not supported.	Interface supports Multiple Inheritance.
3	Supported Variables	final, non-final, static and non-static variables	Only static and final variables are permitted

		supported.	
4	Keyword	A class is declared using class keyword.	Interface is declared using interface keyword.

Package

Package in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces. Packages can be considered as data encapsulation (or data-hiding).

Types of packages:



Built-in Packages

These packages consist of a large number of classes which are a part of Java **API**. Some of the commonly used built-in packages are:

1) **java.lang**: Contains language support classes (e.g. `Class` which defines

primitive data types, math operations). This package is automatically imported.

- 2) **java.io:** Contains classes for supporting input / output operations.
- 3) **java.util:** Contains utility classes which implement data structures like Linked List, Vector and support ; for Date / Time operations.
- 4) **java.applet:** Contains classes for creating Applets.

- 5) **java.awt:** Contain classes for implementing the components for graphical user interfaces (like button , ;menus etc).

- 6) **java.net:** Contain classes for supporting networking operations.

User-defined packages

These are the packages that are defined by the user. First we create a directory **myPackage** (name should be same as the name of the package). Then create the **MyClass** inside the directory with the first statement being the **package names**.

Steps for Creating package

1. Create a Package using keyword package.
Package packagename;

2. Create a folder name same have package name. Note **Package name and folder name must be same**.
3. Save package into folder
4. Create a new class with main function
5. Save this File outside of the package folder

How to import Package

There are Two ways to import package

1. **import packagename.*;**
2. **import packagename.classname;**

```
// Name of the package must be same as the directory  
// under which this file is saved  
package myPackage;
```

```
public class MyClass
```



```

{
    Public void disp()
    {
        System.out.println("hello package");
    }
}

```

Now we can use the **MyClass** class in our program.

```
/* import 'MyClass' class from 'names' myPackage */
```

Accessing package

There are two ways to call package

3. import packagename.*;
4. import packagename.Classname;

example:

```
import myPackage.MyClass;

public class PrintName
{
    public static void main(String args[])
    {
        MyClass ob=new MyClass();
        Ob.disp();
    }
}

```

Note : **MyClass.java** must be saved inside the **myPackage** directory since it is a part of the package.

\

Hiding a java package

```
package myPackage;
```

```
public class MyClass1
{
    Public void disp1()

```

```

    {
        System.out.println("hello package1");    //this is public
    }
}
public class MyClass2
{
    Public void disp()
    {
        System.out.println("hello package2");    //this is private
    }
}

```

/*import multiple packages into single main class*/

```

package pack1;
public class class2
{
    public void disp2()
    {
        System.out.print("hello");
    }
}
package pack;

```

```

public class MyClass1
{
    Public void disp1()
    {
        System.out.println("hello package1");    //this is public
    }
}

```

```

import pack.*;
import pack1.class2;
class test
{

```

```
public Static void main(String args[])
{

myclass2 ob2=new myclass2();
class2 ob=new class2();
ob2.disp1();
ob2.disp2();
ob.call();
}
}
```

Using Scanner in Java

// Java program to read data of various types using Scanner class.

```
import java.util.Scanner;
public class ScannerDemo1
{
    public static void main(String[] args)
    {

        Scanner sc = new Scanner(System.in);

        String name = sc.nextLine();

        char gender = sc.next().charAt(0);

        int age = sc.nextInt();
        long mobileNo = sc.nextLong();
        double cgpa = sc.nextDouble();

        System.out.println("Name: "+name);
        System.out.println("Gender: "+gender);
        System.out.println("Age: "+age);
        System.out.println("Mobile Number: "+mobileNo);
        System.out.println("CGPA: "+cgpa);
    }
}
```

SECTION-C

MANAGING ERRORS AND EXCEPTIONS:- Introduction;
Types of Errors; Exceptions; Exception Handling using
Try, Catch and Finally block; Throwing Our Own Exceptions; Using
Exceptions for Debugging.

Error : An Error “indicates serious problems that a reasonable application should not try to catch.”

Both Errors and Exceptions are the subclasses of java.lang.Throwable class. Errors are the conditions which cannot get recovered by any handling techniques

Exceptions : An Exception “indicates conditions that a reasonable application might want to catch.”

Exceptions are the conditions that occur at runtime and may cause the termination of program. But they are recoverable using try, catch and throw keywords.

In Java, there are two types of exceptions:

- 1) **Checked**: are the exceptions that are checked at compile time.

Example

1.a syntax errors

2. Missing semi colon

3. Use Wrong variable name, function name.

4. Missing brackets .

- 2) **Unchecked** are the exceptions that are not checked at compiled time or called run time error.

Eg

1. A number divide by zero.
2. Store a value in out of memory space. For `int a[5]={1,2,3,4,5,6}`
3. Wrong Logic. `c=a*b`

Below is the list of important **built-in exceptions** in Java.

1. **ArithmeticException**

It is thrown when an exceptional condition has occurred in an arithmetic operation.

2. **ArrayIndexOutOfBoundsException**

It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

3. **ClassNotFoundException**

This Exception is raised when we try to access a class whose definition is not found

4. **FileNotFoundException**

This Exception is raised when a file is not accessible or does not open.

5. **IOException**

It is thrown when an input-output operation failed or interrupted

6. **InterruptedException**

It is thrown when a thread is waiting , sleeping , or doing some processing , and it is interrupted.

7. **NoSuchFieldException**

It is thrown when a class does not contain the field (or variable) specified

8. **NoSuchMethodException**

It is thrown when accessing a method which is not found.

