OOPS through JAVA

1. Java Fundamentals
2. Control Statements
3. String
4. Arrays
5. OOPs Concepts
6. Constructor
7. this object
8. Static members
9. Method Overloading
10. Inheritance
11. Method Overriding
12. final keyword
13. Dynamic Method Dispatching
14. Abstract class
15. Interface
16. Package
17. Exception Handling
18. Multi Threading
19. File Handling
20. AWT (Abstract Window Toolkit)

Prepared by,
M. Srinu, Tech. Trainer,
Technical Hub,Surampalem.

Types of languages:

High-level language:

The human being only understands it. Most of all high-level languages will use mediators. So execution time will be increased because of translation. They will not provide any direct interaction to the hardware so that searching time and storing time will be increased. But they are providing flexibility in their development.

Assembly language:

It is a symbolic language and these symbols are called mnemonics. The instructions directly interact with hardware or memory. Even though there is need of translation, it will take less time when compared to high – level language so it increase the execution speed.

Main drawback of **assembly language is platform dependency**. Whenever you write a program related to particular hardware or operating system. It will not work on another hardware or operating system so that there is a need to develop same program number of times on different platforms. It will increase the cost and time of development.

It is an expert –friendly language means it doesn't have any instructions for frequently used tasks so that we have to develop our own instructions according to our requirement. Since normal developers cannot develop programs by using this language.

Low-level language:

The machines only understand it. This is fully in ones and zeroes.

Reasons for developing java language:

In the case of C language, it supports procedure-oriented approach so that it is giving the freedom to the data so data can move to any program within the project.

It will not support overloading, reusing, encapsulation type of methodologies. It does not provide proper integration among different actions and properties. To overcome these drawbacks C++ was developed. It supports object oriented approach but it is not purely adapt to the oops because both the languages support pointers. Even though pointers will increase the execution speed and decrease the memory wastage, it doesn't provide security to the data. It means pointers violate security.

By considering all these drawbacks and make them advantages, java was developed in 1990 at sun Microsystems by James Goslings, Bill Joy, Patrick Naughton. Initially java was called as Oak. Its main area of application is in the field of micro controller. In these micro-controllers, we are able to develop common program for any type of micro controller. Once program is developed in java, it can be used any where so the caption for java is "**Write Once Run Any Where**"

From 1995 onwards, Internet was introduced so there is a need of distribution of data and application throughout the world on different types of computers and environments, so that java was promoted to Internet to satisfy the web application requirements. Because of platform independency, same java program can run on any type of environment. It makes java as powerful, secured, language in Internet era.

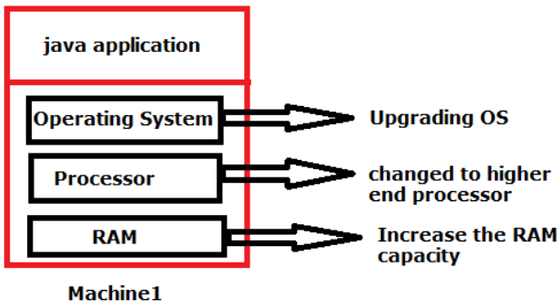
Java Buzzwords/Features

1) Simple:

- Java is very simple and easy to learn and its syntax is simple, clean and easy to understand.
- We can write java programs very easily.
- To learn java no prior knowledge is required.
- Most of the complex and confusing features of other languages (C, C++) like pointers, multiple inheritance… removed from java.
- Java is simple to develop for large-scale application like window based or Internet based.

2) Architectural Neutral:

It can run in any type of hardware architecture. Java Programs never communicates with the platform directly. Changes and upgrades in operating systems, processors and system resources will not force
any changes in java programs.



3) Portable:

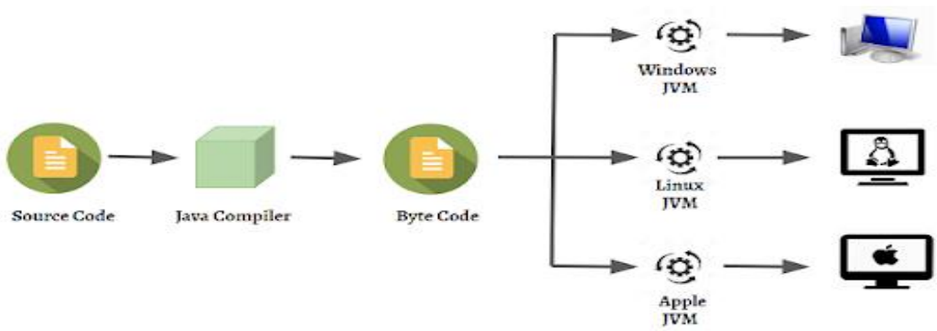
It can run on any type of operating system. We can carry the java byte code to any platform without making any changes.

Ex: Mobile number portability in India. i.e without changing the number moving from one service provider to other.

4) Platform Independent:

The program create and run in one operating System that can allowed to run in all other Operating Systems with out to do any changes in that code is called platform independent.

If we write java program once, we can run on any platform. i.e java follows Write Once Run Anywhere principle (**WORA**).



5) Robust:

In some cases, there is a change of some run time errors to occur in the case of program execution. They will bad to abnormal program termination, so that the data and application may be lost. Such type of errors is called exceptions.

If any language handling these exceptions in a systematic way then those languages will be treated as strictly written languages.

Any application developed by using these types of languages is called robust applications. They will provide safety and security to applications they will provide safety and security to application and data

6) Object Oriented:

Java is Object Oriented Programming language like C++. Everything in Java is an Object. Object Oriented means we organize our software as a combination of different types of objects that incorporates both data and behaviour. It supports all the concepts oops such as data abstraction, encapsulation, polymorphism, and inheritance.

7) Secure:

Java program never communicates directly with the machine. First converted into byte code and then converted into machine code by JVM. If the byte code contains any problem or if it is not properly formatted, then JVM won't allow that code to run and will raise VerifyError. Internally inside
JVM, Byte code verifier is responsible to verify the byte code.
Hence java programs won't cause any problem to the system.

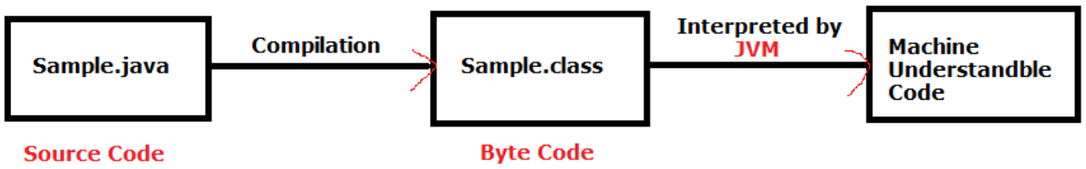
8) Dynamic:

In java, every object will be allocated dynamic to reduce the memory wastage. In the case of Internet applications, reduction in the memory wastage will reduce the transmission time and increases the down loading speed and transmission speed.

9) Compiled and Interpreted:

In any language, language translation is handled by the compilers. But in the case of java, program is first compiled by the java compiler. Then it will generate a code called byte code.

This byte code is a universal code, which can be understood, by any type of JVMs (java virtual machine) so it will interpret that byte code according to the platform on which it was installed. This feature makes the program as platform independent. But JVMs are platform dependant.



10) Distributed:

If the application is distributed across multiple machines (JVM's), such type of application is called Distributed Application. Java Provides inbuilt support for Distributed Programming with RMI, CORBA and EJB.

This feature of Java makes us able to access files by calling the methods from any machine on the internet.

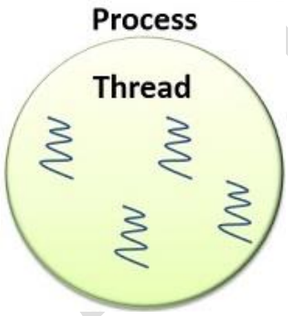
11) High performance:

Java is relatively faster than traditional interpreted languages, since byte code is "close" to native code. But Java is still somewhat slower than C or C++.

12) _Multi-Threaded:

Executing several individual parts of a same program simultaneously is called multi threading. Each part of such program is called a thread. So, threads are light-weight processes within a process.

It will increase the utilization of CPU.



JAVA PROGRAM STRUCTURE:

Components of java programmed:

1. Package declaration part
2. Package import statement
3. Class definitions.

class syntax:

```
class <class-name>
{                              data
        Private members –      methods
        Public members
        Protected members
        No modifier members
}
```

class contains following types of access specifies.

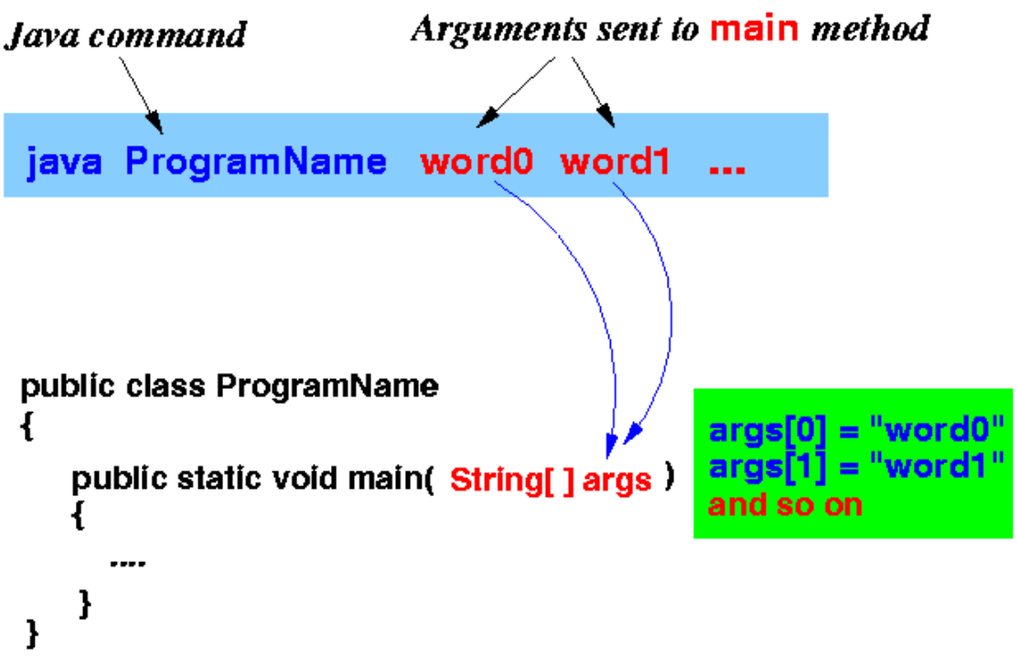
1. **private:**
   These members are accessible only with in the class.
2. **public:**
   These members can be accessed anywhere.
3. **protected:**
   These members can be accessed with in the class and within its child classes of same package and from other packages also.
4. **No modifier:**
   In Java, if you do not specify any access specified it will not be treated as private. The scope of these variables is within the package

```
class Sample
{
        public static void main (String args[])
        {
                System.out.println ("Java Demo");
        }
}
```

Save: Sample.java
Compile: D:\Programs>javac Sample. Java
Run: D:\Programs>java Sample

Commandline Arguments:

The arguments which are passing from command prompt at the time program execution are called command line arguments.
The main objective of command line arguments are we can customize the behaviour of the main().



```
class Sample
{
        public static void main (String args [])
        {
                System.out.println ("Total Args:"+args.length);
                System.out.println ("Given Args");
                for (int i=0; i<args.length;i++)
                        System.out.println (args[i]);
        }
}
```

Compile: javac Sample.java
Run: java Sample 10 20 30 40
Expected Output:

```
Total Args:4
Given Args
10
20
30
40
```

"+" Operator overloading used to concatenate the value to string.
Static -> depends on class
Normal -> depends on objects

Example:

```
class Sample
{
        public static void main (String args[])
        {
                int x=Integer.parseInt(arg[0]);      //Converts Given String to  integer type.
                int y=integer. parseInt (args[i]);
                int z=x+y;
                System.out.println ("Sum:"+z);
        }
}
```

Run c:\> java Sample 10 20
Sum: 30

//To add 2 numbers by taking I/p from keyboard

```
import java.io.*;
class NumberDemo
{
        public static void main (String args []) throws Exception
        {
                InputStreamReader isr=new BufferedReader(System.in)
                BufferedReader br=new BufferedReader (isr);
                int x,y,z;
                System .out.println ("enter 2 no.s:");
                x = Integer.parseInt (br.readLine ());
                y = Integer.parseInt (br.readLine());
                z = x+y;
                System.out.println ("Sum:"+z);
        }
}
```

//To calculate simple interest:

```
import java.io. *;
class Interest
{
   public static void main (String args[]) throws Exception
   {
        InputStreamReader isr=new InputStreamReader (System.in);
        BufferedReader b=new BufferedReader (isr);
        int t,p,r;
        float si;
```

```
                    System.out.println ('Enter principle:");
                    p = inter.parseint(b.readLine ());
                    System.out.println("Enter Time in Months");
                    t = Integer.parseInt (b.readLine());
                    System.out.println("Enter Rate of Interest");
                    r = Integer.parseInt (b.readLine());
                    si=(float)(p*t*r)/100;
                    System.out.println("Simple Interest: "+si);
                }
            }
```

readLine() -> function is available in java.io.BufferedReader class

In the System.out.println()

System is a class in java.lang package.

"out" is an Object to the PrintStream.

```
// Program to find the Roots of the Quadratic Equation
import java.io.*;
class Roots
{
  public static void main(String[] args)throws Exception
      {
          InputStreamReader isr=new InputStreamReader(System.in);
           BufferedReader br=new BufferedReader(isr);
           // Declaration of variable a, b, and c
          double a,b,c,root1, root2;
          a=Double.parseDouble(br.readLine());
          b=Double.parseDouble(br.readLine());
          c=Double.parseDouble(br.readLine());
          // calculate the determinant (b2 - 4ac)
          double determinant = b * b - 4 * a * c;
          // check if determinant is greater than 0
          if (determinant > 0)
          {       // two real and distinct roots
                  root1 = (-b + Math.sqrt(determinant)) / (2 * a);
                  root2 = (-b - Math.sqrt(determinant)) / (2 * a);
                  System.out.printf("root1 = %.2f and root2 = %.2f", root1, root2);
          }
          // check if determinant is equal to 0
          else if (determinant == 0)
          {
                  // two roots are real and equal if determinant is equal to 0
                  // so -b + 0 == -b
                  root1 = root2 = -b / (2 * a);
                  System.out.format("root1 = root2 = %.2f", root1);
          }
          // if determinant is less than zero
          else {
                  // roots are complex number and distinct
                  double real = -b / (2 * a);
                  double imaginary = Math.sqrt(-determinant) / (2 * a);
                  System.out.printf("root1 = %.2f+%.2fi", real, imaginary);
                  System.out.format("\nroot2 = %.2f-%.2fi", real, imaginary);
              }
      }
}
```

Output:

```
D:\ECE-A>java Roots
Enter a, b and c values
2.3
4
5.6
root1 = -0.87+1.30i
root2 = -0.87-1.30i
```

JAVA Fundamentals:

Java Environment Setup on Windows

- Java SE is freely available from the internet resources. So you download a version based on your operating system.
- Follow the instructions to download java and run the .exe to install Java on your machine.
- Once you installed java on your machine, you would need to set environment variables to point to correct installation directories.
- Assuming you have installed Java in C:\ProgramFiles\java\jdk Directory.
- Right click on 'My Computer' and select Properties.
- Click on the Environment variables button under the Advanced Tab.
- Now, after add the path 'C:\Program Files \Java\jdk1.7.0_03\bin' in Path variable.

Java Basic Building Blocks:

1. Identifiers
2. Reserved Words
3. Data Types
4. Literals
5. Variable declaration and its scope
6. Command-line Arguments
7. Java Coding Standards
8. Operators and its types
9. Control Statements

1. Identifier:

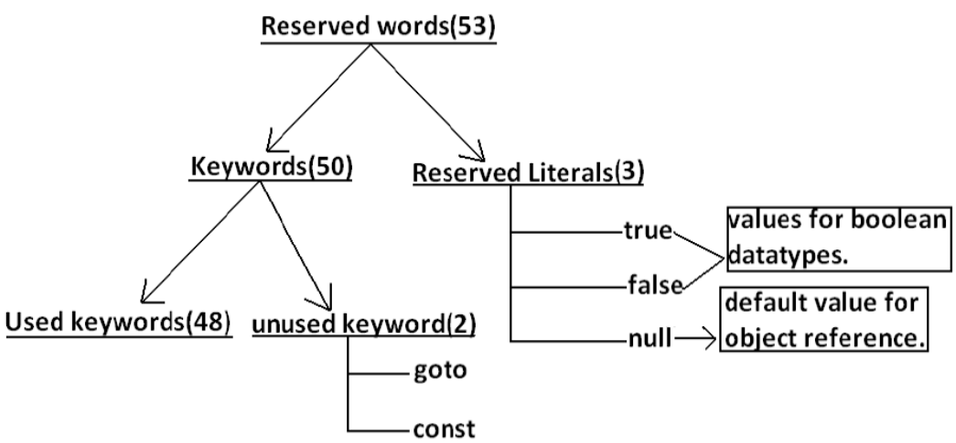
A name in Java Program is called identifier, it can be class name or variable name or method name etc.

Rules to an Identifier:

- The only allowed characters in Java Identifiers are
  a - z
  A - Z
  0 – 9
  _ and $
- Identifier can't starts with digit
- Java Identifiers are Case Sensitive.
- There is no length limit for java identifiers, but it's not recommended to take more than 15 length (>15).
- Reserved words can't be used as identifiers.
- All predefined Java class names and interface names we can use as identifiers even though it is legal but it is not recommended.

2. Reserved Words

- Reserved words which are having predefined meaning in Java.
- All the keywords are defined in lower case and the meaning of these keywords can't be modified.
- We cannot use keywords as names for variables, classes, methods, or as any other identifiers.

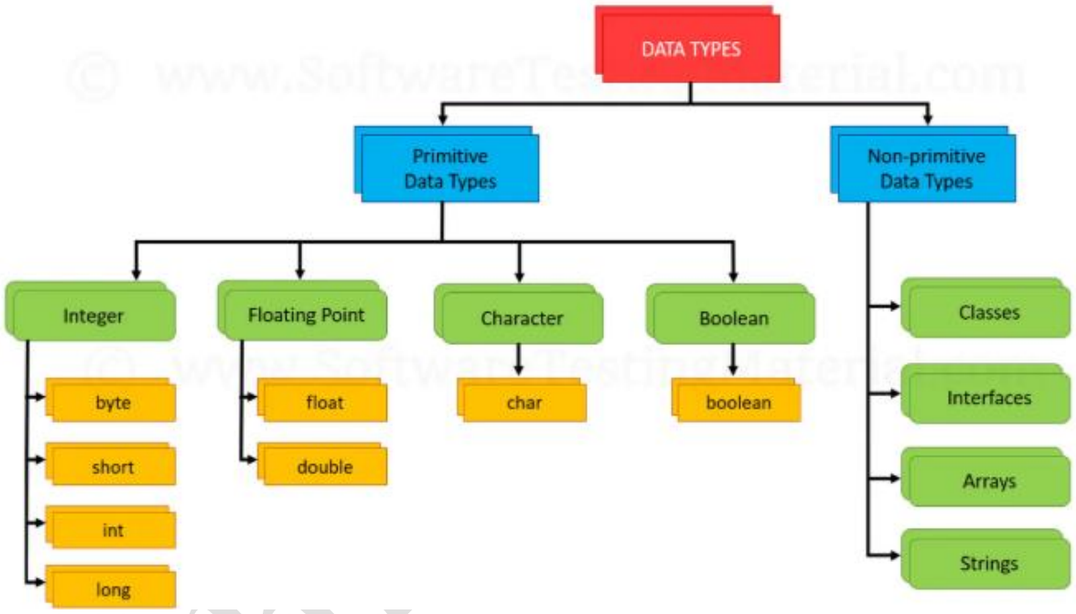


Keywords In Java

1. abstract
2. assert
3. boolean
4. break
5. byte
6. case
7. catch
8. char
9. class
10. continue
11. default
12. do
13. double
14. else
15. enum
16. extends
17. final
18. finally
19. float
20. for
21. if
22. implements
23. import
24. instanceof
25. int
26. interface
27. long
28. native
29. new
30. package
31. private
32. protected
33. public
34. return
35. short
36. static
37. strictfp
38. super
39. switch
40. synchronized
41. this
42. throw
43. throws
44. transient
45. try
46. void
47. volatile
48. while

3. Datatypes

- Data types specify the size and type of values that can be stored.
- Java is language rich in the data types.
- Every variable in java has a data type.



Program to find the size of different data types used in java

```
class SizeofdataTypes
{
  public static void main (String[] args)
  {
        System.out.println("Size of byte: "+(Byte.BYTES) + " bytes.");
        System.out.println("Size of short: "+(Short.BYTES) + " bytes.");
        System.out.println("Size of int: "+(Integer.BYTES) + " bytes.");
        System.out.println("Size of long: "+(Long.BYTES) + " bytes.");
        System.out.println("Size of char: "+(Character.BYTES) + " bytes.");
        System.out.println("Size of float: "+(Float.BYTES) + " bytes.");
        System.out.println("Size of double: "+(Double.BYTES) + " bytes.");
   }
}
```

Output:

```
D:\JAVA-Programs>java SizeofdataTypes
Size of byte: 1 bytes.
Size of short: 2 bytes.
Size of int: 4 bytes.
Size of long: 8 bytes.
Size of char: 2 bytes.
Size of float: 4 bytes.
Size of double: 8 bytes.
```

4. Literals

A constant value which can be assign to a variable is called a "Literal";

Ex: int x = 10;

Integeral Literals:

1) **Decimal Literals**: allowed digits are 0 to 9.
2) **Octal literals:** allowed digits are 0 to 7 and literal value should be prefixed with "0".
3) **Hexadecimal literal:** allowed digits are 0 to 9, a-f or A-F and literal value should be prefixed with either 0x or 0X.

```
Ex: int x=10;
    int y=010;
    int z=0X10;
```

By default every integral literal is of int type but we can specify explicitly as long type by suffixing with l or L.

Floating Point Literal:

Every floating point literal is by default double type and hence we can't assign directly to float variable.

But we can specify explicitly floating point literal is the float type by suffixing with 'f' or 'F'.

Ex: float pi=3.1424f;

Boolean Literal:

The only possible values for the boolean data type are true/false.

Ex: boolean b=true;

Character Literal:

A char literal can be represented as single character with single quotes.

Ex: char ch='a';
char ch='8';

A char literal can be represented in unicode representation which nothing but \uxxxx. It is a 4 digit hexa decimal number.

Ex: char ch='\u0061';
System.out.println(ch); // a

String Literal:

Any sequence of characters with in double quotes("") is called String literal.

Ex: String s="java";
String s="aditya";

5. Variable and its scope

Variable: It is name of a memory location where the actual value is to be stored.

Declaration does three things:

1. It tells the compiler what the variable name is.
2. It specifies what type of data the variable will hold.
3. The place of declaration declares the scope of the variable.

A variable must be declared before it is used in the program.

The general form of declaration of variable is:

```
datatype variable1, variable2,....., variableN;
```

Types of Variables:

1. Local variables
2. Instance variables
3. Class variables

6. Commandline Arguments

The arguments which are passing from command prompt at the time program execution are called command line arguments. The main objective of command line arguments are we can customize the behaviour of the main().

7. Java Coding Standards

Whenever we are writing the code it is highly recommended to follow Coding Conventions. The name of the method or class should reflect the purpose of functionality of that component.

i) Usually class or interface names are start with Upper Case Letters and if it contains multiple words every inner word should starts with Uppercase Letter.

Ex: Student, Employee, String, StringBuffer, Runnable etc.

ii) Usually method names are should starts with lower case letter and if it contains multiple words, every inner word should starts with Upper Case Letter.

```
Ex:  run();          getName();
     sleep();        setName();
     init();         getSalary();
     wait();         addActionListener(); etc..
```

iii) Usually the variable names are should starts with lower case letter and if it contains multiple

words, every inner word should starts with Upper Case Letter.
Ex: name, rollno, mobileNumber, empSal.....
iv) Usually the constants names are should contain only Upper Case Characters, if it contains multiple words, these words are separated with '_' symbol.
We can declare constants by using final modifiers.
Ex: MAX_VALUE
MIN_VALUE
MAX_PRIORITY
MIN_PRIORITY etc.....

8. Operators and its types

Operators is a symbol used to perform an operation. Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups

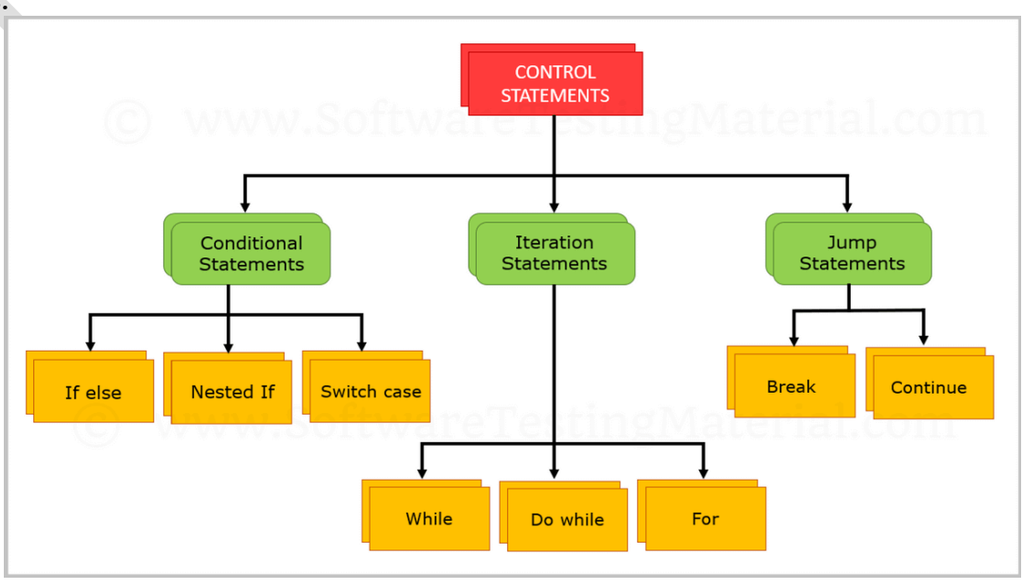
- Arithmetic Operators **(+, -, *, /, %, ++, --)**
- Relational Operators **(==, !=, >, <, >=, <=)**
- Bitwise Operators **(&, |, ^, ~, <<, >>, >>>)**
- Logical Operators **(&&, ||, !)**
- Assignment Operators **(=, +=, -=, *=, /=, %=, &=, ^=, |=)**
- Misc Operators **(?:, instanceof)**

Operator Precedence and Associativity:
The **precedence of operators** determines which operator is executed first if there is more than one operator in an expression.
For example, x = 7 + 3 * 2; here x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with 3 * 2 and then adds into 7.
If an expression contains more than one operator having same precedence then **associativity** (i.e either left -> right or right -> left) rule should applied on the expression.