9. **NullPointerException**

This exception is raised when referring to the members of a null object. Null represents nothing

10. **NumberFormatException**

This exception is raised when a method could not convert a string

into a numeric format.

Exception handling

If an exception occurs, which has not been handled by programmer then program execution gets terminated and a system generated error message is shown to the user.

Java try and catch Statement

The **try statement** allows you to define a block of code to be tested for errors while it is being executed.

The **catch statement** allows you to define a block of code to be executed, if an error occurs in the try block.

Syntax

```
try
{
    // Block of code to try
}
catch(Exception e)
{
    // Block of code to handle errors
}
```

*/*Example divide a number by zero*/*

```
public class error
{
    public static void main (String args[])
    {
        int num1 = 15, num2 = 2;
        int result = 0;
        try
        {
            result = num1/num2;
```

```

        System.out.println("The result is" +result);
    }
    catch (ArithmeticException e)
    {
        System.out.println ("Can't be divided by Zero"+e);
    }
}
}

```

Java Multi-catch block

A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.

Points to remember

- At a time only one exception occurs and at a time only one catch block is executed.
- All catch blocks must be ordered from most specific to most general, i.e. catch for Arithmetic Exception must come before catch for Exception.

Example 1

```

public class MultipleCatchBlock1 {

    public static void main(String[] args) {

        try{
            int a[]=new int[5];
            a[5]=30/0;
        }
        catch(ArithmeticException e)
        {
            System.out.println("can not divide by zero");
        }
        catch(ArrayIndexOutOfBoundsException e)

```

```

        {
            System.out.println("ArrayIndexOutOfBoundsException Exception
occurs");
        }

    finally
    {
        System.out.println("end");
    }
}
}

```

Nested of try block: In Java, we can use a try block within a try block.

```

class NestedTry {

    // main method
    public static void main(String args[])
    {
        try {

            int a[] = { 1, 2, 3, 4, 5 };

            System.out.println(a[5]);

            try {

                int x = a[2] / 0;
            }
            catch (ArithmeticException e2)
            {

```



```

        System.out.println("division by zero is not possible");
    }
}
catch (ArrayIndexOutOfBoundsException e1) {
    System.out.println("Element at such index does not
exists");
}
}
}

```

Throw own Exception

```
import java.io.*;
```

```

class MyException extends Exception
{
    public MyException(String s)
    {
        super(s);
    }
}

```

// A Class that uses above MyException

```

public class Main
{
    public static void main(String args[])
    {
        try
        {
            throw new MyException("hello");
        }
        catch (MyException ex)
        {
            System.out.println(ex.getMessage());
        }
    }
}

```

```
    }  
  }  
}  
Output  
Hello
```

APPLET PROGRAMMING:- Introduction; How Applets Differ from Applications;AppletLife Cycle; Creating anExecutableApplet;PassingParameters toApplets;AligningtheDisplay;MoreaboutHTMLTags;DisplayingNumericalValues;Getting Input from theUser.

Applet

An applet is a Java program that can be embedded into a web page. It runs inside the web browser and works at client side. An applet is embedded in an HTML page using the APPLET or OBJECT tag and hosted on a web server.

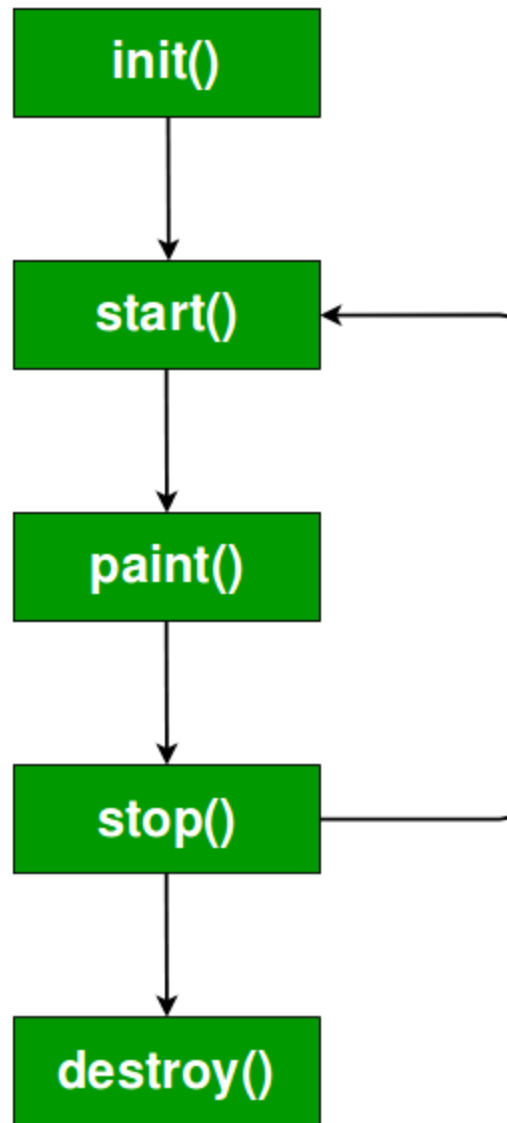
Important points :

1. All applets are sub-classes (either directly or indirectly) of [*java.applet.Applet*](#) class.
2. Applets are not stand-alone programs. Instead, they run within either a web browser or an applet viewer. JDK provides a standard applet viewer tool called applet viewer.
3. In general, execution of an applet does not begin at main() method.
4. Output of an applet window is not performed by *System.out.println()*. Rather it is handled with various AWT methods, such as *drawString()*.

Features of Applets over HTML

- Displaying dynamic web pages of a web application.
- Playing sound files.
- Displaying documents
- Playing animations
- Create online forms

Life cycle of an applet :



1. **init()** : The **init()** method is the first method to be called. This is where you should initialize variables. This method is called **only once** during the run time of your applet.
2. **start()** : The **start()** method is called after **init()**. It is also called to restart an applet after it has been stopped. Note that **init()** is called once i.e. when the first time an applet is loaded whereas **start()** is called each time an applet's HTML document is displayed onscreen.

3. **paint()** : The **paint()** method is called each time an AWT-based applet's output must be redrawn. **paint()** is also called when the applet begins execution. Whatever the cause, whenever the applet must redraw its output, **paint()** is called.
4. **stop()** : The **stop()** method is called when a web browser leaves the HTML document containing the applet—when it goes to another page, for example. When **stop()** is called, the applet is probably running. You should use **stop()** to suspend threads that don't need to run when the applet is not visible.
5. **destroy()** : The **destroy()** method is called when the environment determines that your applet needs to be removed completely from memory. At this point, you should free up any resources the applet may be using. The **stop()** method is always called before **destroy()**.

File one

Creating Hello World applet :

```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorld extends Applet
{

    public void paint(Graphics g)
    {
        g.drawString("Hello World", 20, 20);
    }
}
```

```
}
```

Save it by

HelloWorld.java

File 2

```
<applet code=" HelloWorld.class" height=100  
width=100></applet>
```

Save it by HelloWorld.html

Run it by appletviewer RunHelloWorld.html



Passing arguments to java applet

```
import java.applet.Applet;
```

```
import java.awt.Graphics;
```

```
public class UseParam extends Applet{
```

```
public void paint(Graphics g){
```

```
String str=getParameter("msg");
```

```
g.drawString(str,50, 50);
```

```
}
```

```
}
```

```
<applet code="UseParam.class" width="300" height="300">
  <param name="msg" value="Welcome to applet">
</applet>
```

the User in Java Applet

```
import java.awt.*;
import java.applet.*;
public class user extends Applet
{
    TextField t1, t2;
    public void init()
    {
        t1 = new TextField(10);
        t2 = new TextField(10);

        add(t1);
        add(t2);

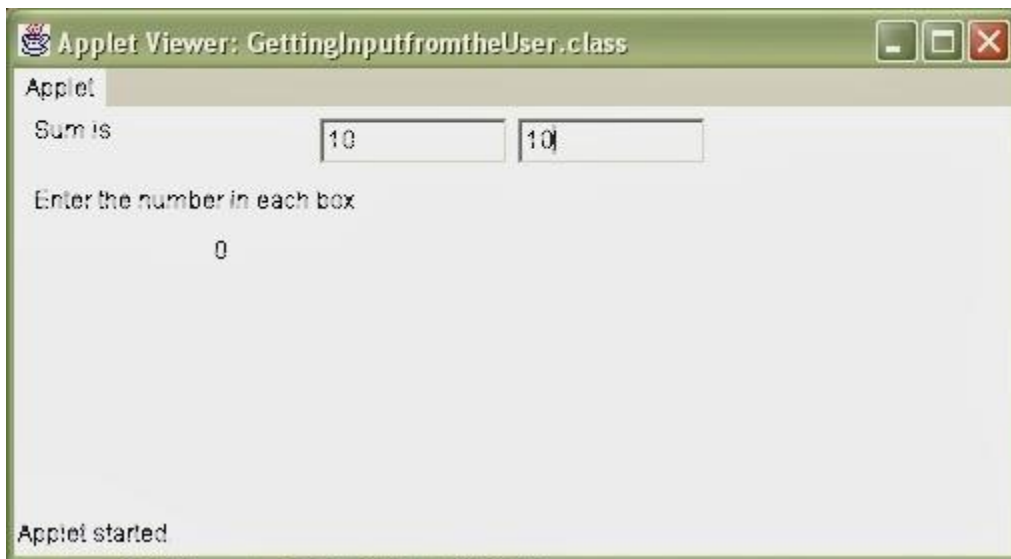
        t1.setText("0");
        t2.setText("0");
    }
    public void paint(Graphics g)
    {
        int a=0,b=0,c=0;
        String str1,str2,str;

        g.drawString("Enter the number in each box",10,50);

        try
        {
            str1=t1.getText();
            a=Integer.parseInt(str1);

            str2=t2.getText();
            b=Integer.parseInt(str2);
```

```
}  
catch(Exception e)  
{  
}  
c=a+b;  
  
str=String.valueOf(c);  
  
g.drawString("Sum is",10,15);  
g.drawString(str,100,75);  
}  
}
```



Graphics programming

Graphics is one of the most important features of Java. Java applets can be written to draw lines, arcs, figures, images and text in different fonts and styles. Different colors can also be incorporated in display.