Category	Operator	Associativity
Postfix	expression++ expression--	Left to right
Unary	++expression –-expression +expression –expression ~ !	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >> >>>	Left to right
Relational	< > <= >= instanceof	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= ^= |= <<= >>= >>>=	Right to left

9. Control Statements

Control statements are ways for a programmer to control what pieces of the program are to be executed at certain times. **Or**
Statements which are used to control the flow of execution of a program are called Control Statements.



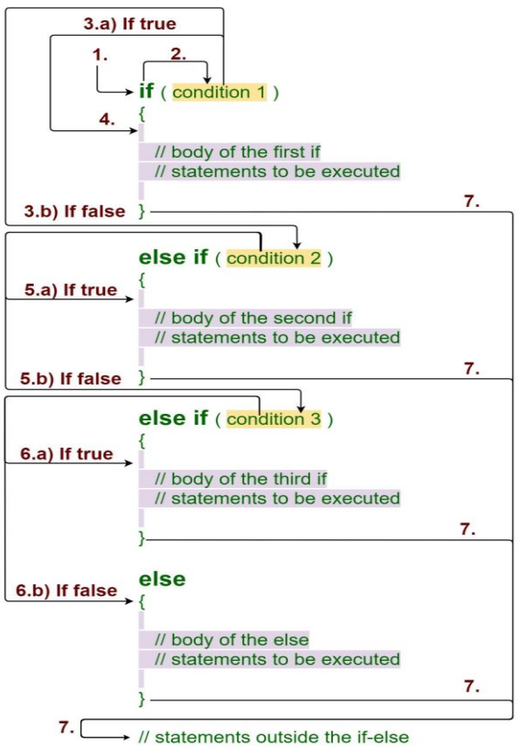
if...else

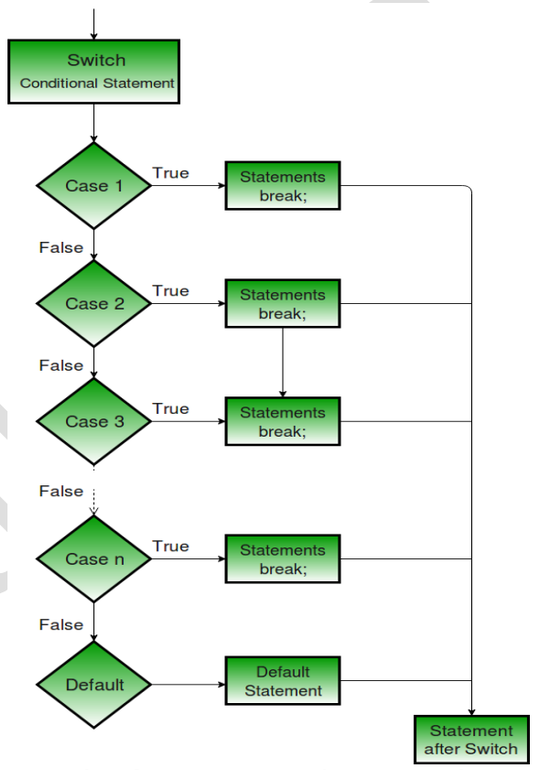
```
if(condtion)
 {
    //true block
 }
else
 {
  //false block
 }
```

Nested if

```
if(condition1)
{
  if(condition2)
   {
     //true block
   }
  else
   {
     //false block
   }
}
else.....
```

If - else if ladder

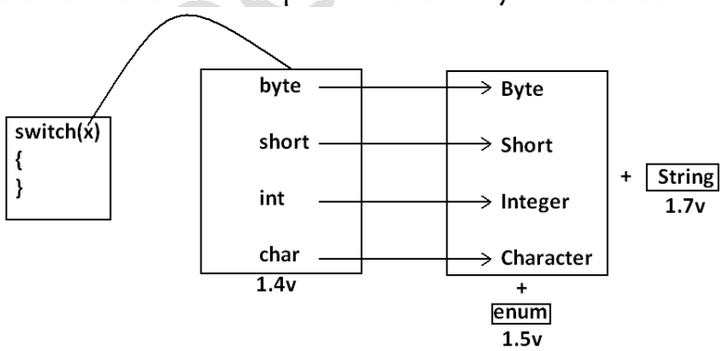




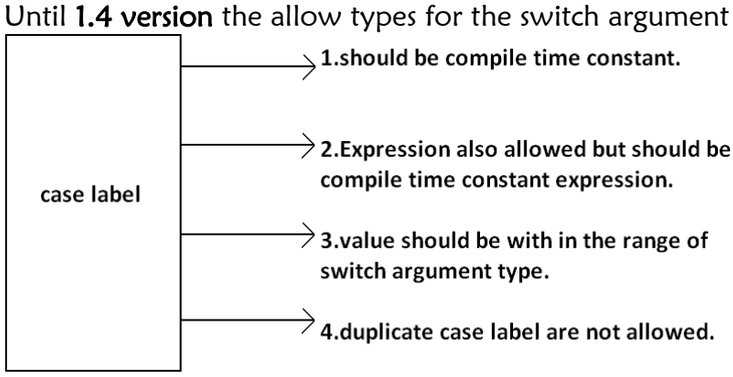
The argument to the if statement should be boolean by mistake if we are providing any other type we will get **"compile time error".**

switch():

If several options are available then it is not recommended to use if-else we should go for switch statement. Because it improves readability of the code.



Until **1.4 version** the allow types for the switch argument are byte, short, char, int.



Fall-through inside the switch:

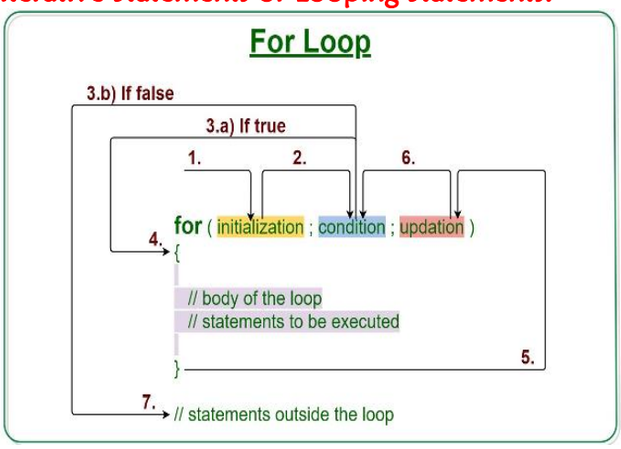
With in the switch statement if any case is matched from that case onwards all statements will be executed until end of the switch (or) break. This is call "fall through" inside the switch .

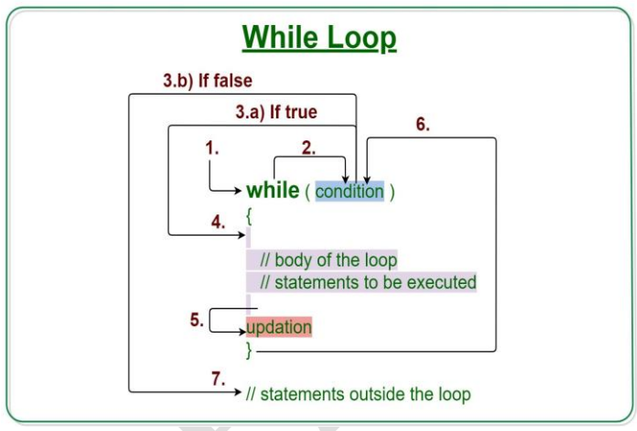
The main advantage of fall-through inside a switch is we can define common action for multiple cases

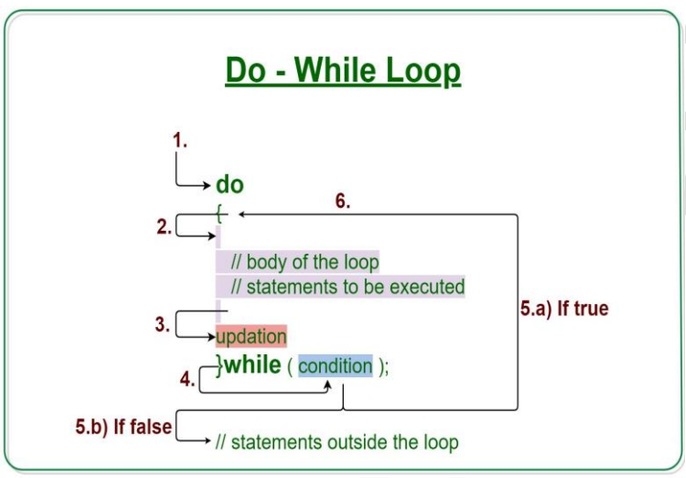
Default Case:

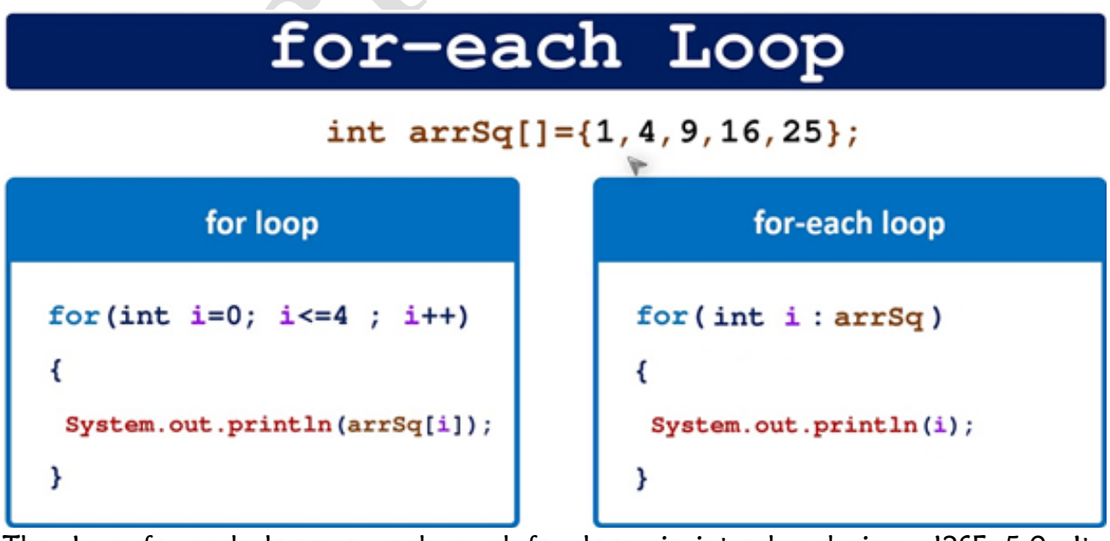
- With in the switch we can take the default only once.
- If no other case matched then only default case will be executed
- With in the switch we can take the default any where, but it is conventions to take default as last case.

Iterative Statements or Looping Statements:









The Java for-each loop or enhanced for loop is introduced since J2SE 5.0. It provides an alternative approach to traverse the array or collection in Java. **The advantage** of the for-each loop is that it eliminates the possibility of bugs and makes the code more readable. It is known as the for-each loop because it traverses each element one by one.

The drawback of the enhanced for loop is that it cannot traverse the elements in reverse order. Here, you do not have the option to skip any element because e it does not work on an index basis.

Practice Programs

Program - 1:

Write a Java Program to find the simple interest.

Source Code:

```
import java.util.*;
class SimpleInterest
{
  public static void main(String args[])
   {
      int P,T;
      float R,I;
      Scanner sc=new Scanner(System.in);
      System.out.println("Enter Principle, Time and Rate of Interest");
      P=sc.nextInt();
      T=sc.nextInt();
      R=sc.nextFloat();
      I=(P*T*R)/100;
      System.out.println("PRINCIPLE AMOUNT : "+P);
      System.out.println("TIME : "+T);
      System.out.println("RATE OF INTEREST : "+R);
      System.out.println("SIMPLE INTEREST : "+I);
   }
}
```

Output:

```
java SimpleInterest
Enter Principle, Time and Rate of Interest
100000
14
2.6
PRINCIPLE AMOUNT : 100000
TIME : 14
RATE OF INTEREST : 2.6
SIMPLE INTEREST : 36399.996
```

Program – 2:

Write a Java program to demonstrate the reading of input from the user using io classes.