The Graphics Class

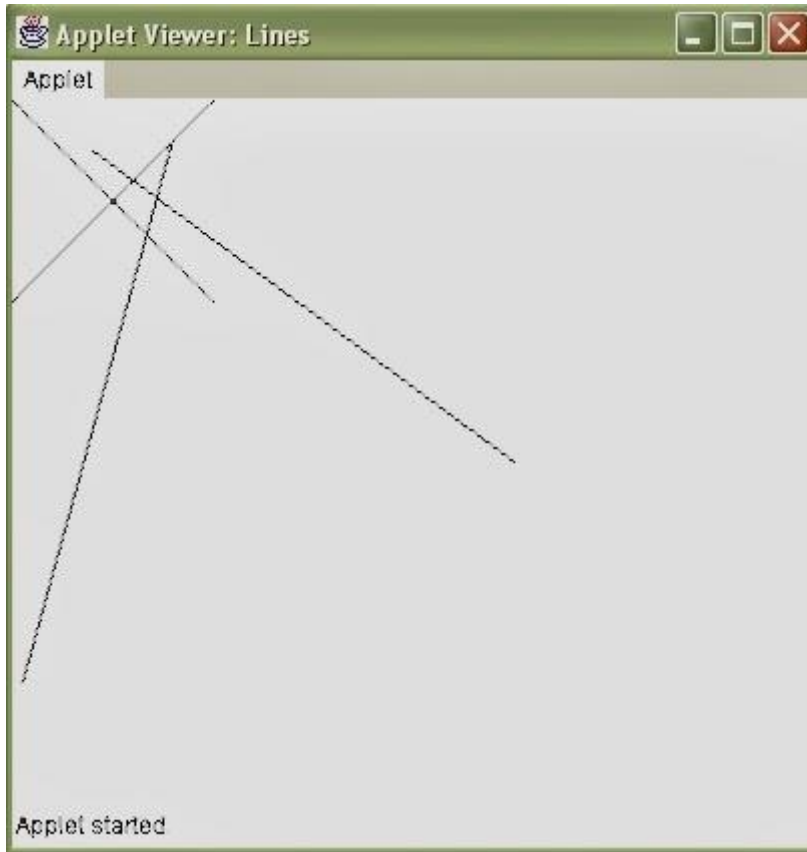
The graphics class defines a number of drawing functions, Each shape can be drawn edge-only or filled. To draw shapes on the screen, we may call one of the methods available in the graphics class.

1.Lines Lines are drawn by means of the **drawLine()** method.

Syntax

```
void drawLine(int startX, int startY, int endX, int endY)
```

```
//Drawing Lines
import java.awt.*;
import java.applet.*;
/*
<applet code="Lines" width=300 Height=250>
</applet>
*/
public class Lines extends Applet
{
    public void paint(Graphics g)
    {
        g.drawLine(0,0,100,100);
        g.drawLine(0,100,100,0);
        g.drawLine(40,25,250,180);
        g.drawLine(5,290,80,19);
    }
}
```



2. Rectangle

The `drawRect()` and `fillRect()` methods display an outlined and filled rectangle, respectively.

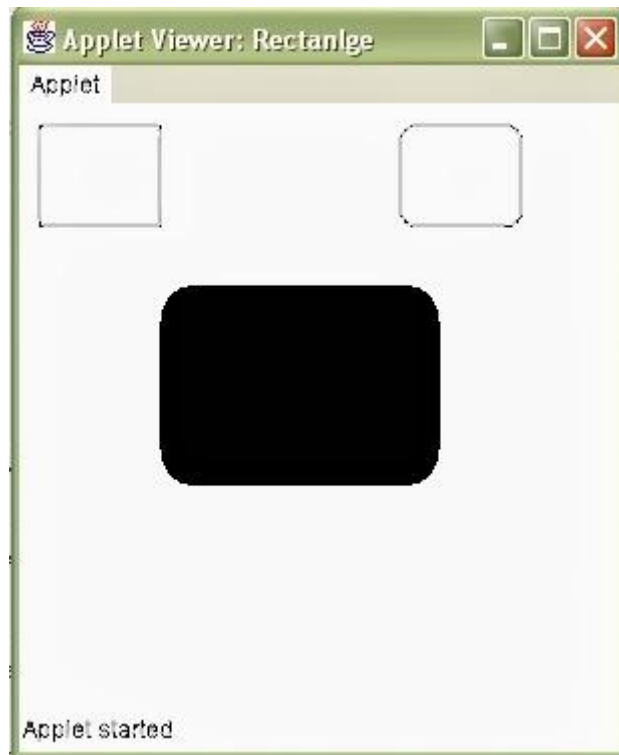
Syntax

```
void drawRect(int top, int left, int width, int height)
void fillRect(int top, int left, int width, int height)
import java.awt.*;
import java.applet.*;
/*
<applet code="Rectanlge" width=300 Height=300>
</applet>
*/
public class Rectanlge extends Applet
```

```

{
    public void paint(Graphics g)
    {
        g.drawRect(10,10,60,50);
        g.fillRect(100,100,100,0);
        g.drawRoundRect(190,10,60,50,15,15);
        g.fillRoundRect(70,90,140,100,30,40);
    }
}

```



3.Circles and Ellipses

The Graphics class does not contain any method for circles or ellipses. To draw an ellipse, use drawOval(). To fill an ellipse, use fillOval().

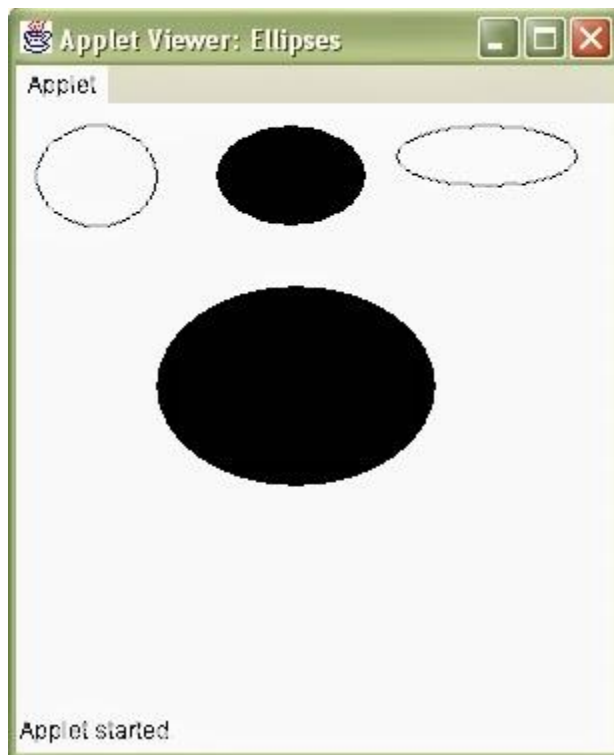
Syntax

```

void drawOval(int top, int left, int width, int height)
void fillOval(int top, int left, int width, int height)
import java.awt.*;

```

```
import java.applet.*;
/*
<applet code="Ellipses" width=300 Height=300>
</applet>
*/
public class Ellipses extends Applet
{
    public void paint(Graphics g)
    {
        g.drawOval(10,10,60,50);
        g.fillOval(100,10,75,50);
        g.drawOval(190,10,90,30);
        g.fillOval(70,90,140,100);
    }
}
```



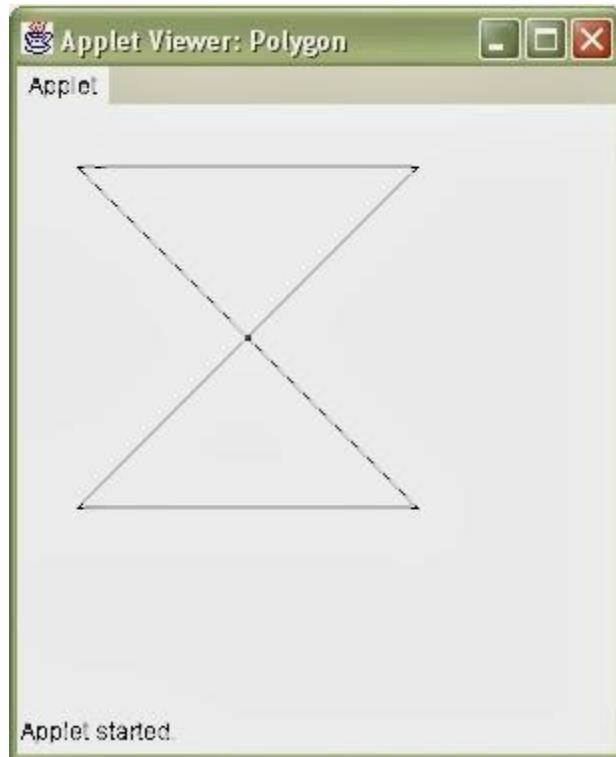
4.Drawing Polygons

Polygons are shapes with many sides. It may be considered a set of lines connected together. The end of the first line is the beginning of the

second line, the end of the second line is the beginning of the third line, and so on. Use drawPolygon() and fillPolygon() to draw arbitrarily shaped figures.

Syntax

```
void drawPolygon(int x[], int y[], int numPointer)
void fillPolygon(int x[], int y[], int numPointer)
import java.awt.*;
import java.applet.*;
/*
<applet code="Polygon" width=300 Height=300>
</applet>
*/
public class Polygon extends Applet
{
    public void paint(Graphics g)
    {
        int xpoints[]={ 30,200,30,200,30};
        int ypoints[]={ 30,30,200,200,30};
        int num=5;
        g.drawPolygon(xpoints,ypoints,num);
    }
}
```



```
/*program display bar charts using java applets */
import java.awt.*;
import java.applet.*;

public class BarChart extends Applet
{
    int n=0;
    String label[];
    int value[];

    public void init() {

        setBackground(Color.pink);
        try {

            int n = Integer.parseInt(getParameter("Columns"));
            label = new String[n];
```

```

value = new int[n];
label[0] = getParameter("label1");
label[1] = getParameter("label2");
label[2] = getParameter("label3");
label[3] = getParameter("label4");
value[0] = Integer.parseInt(getParameter("c1"));
value[1] = Integer.parseInt(getParameter("c2"));
value[2] = Integer.parseInt(getParameter("c3"));
value[3] = Integer.parseInt(getParameter("c4"));
}
catch(NumberFormatException e){ }
}
public void paint(Graphics g)
{
for(int i=0;i<4;i++) {
g.setColor(Color.black);
g.drawString(label[i],20,i*50+30);
g.setColor(Color.red);
g.fillRect(50,i*50+10,value[i],40);
}
}
}

```

```

/* <applet code=BarChart width=400 height=400>
<param name=c1 value=110>
<param name=c2 value=150>
<param name=c3 value=100>
<param name=c4 value=170>
<param name=label1 value=1991>
<param name=label2 value=1992>
<param name=label3 value=1993>
<param name=label4 value=1994>
<param name=Columns value=4>
</applet>
*/

```



Section-D

Java AWT

Java AWT (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java.

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.