Source Code:

```
import java.util.*;
class Student_Info
{
    public static void main(String args[])
    {
        Scanner sc=new Scanner(System.in);
                String id,name,branch;
                int age;
                float height;
                char gender;
                long mobile;
                boolean married;
          System.out.println("Enter student id,name,branch,gender, age,hegiht, married status and mobile number");
                id=sc.next();
                sc.nextLine();
                name=sc.nextLine();
                branch=sc.nextLine();
                gender=sc.next().charAt(0);
                age=sc.nextInt();
                height=sc.nextFloat();
                married=sc.nextBoolean();
                mobile=sc.nextLong();
                 System.out.println("=====================================");
                System.out.println("STUDENT INFORMATION:");
                 System.out.println("=====================================");
                System.out.println("ID ="+id);
```

```
                System.out.println("NAME ="+name);
                System.out.println("BRANCH ="+branch);
                System.out.println("GENDER ="+gender);
                System.out.println("AGE ="+age);
                System.out.println("HEIGHT ="+height);
                System.out.println("MARRIED STATUS ="+married);
                System.out.println("MOBILE NUMBER ="+mobile);
                System.out.println("=====================================");
        }
}
```

Program – 3:

Write a Java program to read and display employee information.

Source Code:

```
import java.util.*;
class Employee
{
   public static void main(String args[])
   {
     int empid;
     String empname,desg,company;
     float salary;
     Scanner sc=new Scanner(System.in);
     System.out.println("Enter an Employee Information");
     System.out.println("Enter Employee ID:");
     empid=sc.nextInt();
     sc.nextLine();
     System.out.println("Enter Employee Name");
     empname=sc.nextLine();
     System.out.println("Enter Employee Desg");
     desg=sc.nextLine();
     System.out.println("Enter Employee Salary");
     salary=sc.nextFloat();
     sc.nextLine();
     System.out.println("Enter Company Name");
     company=sc.nextLine();
     System.out.println("================================");
     System.out.println("Employee Information");
     System.out.println("EMPLOYEE ID: "+empid);
     System.out.println("EMPLOYEE NAME: "+empname);
     System.out.println("EMPLOYEE DESG: "+desg);
     System.out.println("EMPLOYEE SALARY: "+salary);
     System.out.println("COMPANY NAME: "+company);
     System.out.println("================================");
   }
}
```

Output:

```
Java Employee
Enter an Employee Information
Enter Employee ID:
1111
Enter Employee Name
Ramesh K
Enter Employee Desg
Assistant Professor
Enter Employee Salary
35000
Enter Company Name
Aditya Engineering College
================================
Employee Information
EMPLOYEE ID: 1111
EMPLOYEE NAME: Ramesh K
```

```
EMPLOYEE DESG: Assistant Professor
EMPLOYEE SALARY: 35000.0
COMPANY NAME: Aditya Engineering College
==================================
```

Program - 4:

Write a Program to find the biggest of two numbers.

Source Code:

```
import java.io.*;
class Biggest_Number
{
	public static void main(String args[])throws Exception
	{
		Scanner sc=new Scanner(System.in);
		int n1,n2;
		System.out.println("Enter any two numbers");
		n1=sc.nextInt();
		n2=sc.nextInt();
		if(n1>n2)
			System.out.println(n1+" is bigger than "+n2);
		else
			System.out.println(n2+" is bigger than "+n1);
	}
}
```

Save: Biggest_Number.java
Compilation: javac Biggest_Number.java
Run: java Biggest_Number
Output:

```
D:\Java_Programs>java Biggest_Number
Enter any two numbers
25
14
25 is bigger than 14
```

Program -5:

Write a Program find who is best hero based on given input?

Input: Chiru
67
52
27
Balayya
45
33
26

Note: Take the points as follows, 1 hit movie= 10 points, 1 average movie=5, a flop movie= -5

Source Code:

```
import java.util.*;
class Best_Hero
{
	public static void main(String args[])throws Exception
	{
		Scanner sc=new Scanner(System.in);
		String hero1_Name,hero2_Name;
		int hero1_hit,hero2_hit,hero1_avg,hero2_avg,hero1_flop,hero2_flop,hero1_Tot, hero2_Tot;

		System.out.println("Enter hero1 details");
		hero1_Name=sc.nextLine();
		hero1_hit=sc.nextInt();
		hero1_avg=sc.nextInt();
		hero1_flop= sc.nextInt();

		System.out.println("Enter hero2 details");
		hero2_Name= sc.nextLine();
```

```
            hero2_hit= sc.nextInt();
            hero2_avg= sc.nextInt();
            hero2_flop= sc.nextInt();
            hero1_Tot=hero1_hit*10+hero1_avg*5+hero1_flop*-5;
            hero2_Tot=hero2_hit*10+hero2_avg*5+hero2_flop*-5;

            if(hero1_Tot>hero2_Tot)
              System.out.println(hero1_Name+" is the best Hero with "+hero1_Tot+" points");
            else if(hero1_Tot<hero2_Tot)
             System.out.println(hero2_Name+" is the best Hero with "+hero2_Tot+" points");
            else
              System.out.println(hero1_Name+" and "+hero2_Name+" are having equal points");
        }
}
```

Output:

```
D:\Java_Programs>java Best_Hero
Enter hero1 details
Chiru
56
35
45
Enter hero2 details
Balayya
48
33
32
Chiru is the best Hero with 510 points
```

Program – 6:

Five Bikers Compete in a race such that they drive at a constant speed which may or may not be the same as the other. To qualify the race, the speed of a racer must be more than the average speed of all 5 racers.

Take as input the speed of each racer and print back the speed of qualifying racers.

Source Code:

```
import java.io.*;
class Bike_Racers
{
        public static void main(String args[])throws Exception
        {
                BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
                int racer1_Speed,racer2_Speed,racer3_Speed,racer4_Speed,racer5_Speed;
                int sum;
                float avg_Speed;
                System.out.println("Enter 5 Bike Racers Speeds");
                racer1_Speed=Integer.parseInt(br.readLine());
                racer2_Speed=Integer.parseInt(br.readLine());
                racer3_Speed=Integer.parseInt(br.readLine());
                racer4_Speed=Integer.parseInt(br.readLine());
                racer5_Speed=Integer.parseInt(br.readLine());
                sum=racer1_Speed+racer2_Speed+racer3_Speed+racer4_Speed+racer5_Speed;
                avg_Speed=(float)sum/5;
                System.out.println("Average Speed is:"+avg_Speed);
                System.out.println("The Qualified Racers are:");
                if(racer1_Speed>avg_Speed)
                        System.out.println("Racer1");
                if(racer2_Speed>avg_Speed)
                        System.out.println("Racer2");
                if(racer3_Speed>avg_Speed)
                        System.out.println("Racer3");
                if(racer4_Speed>avg_Speed)
                        System.out.println("Racer4");
                if(racer5_Speed>avg_Speed)
                        System.out.println("Racer5");
        }
}
```

Output:

```
D:\Java_Programs>javac Bike_Racers.java

D:\Java_Programs>java Bike_Racers
Enter 5 Bike Racers Speeds
124
132
115
117
123
Average Speed is:122.2
The Qualified Racers are:
Racer1
Racer2
Racer5
```

Program – 7:

Write a program to read temperature in centigrade and display a suitable message according to temperature state below :

Temp < 0 then Freezing weather
Temp 0-10 then Very Cold weather
Temp 10-20 then Cold weather
Temp 20-30 then Normal in Temp
Temp 30-40 then Its Hot
Temp >=40 then Its Very Hot

Source Code:

```
import java.io.*;
class Temperature
{
	public static void main(String args[])throws Exception
	{
		BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
		int temp;
		System.out.println("Enter the value for temperature");
		temp=Integer.parseInt(br.readLine());

		if(temp<0)
			System.out.println("Freezing weather");
		else if(temp>=0 && temp<10)
			System.out.println("Very Cold weather");
		else if(temp>=10 && temp<20)
			System.out.println("Cold weather");
		else if(temp>=20 && temp<30)
			System.out.println("Normal in Temperature");
		else if(temp>=30 && temp<40)
			System.out.println("Hot");
		else
			System.out.println("Very Hot");
	}
}
```

Output:

```
D:\Java_Programs>javac Temperature.java

D:\Java_Programs>java Temperature
Enter the value for temperature
45
Very Hot

D:\Java_Programs>java Temperature
Enter the value for temperature
19
Cold weather
```

Program – 8:

Write a program to display the given digit(0 to 9) in words as follows

Input: 9

Output: Nine

Source Code:

```
import java.io.*;
class Digit_Word
{
	public static void main(String args[])throws Exception
	{
		BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
		int num;
		System.out.println("Enter a digit (0-9)");
		num=Integer.parseInt(br.readLine());

		if(num==0)
			System.out.println("Zero");
		else if(num==1)
			System.out.println("One");
		else if(num==2)
			System.out.println("Two");
		else if(num==3)
			System.out.println("Three");
		else if(num==4)
			System.out.println("Four");
		else if(num==5)
			System.out.println("Five");
		else if(num==6)
			System.out.println("Six");
		else if(num==7)
			System.out.println("Seven");
		else if(num==8)
			System.out.println("Eight");
		else if(num==9)
			System.out.println("Nine");
		else
			System.out.println("Enter a valid digit");
	}
}
```

Output:

```
D:\Java_Programs>java Digit_Word
Enter a digit (0-9)
8
Eight

D:\Java_Programs>java Digit_Word
Enter a digit (0-9)
12
Enter a valid digit
```

Program – 9:

Write a program to display Colour name by taking input as colour code.

For example:

Y -> Yellow
R -> Red
G -> Green
B -> Blue
V -> Violet
O -> Orange
I -> Indigo

Source Code:

```
import java.io.*;
class Color_Codes
{
	public static void main(String args[])throws Exception
	{
		BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
		char code;
		System.out.println("Enter a Color Code");
```

```
                code=(char)br.read();
                switch(code)
                {
                        case 'R':
                                        System.out.println("Red");
                                        break;
                        case 'G':
                                        System.out.println("Green");
                                        break;
                        case 'B':
                                        System.out.println("Blue");
                                        break;
                        case 'Y':
                                        System.out.println("Yellow");
                                        break;
                        case 'V':
                                        System.out.println("Violet");
                                        break;
                        case 'O':
                                        System.out.println("Orange");
                                        break;
                        case 'I':
                                        System.out.println("Indigo");
                                        break;
                        default:
                                        System.out.println("Enter valid Color Code");
                                        break;
                }
        }
}
```

Output:

```
D:\Java_Programs>javac Color_Codes.java

D:\Java_Programs>java Color_Codes
Enter a Color Code
G
Green
```

Program – 10:

Write a program to display season by taking the input as month.

Assume: 3, 4, 5 – Summer
6, 7, 8, 9 – Rainy
10, 11, 12, 1, 2 - Winter

Input: 9
Output: Rainy
Source Code:

```
import java.io.*;
class Seasons
{
        public static void main(String args[])throws Exception
        {
                BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
                int month;
                System.out.println("Enter a month number");
                month=Integer.parseInt(br.readLine());
                switch(month)
                {
                        case 3:
                        case 4:
                        case 5:
                                System.out.println("Summer");
```

```
                        break;
                case 6:
                case 7:
                case 8:
                case 9:
                        System.out.println("Rainy");
                        break;
                case 10:
                case 11:
                case 12:
                case 1:
                case 2:
                        System.out.println("Winter");
                        break;
                default:
                        System.out.println("Enter the valid month number");
                        break;
            }
        }
}
```

Output:

```
D:\Java_Programs>java Seasons
Enter a month number
10
Winter
```

Iterative (or) Looping Statements Example

Program – 11:

Write a program to find the factors of a given number?

Input: 6
Output: 1 2 3 6

Source Code:

```
import java.io.*;
class Factors
{
        public static void main(String args[])throws Exception
        {
                BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
                int num, i;

                System.out.println("Enter a number to find factors of it");
                num=Integer.parseInt(br.readLine());

                System.out.println("Factors of "+num+" are:");
                for(i=1;i<=num;i++)
                {
                        if(num%i==0)
                                System.out.print(i+" ");
                }
        }
}
```

Output:

```
D:\Java_Programs>javac Factors.java
D:\Java_Programs>java Factors
Enter a number to find factors of it
24
Factors of 24 are:
1 2 3 4 6 8 12 24
```

Program – 12:

Write a program to find the given number is prime or not?

Input: 11

Output: Prime Number

Source Code:

```
import java.io.*;
class Prime_Numbers
{
	public static void main(String args[])throws Exception
	{
		BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
		int num,i,count=0;
		System.out.println("Enter a number to find it is prime or not");
		num=Integer.parseInt(br.readLine());

		for(i=1;i<=num;i++)
		{
			if(num%i==0)
				count++;
		}
		if(count==2)
			System.out.println("Given number "+num+" is prime");
		else
			System.out.println("Given number "+num+" is not a prime");

		/*// Program to find prime numbers upto given number
		System.out.println("Enter a number to find prime numbers upto it");
		num=Integer.parseInt(br.readLine());

		for(i=2;i<=num;i++)
		{
			for(j=1;j<=i;j++)
			{
				if(i%j==0)
					count++;
			}
			if(count==2)
				System.out.print(i+" ");
		}
		*/
	}
}
```

Output:

```
D:\Java_Programs>java Prime_Numbers
Enter a number to find it is prime or not
13
Given number 13 is prime
```

Program – 13:

Write a program to find the number of digits in a given number?

Input: 1212

Output: 4

Source Code:

```
import java.io.*;
class NumberOfDigits
{
	public static void main(String args[])throws Exception
	{
		BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
		int num,count=0;
		System.out.println("Enter a number");
		num=Integer.parseInt(br.readLine());
```

```
		while(num>0)
		{
			num=num/10;
			count++;
		}
		System.out.println("The number of digits of a given number is: "+count);
		/*
			// Another approach to find the number of digits
			count=(int)Math.log10(num))+1;
			System.out.println("The number of digits of a given number is: "+count);
		*/
	}
}
```

Output:

```
D:\Java_Programs>java NumberOfDigits
Enter a number
12345
The number of digits of a given number is: 5
```

Program – 14:

Write a program to find sum of the digits of a given number?

Input: 12345

Output: 15

Source Code:

```
import java.io.*;
class SumOfDigits
{
	public static void main(String args[])throws Exception
	{
		BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
		int num,sum=0,rem;
		System.out.println("Enter a number");
		num=Integer.parseInt(br.readLine());
		while(num>0)
		{
			rem=num%10;
			sum=sum+rem;
			num=num/10;
		}
		System.out.println("Sum of the digits of a given number is: "+sum);
	}
}
```

Output:

```
D:\Java_Programs>java SumOfDigits
Enter a number
12345
Sum of the digits of a given number is: 15
```

Program – 15:

Write a program to find the given number is palindrome or not?

Input: 121

Output: Palindrome

Source Code:

```
import java.io.*;
class Palindrome
{
	public static void main(String args[])throws Exception
	{
		BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
		int num,sum=0,rem,m;
		System.out.println("Enter a number");
		num=Integer.parseInt(br.readLine());
		m=num;
```

```
                while(num>0)
                {
                        rem=num%10;
                        sum=sum*10+rem;
                        num=num/10;
                }
                if(sum==m)
                        System.out.println(m+" is Palindrome");
                else
                        System.out.println(m+" is not a Palindrome");
        }
}
```

Output:

```
D:\Java_Programs>java Palindrome
Enter a number
121
121 is Palindrome
```

Program – 16:

Write a program to find the given number is Armstrong or not?

Input: 153

Output: 153 is Armstrong Number

Source Code:

```
import java.io.*;
class Armstrong
{
        public static void main(String args[])throws Exception
        {
                BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
                int num,sum=0,rem,m,d;
                System.out.println("Enter a number");
                num=Integer.parseInt(br.readLine());
                d=(int)Math.log10(num)+1;
                m=num;
                while(num>0)
                {
                        rem=num%10;
                        sum=sum+(int)Math.pow(rem,d);
                        num=num/10;
                }
                if(sum==m)
                        System.out.println(m+" is Armstrong Number");
                else
                        System.out.println(m+" is not an Armstrong Number");
        }
}
```

Output:

```
D:\Java_Programs>java Armstrong
Enter a number
153
153 is Armstrong Number
D:\Java_Programs>java Armstrong
Enter a number
8
8 is Armstrong Number
D:\Java_Programs>java Armstrong
Enter a number
1634
1634 is Armstrong Number
D:\Java_Programs>java Armstrong
Enter a number
4150
4150 is not an Armstrong Number
```

Program – 17:

Write a program to explain working with do-while loop.

Source Code:

```
import java.io.*;
class doWhileDemo
{
	public static void main(String args[])throws Exception
	{
		BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
		int op,num1,num2;
		char ch;
		do
		{
		    System.out.println("1. Addition\n2. Subtraction\n3. Multiplication\n4. Division");
			System.out.println("Enter your Choice");
			op=Integer.parseInt(br.readLine());
			System.out.println("Enter two numbers");
			num1=Integer.parseInt(br.readLine());
			num2=Integer.parseInt(br.readLine());
			switch(op)
			{
				case 1: System.out.println("SUM="+(num1+num2));
					break;
				case 2:System.out.println("SUB="+(num1-num2));
					break;
				case 3:System.out.println("MUL"+(num1*num2));
					break;
				case 4:System.out.println("DIV"+(num1/num2));
					break;
				default:System.out.println("Enter valid Option");
			}
			System.out.println("Do you want to continue Yes -Y/No -N");
			ch=(char)br.read();
			br.readLine();
		}
		while (ch!='N');
	}
}
```

Output:

```
D:\Java_Programs>javac doWhileDemo.java
D:\Java_Programs>java doWhileDemo
1. Addition
2. Subtraction
3. Multiplication
4. Division
Enter your Choice
1
Enter two numbers
10
20
SUM=30
Do you want to continue Yes -Y/No -N
Y
1. Addition
2. Subtraction
3. Multiplication
4. Division
Enter your Choice
3
Enter two numbers
25
12
```

MUL300
Do you want to continue Yes -Y/No –N
.....

Program – 18:

Write a Java program to display the sum of all the values passed through commandline arguments.

Source Code:

```
class CommandlineExample
{
	public static void main(String args[])
	{
		int i,sum=0;
		for(i=0;i<args.length;i++)
		{
			sum=sum+Integer.parseInt(args[i]);
		}

		System.out.println("The Sum of "+args.length+" Values is: "+sum);
	}
}
```

```
java Commandline4 10 20 30
The Sum of 3 Values is: 60
```

TYPE CASTING:

Type casting is when you assign a value of one primitive data type to another type.
In Java, there are two types of casting:

- **Widening Casting** (automatically) - converting a smaller type to a larger type size
  byte -> short -> char -> int -> long -> float -> double
- **Narrowing Casting** (manually) - converting a larger type to a smaller size type
  double -> float -> long -> int -> char -> short -> byte

Widening: Widening casting is done automatically when passing a smaller size type to a larger size type.

Example:

```
public class Main {
  public static void main(String[] args) {
    int myInt = 9;
    double myDouble = myInt; // Automatic casting: int to double

    System.out.println(myInt);      // Outputs 9
    System.out.println(myDouble);   // Outputs 9.0
  }
}
```

Narrowing: Narrowing casting must be done manually by placing the type in parentheses in front of the value.

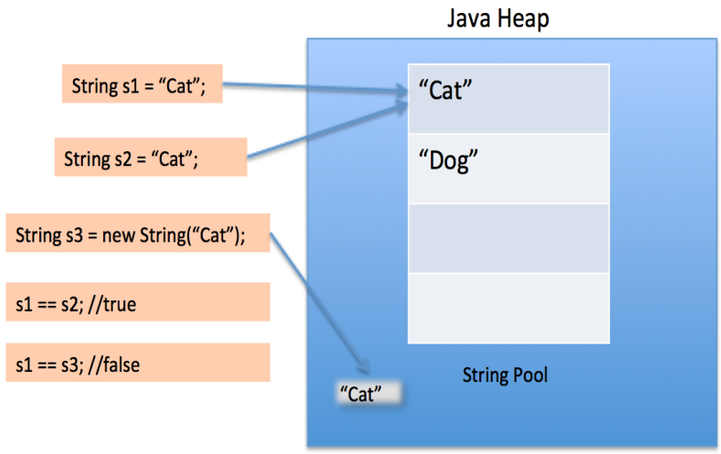
Example:

```
public class Main {
  public static void main(String[] args) {
    double myDouble = 9.78d;
    int myInt = (int) myDouble; // Manual casting: double to int

    System.out.println(myDouble);   // Outputs 9.78
    System.out.println(myInt);      // Outputs 9
  }
}
```

String

Generally, String is a sequence of characters. But in Java, string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object.
There are two ways to create String object:

1) **By string literal**
   Java String literal is created by using double quotes.
   For Example String s1="Cat";
   Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool.
2) **By new keyword**
   String s2=new String("Cat"); **//creates two objects and one reference variable**
   In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Cat" will be placed in the string constant pool. The variable s2 will refer to the object in a heap (non-pool).



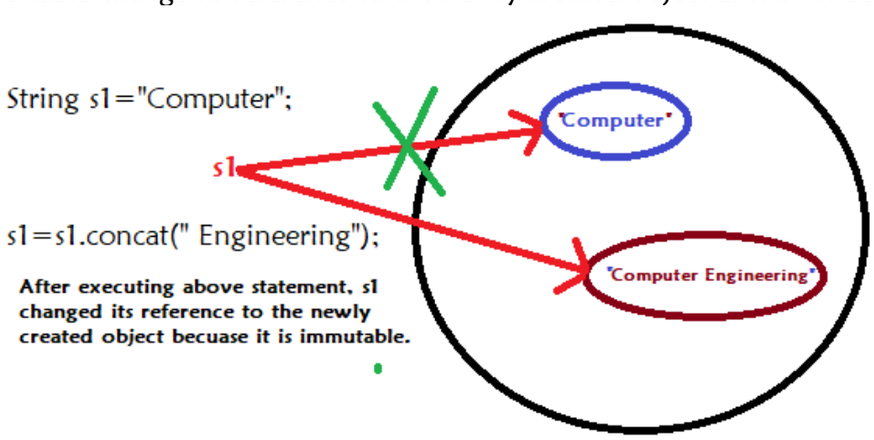
Mutability vs Immutability:

The **mutable objects** can be allowed to change its content or state without creating a new object. Whereas, the **immutable objects** can not be changed to its value or state once it is created. In the case of immutable objects, whenever we change the state of the object, **a new object will be created.**

In Java, String objects are immutable. Immutable simply means unmodifiable or unchangeable. Once String object is created its data or state can't be changed but a new String object is created. Let's try to understand the concept of immutability by the example given below:

```
class Testimmutablestring
{
  public static void main(String args[])
  {
        String s1="Computer";
        s1.concat(" Engineering");    //concat() method appends the string at the end
        System.out.println(s1);       //will print Computer because strings are immutable objects
 }
}
```

If you want to update that in s1 varaible then write the statement like s1=s1.concat(" Engineering"). So s1 reference variable change its reference to the newly created object as shown below.



Important String Constructors:

1) String s=new String(); -> Create an empty String object
2) String s=new String(String literal); -> Create a String object that stores the string literal value
3) String s=new String(StringBuffer sb);
4) String s=new String(StringBuilder sb);
5) String s=new String(char[] a); -> Used to convert a character array to string object

Example:

```
char[] ch={'j','a','v','a'};
String s=new String(ch);
System.out.println(s);          // it prints "java"
```

Important Methods of String class:

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

No.	Method Signature	Description
1	char charAt(int index)	It returns char value for the particular index
2	int length()	It returns string length
3	String substring(int beginIndex)	It returns substring for given begin index.
4	String substring(int beginIndex, int endIndex)	It returns substring for given begin index and end index.
5	boolean contains(CharSequence s)	It returns true or false after matching the sequence of char value.
6	String join(CharSequence delimiter, CharSequence... elements)	It returns a joined string.
7	boolean equals(Object another)	It checks the equality of string with the given object.
8	boolean isEmpty()	It checks if string is empty.
9	String concat(String str)	It concatenates the specified string.
10	String replace(char old, char new)	It replaces all occurrences of the specified char value.
11	String replace(CharSequence old, CharSequence new)	It replaces all occurrences of the specified CharSequence.
12	String equalsIgnoreCase(String another)	It compares another string. It doesn't check case.
13	String[] split(String regex)	It returns a split string matching regex.
14	int indexOf(int ch)	It returns the specified char value index.
15	int indexOf(int ch, int fromIndex)	It returns the specified char value index starting with given index.
16	int indexOf(String substring)	It returns the specified substring index.
17	int indexOf(String substring, int fromIndex)	It returns the specified substring index starting with given index.
18	String toLowerCase()	It returns a string in lowercase.
19	String toLowerCase(Locale l)	It returns a string in lowercase using specified locale.
20	String toUpperCase()	It returns a string in uppercase.
21	String toUpperCase(Locale l)	It returns a string in uppercase using specified locale.
22	String trim()	It removes beginning and ending spaces of this string.
23	String valueOf(int value)	It converts given type into string. It is an overloaded method.
24	char[] toCharArray()	It return a character array of the given string.