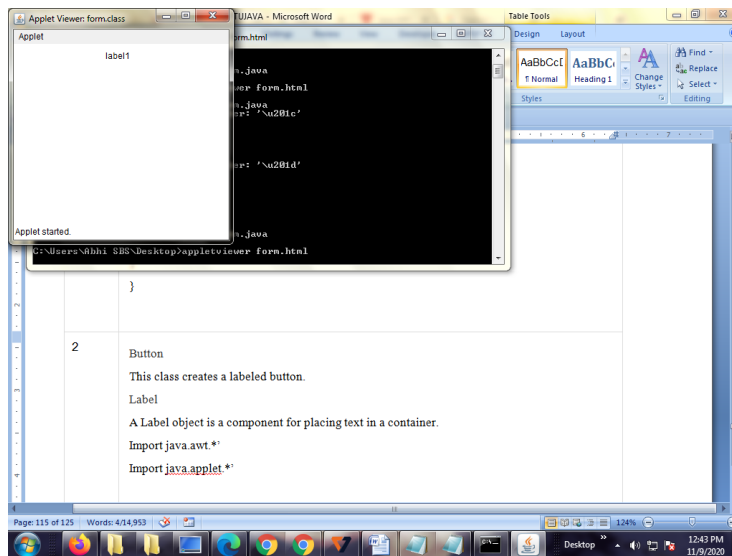
The `java.awt` package provides classes for AWT api such as `TextField`, `Label`, `TextArea`, `RadioButton`, `CheckBox`, `Choice`, `List` etc.

- **AWT UI Elements:**
- Following is the list of commonly used controls while designed GUI using AWT.

Sr. No.	Control & Description
1	<p>Label</p> <p>A Label object is a component for placing text in a container.</p> <pre>import java.awt.*; import java.applet.*;</pre>

class demo extends Applet

```
{  
    public void init()  
    {  
        Label lbl1=new Label("label1");  
        add(lbl1);  
    }  
}
```



2

Button

This class creates a labeled button.

Label

A Label object is a component for placing text in a container.

Import java.awt.*

Import java.applet.*

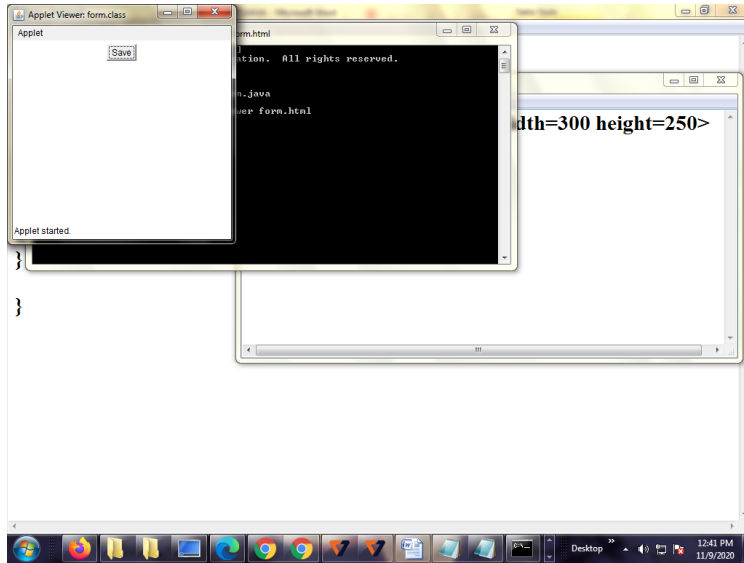
public Class demo extends Applet

```
{  
    Public void init()  
{
```

```

Button a=new Button("Save");
add(a);
}
}

```



3

Check Box

A check box is a graphical component that can be in either an **on** (true) or **off** (false) state.

Label

A Label object is a component for placing text in a container.

```
Import java.awt.*'
```

```
Import java.applet.*'
```

```
public Class demo extends Applet
```

```
{
```

```
Public void init()
```

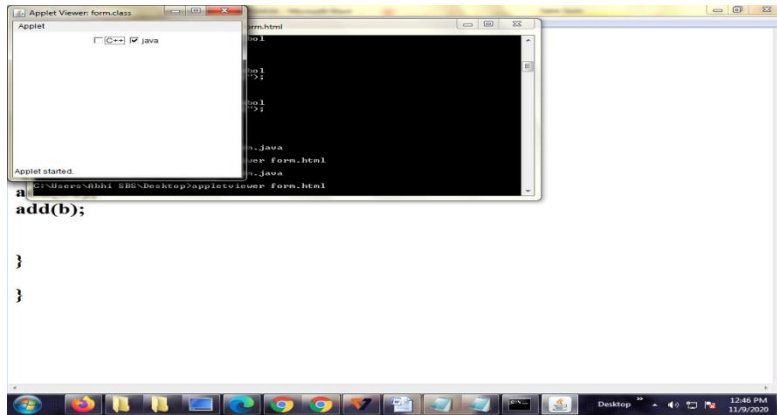
```
{
```

```
Checkbox b=new Checkbox("java",true);
```

```
add(b);
```

```
}
```

```
}
```



4

Check Box Group

The CheckboxGroup class is used to group the set of checkbox.

```

import java.awt.*;
public class CheckboxExample extends Applet
{
    void init()
    {
        Label lbl = new Label("Gender");
        CheckboxGroup s= new CheckboxGroup();
        Checkbox ChkMale = new Checkbox("Male",true,s);
        Checkbox Chkfemale = new Checkbox("Female",false,s);
        add(lbl);
        add(ChkMale);
        add(Chkfemale);
    }
}

```

5

List

The List component presents the user with a scrolling list of text items.

```

import java.awt.*;
public class ListExample extends Applet
{
    public void init()
    {
        Label lblLanguage = new Label("Choose the Language");
        List lstLanguage = new List(3,true);
        lstLanguage.add("1");
    }
}

```

```

        lstLanguage.add("2");
        lstLanguage.add("3");
        lstLanguage.add("C++");
        lstLanguage.add("java");
        add(lblLanguage); add(lstLanguage);

    }
}

```

6

Text Field

A TextField object is a text component that allows for the editing of a single line of text.

```

import java.awt.*;
public class ListExample extends Applet
{
    public void init()
    {

        Label lblLanguage = new Label("Choose the Language");
        List lstLanguage = new List(3,true);
        lstLanguage.add("pascal");
        lstLanguage.add("fortran");
        lstLanguage.add("C");
        lstLanguage.add("C++");
        lstLanguage.add("java");
        add(lblLanguage); add(lstLanguage);

    }
}

```

7

Text Area

A TextArea object is a text component that allows for the editing of a multiple lines of text.

```

import java.awt.*;
public class ListExample extends Applet
{
    public void init()
    {

        TextAreaExample(){

```

```
Frame f= new Frame();
    TextArea area=new TextArea("Welcome to javatpoint");
area.setBounds(10,30, 300,300);
f.add(area);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
    }
}
```

8

Text Field

A TextField object is a text component that allows for the editing of a single line of text.

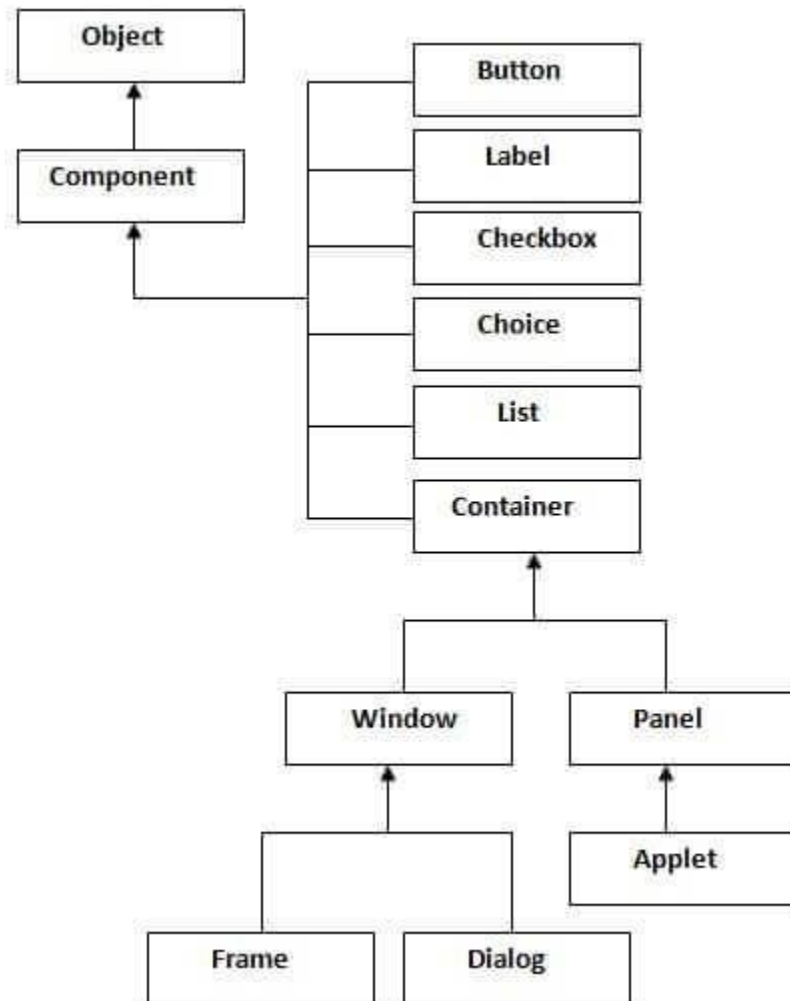
```
import java.awt.*;
public class ListExample extends Applet
{
    public void init()
    {

        Choice c=new Choice();
        c.setBounds(100,100, 75,75);
        c.add("Item 1");
        c.add("Item 2");
        c.add("Item 3");
        c.add("Item 4");
        c.add("Item 5");
        f.add(c);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);

    }
}
```

Java AWT Hierarchy

The hierarchy of Java AWT classes are given below.



Container

The Container is a component in AWT that can contain other components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

Window

The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

Panel

The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

Frame

The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

Delegation Event Model

Back in the old days, Java used a *Chain of Responsibility* pattern to process events. For example, when a button is clicked, a event is generated, which then is passed through a chain of components. The chain of components is defined by the hierarchy **of classes and interfaces**. An event is caught and handled by the handler class.

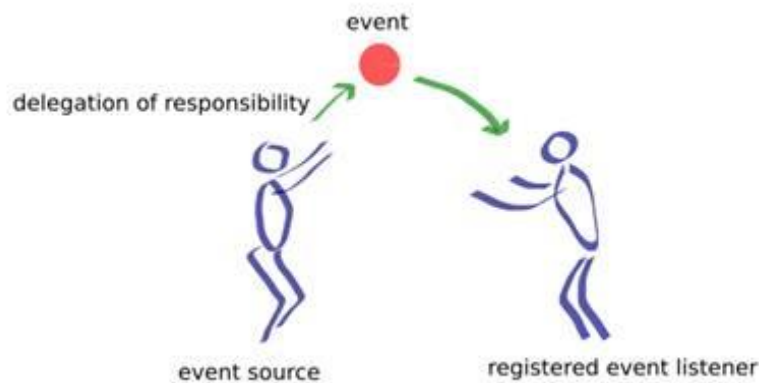
Events, Sources, and Listeners

The delegation event model can be defined by three components: event, event source, and event listeners.