Working with String Methods:

```
import java.util.*;
public class String_MethodsDemo
{
        public static void main(String args[])
        {
          // To understand Mutability of a String
              /*
               String s=new String("Aditya");
               StringBuffer s1=new StringBuffer("Aditya");
               s.concat("College");
               s1.append("College");

               System.out.println(s);
               System.out.println(s1);
              */

          //length() - used to return the number of characters in the string.
           /*
            String s1=new String("Aditya Group of Educations");
            int len=s1.length();
                System.out.println(s1+" "+len);
           */

         // == -> reference comparision and equals() -> Content Comparision
          /*
                String s1=new String("aditya");
                String s2=new String("aditya");
                String s3="aditya";
                String s4="aditya";
                System.out.println(s1==s2);
                System.out.println(s1.equals(s2));

                System.out.println(s3==s4);
                System.out.println(s3.equals(s4));

         */
          //equalsIgnoreCase() - Compares two strings, ignoring case considerations
           /*
                String s1=new String("aditya");
                String s2=new String("AdityA");

                System.out.println(s1.equals(s2));
                System.out.println(s1.equalsIgnoreCase(s2));
         */

          //charAt(int) - return the character at the given index.
          /*
                String s11=new String("Aditya Group of Educations");
                System.out.println(s11.charAt(20));
          */

          //indexOf()- from start , lastIndexOf() - from last
           /*
                String s3=new String("Aditya Engineering College");
               System.out.println(s3.lastIndexOf('e'));
               System.out.println(s3.indexOf('a'));                    //5
               System.out.println(s3.lastIndexOf('a'));     //5
            */
```

```
//replace(old char,new char), replaceFirst(String,String), replaceAll()
 /*
        String s1=new String("Aditya Group of Colleges");
        System.out.println(s1.replace('o','e'));   //Aditya Greup ef Celleges

        String s2=new String("Aditya College is best Engineering College");
        System.out.println(s2.replaceFirst("College","Group"));
        System.out.println(s2.replaceAll("College","Group"));
 */
//Program to find the how many times a word available in a given string.
      /*
         String s1=new String("arehowruaresyeare");
         int l=(s1.length()-s1.replaceAll("are","").length())/3;
         System.out.println(l);
      */
//split(String reg,int limit) - return the string array
 /*
        String s1=new String("06-10-2020");
        String s2[]=s1.split("-",0);
        for(int i=0;i<s2.length;i++)
        System.out.println(s2[i]);
 */
// Program to read line of integers by using either BR or Scanner class
      /*
        Scanner sc=new Scanner(System.in);
        String value=sc.nextLine();
        String ele[]=value.split(" ",0);

        int number[]=new int[ele.length];
        for(int i=0;i<ele.length;i++)
                number[i]=Integer.parseInt(ele[i]);

        for(int i=0;i<number.length;i++)
               System.out.println(number[i]);
      */
 /*
        String s=new String("Aditya engineering college is best engineering college");
        String s1[]=s.split(" ",0);
        System.out.println(s1.length);
 */
//startWith(), endsWith()
 /*
      String s=new String("Aditya Degree Colleges");
      System.out.println(s.startsWith("Adi"));  // true
      System.out.println(s.endsWith("ges"));   // true
 */
//contains(String) - search for the given string in it or not.
      /*
        String myStr = "Aditya";
        System.out.println(myStr.contains("Adi"));   // true
        System.out.println(myStr.contains("y"));     // true
        System.out.println(myStr.contains("dtya"));    // false
      */
//toCharArray() and copyValueOf()
 /*
       String s1=new String("12-34-56");
       //System.out.println(s1[0]);
       char x[]=s1.toCharArray();
       for(int i=0;i<x.length;i++)
          System.out.println(x[i]);
```

```
                x[2]='%';
                x[5]='$';
                String s2=new String(x);
                System.out.println(s2);  // "12%34$56
            */
            /*
                String s3="";
                s3=s3.copyValueOf(x,0,5);
                System.out.println(s3);

                String s4=String.valueOf(x);
                System.out.println(s4);
            */

        //substring(start,end) - to return the specified substring from a string start to end-1.
            /*
                String s1=new String("Aditya Group of Educations");

                String s2=s1.substring(7);
                System.out.println(s2);     // Group of Educations

                String s3=s1.substring(7,11);
                System.out.println(s3);     // Grou
              */
        }
}
```

StringBuffer:

StringBuffer is also class in java.lang package used to store group of characters**. It is a Mutable object.**

Need of StringBuffer:

If the content is fixed and never be changed, then we go for String objects like city name, college name, address.

Drawback of String class:

For every change a new object is created in memory. it degrades the performance. if the content keep on changing then we should go for **StringBuffer.** All the required changes are applied in the existing object only because StringBuffer is mutable object.

StringBuffer Object creation:

1) StringBuffer sb=new StringBuffer(); **--> 16 characters is the default capacity**

   If the appended contents its capacity then capacity will increase like this

   new capacity=(currentcapacity+1)*2 = 34

   Example:

```
StringBuffer sb=new StringBuffer();
System.out.println(sb.capacity());              // 16
sb.append("abcdefghijklmnop");
System.out.println (sb.capacity());             //16
sb.append("q");
System.out.println (sb.capacity());             // 70
```

2) StringBuffer sb=new StringBuffer(int capcity);

3) StringBuffer sb=new StringBuffer(String s);

   capacity=s.length()+16

Important methods of StringBuffer

* length()	* capacity()
* charAt()	* setCharAt()
* append()	* insert()
* delete()	* deleteCharAt()
* reverse()	* setlength()
* ensureCapacity()	* trimToSize()

Working with StringBuffer methods:

```
class StringBufffer_Demo
{
        public static void main(String args[])
        {
                StringBuffer s1=new StringBuffer();
                System.out.println(s1.capacity());                  // 16
                System.out.println(s1.length());                    // 0
                s1.append("abcdefghijklmnopq");
                System.out.println(s1.capacity());                  // 34
                System.out.println(s1.length());                    // 17
                StringBuffer s2=new StringBuffer("Welcome ");

                System.out.println(s2.charAt(4));                   // o
                s2.setCharAt(4,'a');
                System.out.println(s2);                             // Welcame

                s2.deleteCharAt(4);
                System.out.println(s2);                             // Welcme

                s2.append(" Srinu");
                System.out.println(s2);                             // Welcme Srinu

                s2.insert(4,"a");
                System.out.println(s2);                             // Welcame Srinu

                s2.delete(8,13);
                System.out.println(s2);                             // Welcame u

                s2.append(true);
                System.out.println(s2);                             // Welcame utrue

                s2.reverse();
                System.out.println(s2);                             // eurtu emacleW

     /*
      StringBuffer sb=new StringBuffer("Aditya Engineering College");
      System.out.println(sb.capacity());
           sb.setLength(6);
           System.out.println(sb);

           sb.ensureCapacity(1000);
           System.out.println(sb.capacity());

           sb.trimToSize();
           System.out.println(sb.capacity());
          */

        }
}
```

StringBuilder:

Every method is inside the StringBuffer is synchronized. i.e Only one thread can access at a time. In multithreading environment StringBuffer is not suitable. So StringBuilder having all the features of StringBuffer only the difference is that all the methods in StringBuilder are non synchronized methods hence multiple threads can allow to access at same time.

Practice Programs:

1. **Write a program which reads a line of text and then prints the word number, the number of characters in it.**
   Sample Input:
   AEC AECT ACOE

Sample Output:

1 AEC 3
2 ACET 4
3 ACOE 4

```
import java.io.*;
class Word_Count
{
    public static void main(String args[])throws Exception
    {
            BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
         String str1;
         str1=br.readLine();
         String words[]=str1.split(" ");
         for(int i=0;i<words.length;i++)
        {
            System.out.print("(i+1)+" "+words[i]+" "+words[i].length());
        }
    }
}
```

2. **Write a program which takes two strings and checks whether the second string was embedded in first or not.**

```
import java.io.*;
class String1
{
        public static void main(String args[])throws Exception
        {
       BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
                String str1,str2;
                str1=br.readLine();
                str2=br.readLine();

                if(str1.contains(str2))
                        System.out.println("YES");
                else
                        System.out.println("NO");
        }
}
```

3. **Write a program to read a line of text and then find the number of uppercase, lowercase, digits and symbols in it.**

```
import java.io.*;
class String2
{
        public static void main(String args[])throws Exception
        {
            BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
            String str1;
            int i,l=0,u=0,d=0,s=0;
              str1=br.readLine();
            char ch[]=str1.toCharArray();
                 for(i=0;i<ch.length;i++)
                {
                        if(Character.isLowerCase(ch[i]))
                                l++;
                        else if(Character.isUpperCase(ch[i]))
                                u++;
                        else if(Character.isDigit(ch[i]))
                                d++;
                        else
                                s++;
                }
                System.out.println(l+" "+u+" "+d+" "+s);
        }
```

```
}
```

4. **Write a program to remove characters in the given string except alphabets.**

Input: pr'og2^ram-ming
Output: programming

```
import java.io.*;
class String3
{
   public static void main(String args[])throws Exception
   {
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        String str1,str2="";
         int i,sum=0;
        str1=br.readLine();
           char ch[]=str1.toCharArray();
        for(i=0;i<ch.length;i++)
        {
                if(Character.isLetter(ch[i]))
                        str2=str2+ch[i];
        }
        System.out.println(str2);
   }
}
```

5. **Write a program which reads a line of text and prints all the words in reverse order in it.**

Input: This is Coding Platform
Output: sihT si gnidoC mroftalP

```
import java.io.*;
class String_Reverse
{
  public static void main(String args[])throws Exception
   {
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        String str1;
        StringBuffer sb1;
        str1=br.readLine();

        String words[]=str1.split(" ");
        for(int i=0;i<words.length;i++)
        {
                sb1=new StringBuffer(words[i]);
                System.out.print(sb1.reverse()+" ");
        }
    }
}
```

6. **Write a program to sort given set of strings.**

```
import java.io.*;
class SortStringArray
{
        public static void main(String args[])throws Exception
        {
                BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
                String input=br.readLine();
                String[] countries = input.split(" ");
                int size = countries.length;
                int i,j;
                //logic for sorting
                for(i=0;i<size-1;i++)
                {
                        for (j=i+1;j<countries.length;j++)
                        {
                            //compares each elements of the array to all the remaining elements
                              if(countries[i].compareTo(countries[j])>0)
                              {
```

```
                                        //swapping array elements
                                        String temp = countries[i];
                                        countries[i] = countries[j];
                                        countries[j] = temp;
                                }
                        }
                }
                //prints the sorted array in ascending order
                for(String name:countries)
                        System.out.println(name);
        }
}
```

Output:

```
D:\JAVA-Programs>java SortStringArray
India Afghnistan America Britan Pakistan England
Afghnistan
America
Britan
England
India
Pakistan
```

7. **Program to find the sum of the digits in a given string.**

```
import java.io.*;
class String3
{
        public static void main(String args[])throws Exception
        {
                BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
                String str1,str2="";
                 int i,sum=0;
                str1=br.readLine();
                   char ch[]=str1.toCharArray();
                for(i=0;i<ch.length;i++)
                {
                        if(Character.isDigit(ch[i]))
                           sum=sum+Integer.parseInt(ch[i]+"");
                         // or sum=sum+(ch[i]-48);
                }
                System.out.println(sum);
        }
}
```

8. **Program to find the no of vowels and consonants in a given string.**

```
import java.io.*;
class string12
{
        public static void main(String args[])throws Exception
        {
                 BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
                 String input;
                 int v=0,c=0;
                 input=br.readLine();
                 char x[]=input.toCharArray();
                 for(int i=0;i<x.length;i++)
                {
                         if(x[i]=='a' || x[i]=='e' || x[i]=='i' || x[i]=='o' ||x[i]=='u')
                                v++;
                        else
                                c++;
                }
                System.out.println(v+" "+c);
        }
}
```

Array

An array is an indexed collection of fixed number of homogeneous data elements.

The main **advantage** of arrays is we can represent multiple values with the same name so that readability of the code will be improved.

But the main **disadvantage** of arrays is:

Fixed in size i.e once we created an array there is no chance of increasing or decreasing the size based on our requirement that is to use arrays concept compulsory we should know the size in advance which may not possible always.

We can resolve this problem by using collections.

Single dimensional array declaration:

Example:

```
int[] a;        //recommended to use because name is clearly separated from the type
int []a;
int a[];
```

At the time of declaration we can't specify the size otherwise we will get compile time error.

Example:

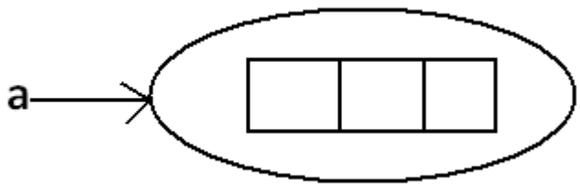
```
int[] a;        //valid
int[5] a;       //invalid
```

Every array in java is an object hence we can create by using new operator.

Example:

```
int[] a=new int[3];
```

Diagram:



Example:

```
import java.io.*;
class  Array
{
   public static void main (String args[])throws Exception
  {
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader (isr);
        int x[],i,n;
        System .out.println ("Enter no.of elements:");
        n = Integer.parseInt(br.readLine());
        x = new int[n];

        System.out.println("Enter  "+n+" Elements");
        for(i=0;i<n;i++)
         x[i] = Integer.parseInt(br.readLine ());

        System.out.println ("Given array:\n");
        for (i=0;i<n;i++)
          System.out.println(x[i]);
  }
}
```

```
//To calculate standard deviation sd = SQRT( Σ(xi-mean)²/n)
import java.io.*;
class StdDev
{
  public static void main (String args [])throws Exception
   {
            InputStreamReader isr = new InputStreamReader (System.in);
            BufferedReader b = new BufferedReader (isr);
            int x [],n,i,sum=0,t=0,mean;
            float sd;
            System.out.println("Enter No.of elements");
            n = Integer.parseInt(b.readLine());
            x=new int [n];
            System.out.println("Enter "+n+" Elements");
            for(i=0;i<n;i++)
            {
                    x[i] = Integer.parseInt(b.readLine());
                    sum = sum + x[i];
            }
            mean = sum/n;
            for(i=0;i<n;i++)
            {
                    t = t +(x[i]-mean)*(x[i]-mean);
            }
            sd = (float)Math.sqrt(t/n);
            System.out.println(sd);
    }
}
// Program to find the sorted order of given array
import java.io.* ;
class sort
{
  public static void main (string args [])throws exception
  {
    InputStreamReader isr = new BufferedReader(System.in);
    BufferedReader b = new BufferedReader (isr);
    int x [], i,j,t
    System.out.println ("Enter the no.of elements ");
    n = Integer.parseInt (b.readLine ());
    x = new int [n];
    System.out.println("enter "+n" elements ");
    for(i = 0 ;i<n;i++)
    {
        x [i] = Integer.parseInt (b.readLine());
    }
    for(i = 0;i<n-1; i++)
    {
        for(j =i+1; j<n; j++)
        {
            if (x[i]>x[j])
            {
                t = x[i];
                x[j] = x[i];
                x [j] = t;
            }
        }
    }
    System.out.println( "Sorted Array:");
    for (i = 0;i < n;i++)
            System.out.println(x[i]);
  }
}
```

```
//Program to Implement Binary Search
import java.io.*;
class Binary_Search
{
  public static void binarySearch(int arr[], int first, int last, int key)
        {
                int mid = (first + last)/2;
                while(first <= last)
                {
                        if(arr[mid]<key)
                        {
                                first = mid + 1;
                        }
                        else if(arr[mid] == key)
                        {
                                System.out.println("Element is found at index: " + mid);
                                break;
                        }
                        else
                        {
                                last = mid - 1;
                        }
                        mid = (first + last)/2;
                }
                if(first > last )
                {
                        System.out.println("Element is not found!");
                }
        }
   public static void main(String args[])throws Exception
        {
                int n,key,arr[],i;
                BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
                n=Integer.parseInt(br.readLine());
                arr=new int[n];

                for(i=0;i<n;i++)
                        arr[i]=Integer.parseInt(br.readLine());
                key=Integer.parseInt(br.readLine());

                binarySearch(arr,0,n-1,key);
   }
}

//Program to implement Merge Sort
import java.io.*;
class Merge_Sort
{
        /* Function to merge the subarrays of a[] */
        void merge(int a[], int beg, int mid, int end)
        {
                int i, j, k;
                int n1 = mid - beg + 1;
                int n2 = end - mid;

                /* temporary Arrays */
                int LeftArray[] = new int[n1];
                int RightArray[] = new int[n2];

                /* copy data to temp arrays */
                for (i = 0; i < n1; i++)
                        LeftArray[i] = a[beg + i];
```

```
        for (j = 0; j < n2; j++)
                RightArray[j] = a[mid + 1 + j];

        i = 0;          /* initial index of first sub-array */
        j = 0;          /* initial index of second sub-array */
        k = beg;        /* initial index of merged sub-array */

        while (i < n1 && j < n2)
        {
                if(LeftArray[i] <= RightArray[j])
                {
                        a[k] = LeftArray[i];
                                i++;
                }
                else
                {
                        a[k] = RightArray[j];
                        j++;
                }
                k++;
        }
        while (i<n1)
        {
                a[k] = LeftArray[i];
                i++;
                k++;
        }

        while(j<n2)
        {
                a[k]=RightArray[j];
                j++;
                k++;
        }
}

void mergeSort(int a[], int beg, int end)
{
      if(beg < end)
     {
        int mid = (beg + end) / 2;
        mergeSort(a, beg, mid);
        mergeSort(a, mid + 1, end);
        merge(a, beg, mid, end);
     }
}
/* Function to print the array */
void printArray(int a[], int n)
{
        int i;
        for (i = 0; i < n; i++)
        System.out.print(a[i] + " ");
}
public static void main(String args[])
{
         int a[] = { 11, 30, 24, 7, 31, 16, 39, 41 };
        int n = a.length;
        Merge_Sort m1 = new Merge_Sort();
        System.out.println("\nBefore sorting array elements are - ");
        m1.printArray(a, n);
        m1.mergeSort(a, 0, n - 1);
        System.out.println("\nAfter sorting array elements are - ");
```

```
                m1.printArray(a, n);
                System.out.println("");
        }
}
D:\JAVA-Programs\Arrays>java Merge_Sort
Before sorting array elements are -
11 30 24 7 31 16 39 41
After sorting array elements are -
7 11 16 24 30 31 39 41
```

//Double Dimensional array

Two dimensional array declaration:
Example:

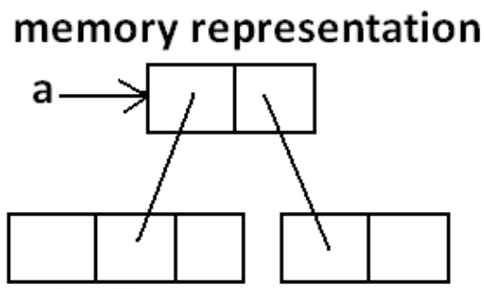
```
int[][] a;
int [][]a;
int a[][];        All are valid.(6 ways)
int[] []a;
int[] a[];
int []a[];
```

In java multidimensional arrays are implemented as array of arrays approach but not matrix form. The main advantage of this approach is to improve memory utilization.

Example1:

```
int[][] a=new int[2][];
a[0]=new int[3];
a[1]=new int[2];
```

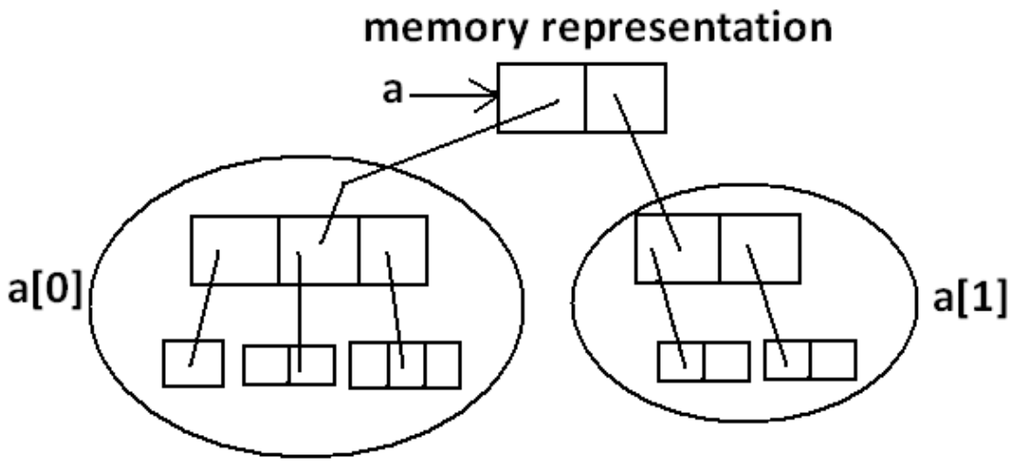
Diagram:



Example 2:

```
int[][][] a=new int[2][][];
a[0]=new int[3][];
a[0][0]=new int[1];
a[0][1]=new int[2];
a[0][2]=new int[3];
a[1]=new int[2][2];
```

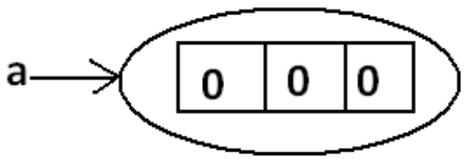
Diagram:



Whenever we are creating an array every element is initialized with default value automatically.

Example3:

```
int[] a=new int[3];
System.out.println(a);//[I@3e25a5
System.out.println(a[0]);//0
```




```
import  java.io.*;
class Matrix
{
  public static void main (String args [])throws Exception
  {
        int x[][],i,j,m,n;
        BufferedReader b = new BufferedReader (new InputStreamReader (System. in));
        System.out.printin("enter no.of rows &columns");
        m = Integer.parseInt(b.readLine());
        m = Integer. parseInt(b.readLine ());
        x =new int [m] [n];
        System.out.printin("enter"+(m*n) "elements");
        for (i = 0;i<m;i++)
           for (j = 0;j<n;j++)
                x[i][j] = Integer.parseInt(b.readLine());

        System.out.printin("Given Matrix");
        for(i = 0;i<m;i++)
        {
                for(j = 0;j<n;j++)
                        System.out.print(x[i][j]+ "\t");
                System.out.println();
        }
    }
}
```

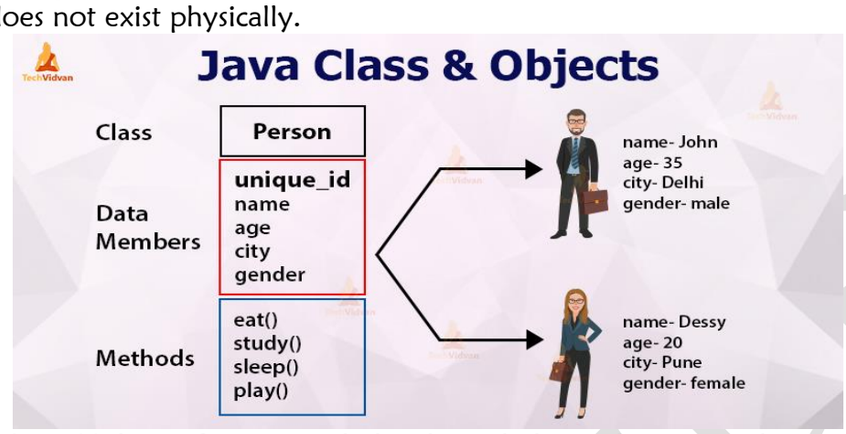
Class and Object:

Object Oriented programming is a **programming** style which is associated with the concepts like class, object, Inheritance, Encapsulation, Abstraction, Polymorphism.

Most popular **programming** languages like Java, C++, C#, Ruby, etc. follow an **object-oriented programming** paradigm.

OOPS Concepts:

1) **Object:** Any thing that exist physically in the real world is an Object. Every Object has set of properties and behaviours.
2) **Class:** A class is a group of similar objects. A class is a **model or a blue print** for creating objects and does not exist physically.



3) **Data Abstraction:** Hiding of unwanted details and show the essential one is called data abstraction.
4) **Encapsulation:** Encapsulation in Java is a process of wrapping code and data together into a single unit, for example, a capsule which is mixed of several medicines.
5) **Inheritance:** Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. ... The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as super class (base class, parent class).
6) **Polymorphism:** The word polymorphism means having many forms. ... For example the same person posses different behaviour in different situations. This is called polymorphism.

Syntax of creating an object:

<class-name> object=new <class-name>();

Example: Sample s=new Sample();

new it is an operator. This is equivalent to malloc() but there is no need of typecasting and size. For normal functions we have to create objects in main to access them. Object stores the address just like pointer in C.

```
class Sample
{
        void wish (){
                System.out.print in ("have a nice day");
        }
        public static void main (String args [])
        {
                Sample s=new Sample();
                s.wish ();
        }
}
```

To call normal functions in normal functions, we do not need objects if they are in same class.

```
class Sample
{
        void wish (){
            System.out.println ("Test Fun");
        }
        void test(){
                wish ();
        }
        public static void main (String args[])
        {
                Sample s=new Sample ();
                s.test();
        }
}
```

Constructor

Object creation is not enough compulsory we should perform initialization then only the object is in a position to provide the response properly. Whenever we are creating an object some piece of the code will be executed automatically to perform initialization of an object this piece of the code is nothing but constructor.

Hence the main objective of constructor is to perform initialization of an object.

Constructor is a special type of member function. It is used to initialize the member variables at the time of creating objects.

Rules:

1. Constructor name is always same as class-name
2. Constructor will take arguments but will not return any value.
3. Even though it is not returning the value, we should not use void before constructor name.
4. We can define more than 1 constructor with in a class.

Types of constructor:

1. Default constructor: - It will not take arguments.
2. Parameterized constructor: It will take arguments.
3. Copy constructor: It will copy the values of one object into other object.

Destructors: It is also a special type of member function, which is used to destroy the objects, which are constructed by the constructor.

Note: Java does not support destructors. But it supports garbage collectors. It is a special type of program which will destroy the objects which are not referred for long time the time value which is used for garbage collection is called lease value.

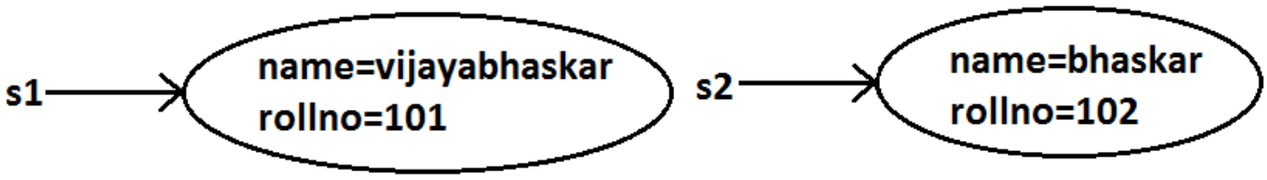
Whenever any object will not be referred within that lease time it will automatically garbage collected by systems default garbage collector.

Example1:

```
class Test
{
    public Test()                          // Default Constructor
    {
        System.out.println("Default Constructor");
        System.out.println("Welcome to Guest");
    }
    public Test(String name)               // Parameterized Constructor
     {
         System.out.println("Parameterized Constructor");
        System.out.println("Welcome to "+name);
     }
    public static void main(String args[])
     {
           Test t1=new Test();                    // called Default Constructor
          Test t2=new Test("Srinu");              // called Parameterized Constructor
     }
}
```

Example2:

```
class Student
{
        String name;
        int rollno;
        Student(String name,int rollno)     //Constructor
        {
                this.name=name;
                this.rollno=rollno;
        }
        public static void main(String[] args)
        {
                Student s1=new Student("vijayabhaskar",101);
                Student s2=new Student("bhaskar",102);
        }
}
```



Example3:

```
class demo
{
  int a,b;
  demo()
  {
     a=10;
     b=20;
  }
  public void print()
   {
      System.out.print("a= "+a+" b = "+b);
   }
}
class one
{
  public static void main(String args[])
  {
    demo obj1=new demo();
    demo obj2=obj1;
    obj1.a+=1;
    obj1.b+=1;
    System.out.println("values of obj1: ");
    obj1.print();
    System.out.println("\nvalues of obj2: ");
    obj2.print();
   }
}
```

Output:

```
D:\java_prog>java one
values of obj1:
a= 11 b = 21
values of obj2:
a= 11 b = 21
```

Overloaded Constructors

A class can contain more than one constructor and all these constructors having the same name but different arguments and hence these constructors are considered as overloaded constructors.