- **Events:** The event object defines the change in state in the event source class. For example, interacting with the graphical interfaces, such as clicking a button or entering text via keyboard in a text box, item selection in a list, all represent some sort of change in the state. The event object is used to carry the required information about the state change. However, all events are not cause by user interaction. There are events such as timer event, hardware/software events, and so forth, that do not depend upon user interaction. They occur automatically. We can define the procedure to handle them once they occur.
- **Event sources:** Event sources are objects that cause the events to occur due to some change in the property of the component. Because there can be

various types a component can trigger, each must be registered to a listener to provide a suitable response.

- **Event listeners:** Event listeners are objects that are notified as soon as a specific event occurs. Event listeners must define the methods to process the notification they are interested to receive.



What is an Event?

Change in the state of an object is known as event i.e. event describes the change in state of source. Events are generated as result of user interaction with the graphical user interface components. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page are the activities that causes an event to happen.

Types of Event

The events can be broadly classified into two categories:

- **Foreground Events** - Those events which require the direct interaction of user.They are generated as consequences of a person interacting with the graphical components in Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through keyboard,selecting an item from list, scrolling the page etc.
- **Background Events** - Those events that require the interaction of end user are known as background events. Operating system interrupts, hardware or software failure, timer expires, an operation completion are the example of background events.

What is Event Handling?

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism have the code which is known as event handler that is executed when an event occurs. Java Uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events.Let's have a brief introduction to this model.

The Delegation Event Model has the following key participants namely:

- **Source** - The source is an object on which event occurs. Source is responsible for providing information of the occurred event to it's handler. Java provide as with classes for source object.
- **Listener** - It is also known as event handler.Listener is responsible for generating response to an event. From java implementation point of view the listener is also an object. Listener waits until it receives an event. Once the event is received , the listener process the event an then returns.

- **AWT Event Classes:**

- Following is the list of commonly used event classes.

Sr. No.	Control & Description
1	<p>AWTEvent</p> <p>It is the root event class for all AWT events. This class and its subclasses supercede the original java.awt.Event class.</p>
2	<p>ActionEvent</p> <p>The ActionEvent is generated when button is clicked or the item of a list is double clicked.</p>
3	<p>InputEvent</p> <p>The InputEvent class is root event class for all component-level input events.</p>
4	<p>KeyEvent</p>

	On entering the character the Key event is generated.
5	MouseEvent This event indicates a mouse action occurred in a component.
6	TextEvent The object of this class represents the text events.
7	WindowEvent The object of this class represents the change in state of a window.
8	AdjustmentEvent The object of this class represents the adjustment event emitted by Adjustable objects.
9	ComponentEvent The object of this class represents the change in state of a window.
10	ContainerEvent The object of this class represents the change in state of a window.
11	MouseEvent The object of this class represents the change in state of a window.

JAVA I/O HANDLING : I/O File Handling(InputStream &OutputStream,FileInputStream&FileOutputStream,DataI/PandO/P Streams,FileClass,ReaderandWriterStreams,RandomAccessFile).

File handling

File handling in Java implies reading from and writing data to a file. The File class from the java.io package, allows us to work with different formats of files. In order to use the File class, you need to create an object of the [class](#) and specify the filename or directory name.

```
// Import the File class
import java.io.File

// Specify the filename
File obj = new File("filename.txt");
```

What is a Stream?

In Java, Stream is a sequence of data which can be of two types.

1. Byte Stream

This mainly incorporates with byte data. When an input is provided and executed with byte data, then it is called the file handling process with a byte stream.

2. Character Stream

Character Stream is a stream which incorporates with characters. Processing of input data with character is called the file handling process with a character stream.

1. Create a File

In this case, to create a file you can use the *createNewFile()* method. This method returns true if the file was successfully created, and false if the file already exists.

Let's see an example of how to create a file in [Java](#).

```
package FileHandling;

import java.io.*;

public class CreateFile {
    public static void main(String[] args) {
        try
        {
            File myObj = new File("D:FileHandlingNewFilef1.txt");
        }
        catch (IOException e)
        {
            System.out.println("An error occurred.");
        }
    }
}
```

Output

```
1 File created: NewFile1.txt
```

2) Write to a File

In the following example, I have used the **FileWriter** class together with its **write()** method to write some text into the file. Let's understand this with the help of a code.

```
import java.io.*;

public class WriteToFile
{
public static void main(String[] args)
{
try {
FileWriter myWriter = new FileWriter("NewFile1.txt");
myWriter.write("hello java");

myWriter.close();
System.out.println("Successfully wrote to the file.");
}
catch (IOException e)
{
System.out.println("An error occurred.");
}
}
}
```

3. Read from a File

In the following example, I have used the Scanner class to read the contents of the text file.

```
package FileHandling;
```

```
// Import the File class
import java.io.File;
```

```
// Import this class to handle errors
import java.io.FileNotFoundException;
```

```
// Import the Scanner class to read text files
import java.util.Scanner;

public class ReadFromFile {
public static void main(String[] args) {
try {
// Creating an object of the file for reading the data
File myObj = new File("D:FileHandlingNewFilef1.txt");
Scanner myReader = new Scanner(myObj);
while (myReader.hasNextLine()) {
String data = myReader.nextLine();
System.out.println(data);
}
myReader.close();
} catch (FileNotFoundException e)
{
System.out.println("An error occurred.");

}
}
}
```