Example1:

```
class Test
{
   int x;
   public Test ()
        {
                System.out.println("Default Constructor");
                x = 1;
        }
        public Test (int   v)
        {
                System.out.println("Parameterized constructor");
                x = v;
        }
        public Test (Test  t)
        {
                x = t.x;
                System.out.println("Copy Consturctor");
        }
        void show()
        {
                System.out.println("X:"+x);
        }
        public static void main (String args [])
        {
                Test t1 = new test (); // default
                Test  t2 = new test (13); // parameterized
                Test  t3 = new test (t2); // copy
                T1.show (); // 1
                T2. show (); // 13
                T3.show (); // 13
        }
}
```

Example2: //Stacks using Constructors

```
class Stacks
{       int x[], top, size;
        public Stacks ()
        {       top = -1;
                size = 5;
                x = new int [size];
        }
        public Stacks (int  s)
        {       top = -1;
                size = s;
                x = new int [s];
        }
        int getSize ()
        {
                return size ;
        }

        void push(int ele)
        {       top++;
                if(top>=size )
                {
                        System.out.println("Stack is Overflow");
                        top--;
                }
                else
```

```
                x[top]=ele;
        }
        void pop()
        {
                if(top<0)
                        System.out.println("Stack is Underflow");
                else
                {
                        System.out.println ("Deleted Ele:" + x[top]);
                        top--;
                }
        }
        void show ()
        {
                if (top>= 0)
                {
                        for (int  i =top; i >= 0; i--)
                        System.out.println(x[I]);
                }
        }

        public static void main (String args [])
        {       int   o;
                Stacks s1 = new Stacks ();
                Stacks s2 = new Stacks(10);
                int n1 = s1.getSize();
                int n2 = s2. getSize();
                for(i=1;i<=n1;i++)
                s1.push(i+i);
                for (i = 1;i<= n2;i++)
                s2.push(i*i);
                s1.show();
                s2.show();
                s1.pop();
                s2.pop();
                s1.show();
                s2.show();
        }
}
```

this Object

It is an implicit, dynamic, generic object to store the address, which is in the currently executing object. It is used to differentiate local variable with member variable, if both of these variables having same name. We have to place "this" keyword before the member variable name.

Eg:

```
class Stack
{
        int x[],top, size.
        public stack (int size)
        {
                top = -1;
                this .size = size;
                x = new int [size]
        }
}
```

//Usage of "this" object to add 2no.s:

```
import java.io.*;
class Numbers
{
        int x;
        BufferedReader b=new BufferedReader(new InputStreamReader(System.in));
        public void getX() throws Exception
        {
                System.out.println("Enter a No");
                x = Integer.parseInt(b.readLine());
        }
        public Numbers add(Numbers n)
        {
                Numbers n3 =new Numbers();
                n3.x=x+ n.x;
                return n3;
        }
        public Numbers max(Numbers n)
        {
                if(x>=n.x)
                        return this;
                else
                        return n3;
        }

        public void show()
        {
                System.out.println("X:"+x);
        }
        public static void main (String args [])throws Exception
        {
                Numbers n1 = new Numbers();
                Numbers n2 = new Numbers ();
                n1.getX();
                n2.getX();
                Numbers n3 = n1.add(n2);
                Numbers n4=n1.max(n2);
                n1.show();
                n2.show();
                n3.show();
                n4.show();
        }
}
```

Note:
In java, call by reference cannot be applied to the normal variables. Objects cannot use call by value.
Whenever you pass object into a function, it will always be treated as call by reference.

// Multiplication of two complex numbers:

```
import java.io.*;
class Complex
{
        int real,imag;
        void getX()
        {
                System.out.println("enter real & imaginary parts");
                real = Integer.parseInt (b.readLine());
                imag = Integer.parseInt(b.readLine());
        }
        Complex multiply(Complex n)
        {
                Complex  n3 =new Complex ();
                n3.real = real*n.real-imag*n.imag;
                n3.imag = real*n.imag+imag*n.real;
                return n3;
        }
        void show()
        {
                if (imag >= 0)
                        System.out.println(real + "+" + imag + "j");
                else
                        System.out.println(real + "" + imag + "j");
        }
        public static void main (string args[])throws Exception
        {
                Complex c1 = new Complex();
                Complex c2 = new Complex();
                c1.getX();
                c2.getX();
                Complex c3 = c1.multiply(c2);
                c1.show();
                c2.show();
                c3.show();
        }
}
```

Static Members

Whenever we declare some properties within a class, if we want to access those properties we have to create objects to that class. Such type of properties is called normal property. For those properties values are different from one object to another object but properties are common. In some of the cases, properties as well as their values are common to any object. Such types of properties are called **"Static Properties"**. They are independent from objects. So we cannot use objects to call those properties. We have to use class name to invoke static members. "this" object cannot be applied to the static members.

```
import java.io. *;
class Emp
{
        int  eno;
        String  ename;
        float  sal;
        static int totsal;
        static BufferedReader b = new BufferedReader (new InputStreamReader(System.in));
        void getemp ()throws Exception
        {
                System.out.println ("Enter eno, ename, sal:");
                eno = Integer.parseInt(Emp.b.readLine());
                ename =       Emp.b.readLine ();
                sal = Float.parseFloat (Emp.b.readLine ());
                Emp.totsal += sal;
        }
        void showEmp ()
        {
                System.out.println("Eno:"+eno);
                System.out.println ("Name:"+ename);
                System.out.println("Sal:"+ sal);
        }
        static void showTotal ()
        {
                System.out.println ("Total:"+totsal);
        }
        public static void main (string args [])throws Exception
        {
                int i,n;
                System.out.println ("Enter no. Of Emps");
                n = Integer.parseInt(b.readLine ());
                Emp e[]=new Emp [n];
                for (i = 0; i<n; i++)
                {
                        e[i] = new Emp();
                        e[i].getEmp();
                }
                System.out.println("Given Emps:");
                for (i = 0 ;i<n; i++)
                        e[i]. showEmp();

                showTotal ();
        }
}
```

Method Overloading

If a class have multiple methods with same name but with different parameters list, it is known as Method Overloading.

Parameters lists should differ in either,

- Number of parameters.
- Data type of parameters.
- Sequence of data type of parameters.

1. Two methods are said to be overload if and only if both having the same name but different argument types.
2. In 'C' language we can't take 2 methods with the same name and different types. If there is a change in argument type compulsory we should go for new method name.

Example :

abs() ———— for int type
labs() ———— for long type
fabs() ———— for float type
.
.
etc

3. Lack of overloading in "C" increases complexity of the programming.
4. But in java we can take multiple methods with the same name and different argument types.

Example:

abs(int)
abs(long)
abs(float)
.
.
.

5. Having the same name and different argument types is called method overloading.
6. All these methods are considered as overloaded methods.
7. Having overloading concept in java reduces complexity of the programming.

Example:

```
class Test
 {
   public void methodOne()
   {
        System.out.println("no-arg method");
   }
   public void methodOne(int i)
   {
        System.out.println("int-arg method"); //overloaded methods
   }
   public void methodOne(double d)
   {
     System.out.println("double-arg method");
   }

   public static void main(String[] args)
   {
        Test t=new Test();
        t.methodOne();//no-arg method
        t.methodOne(10);//int-arg method
        t.methodOne(10.5);//double-arg method
    }
}
```

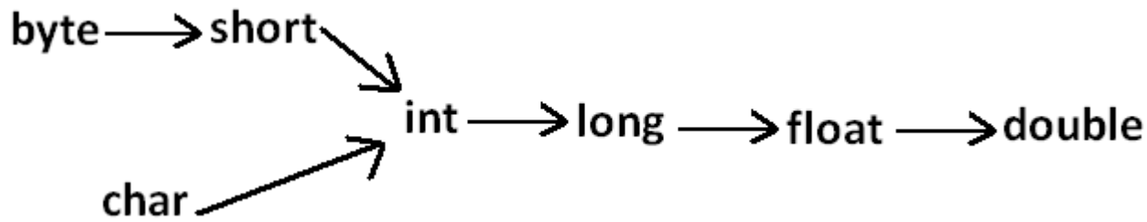
Conclusion : In overloading compiler is responsible to perform method resolution(decision) based on the reference type(but not based on run time object). Hence overloading is also considered as compile time polymorphism(or) static polymorphism (or)early biding.

Automatic promotion in overloading:

In overloading if compiler is unable to find the method with exact match we won't get any compile time error immediately.1st compiler promotes the argument to the next level and checks whether the matched method is available or not if it is available then that method will be considered if it is not available then compiler promotes the argument once again to the next level. This process will be continued until all possible promotions still if the matched method is not available then we will get compile time error.

This process is called **automatic promotion in overloading.**

The following are various possible automatic promotions in overloading.



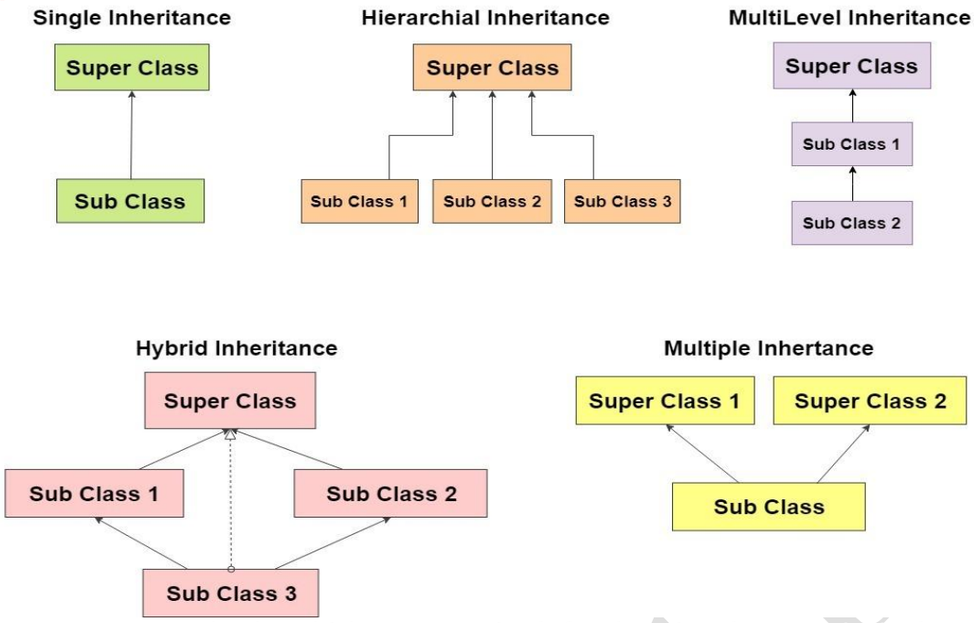
Example:

```
class Test
{
        public void methodOne(int i)
        {
                System.out.println("int-arg method");
        }
        public void methodOne(float f) //overloaded methods
        {
                System.out.println("float-arg method");
        }
        public static void main(String[] args)
        {
                Test t=new Test();
                //t.methodOne('a');         //int-arg method
                //t.methodOne(10l);         //float-arg method
                t.methodOne(10.5);          //C.E:cannot find symbol
        }
}
```

Inheritance

1. It is a process of extracting some of the properties of existing class and adds some more features, by that create a new class.
2. The existing class is called super class or base class or parent class.
3. The new class is called derived class or sub-class or child class.
4. It introduces the concept of reusability, i.e. without modifying the existing properties use them again & again

Types of Inheritances:



The common methods which are required for housing loan, vehicle loan, personal loan and education loan we can define into a separate class in parent class loan. So that automatically these methods are available to every child loan class.

Example:

```
class Loan {
        //common methods which are required for any type of loan.
}
class HousingLoan extends Loan {
        //Housing loan specific methods.
}
class EducationLoan extends Loan {
        //Education Loan specific methods.
}
```

Example:

```
class Parent {
        public void methodOne(){ }
}
class Child extends Parent {
        public void methodTwo() { }
}
```

```
class Test
{
    public static void main(String[] args)
    {
        Parent p=new Parent();
        p.methodOne();
        p.methodTwo();
        Child c=new Child();
        c.methodOne();
        c.methodTwo();
        Parent p1=new Child();
        p1.methodOne();
        p1.methodTwo();
        Child c1=new Parent();
    }
}
```

C.E: cannot find symbol
symbol : method methodTwo()
location: class Parent

C.E: incompatible types
found : Parent
required: Child

Conclusion:

1. Whatever the parent has by default available to the child but whatever the child has by default not available to the parent. Hence on the child reference we can call both parent and child class methods. But on the parent reference we can call only methods available in the parent class and we can't call child specific
methods.
2. Parent class reference can be used to hold child class object but by using that reference we can call only methods available in parent class and child specific methods we can't call.
3. Child class reference cannot be used to hold parent class object.

Example1:

```
class Parent
{
   int Currency1=20000;
   String Gold="2KG";
   String Properites="Building";
        public void Take()
        {
                System.out.println("Currency+ Gold+Properties");
        }
}
class Child extends Parent
{
        public void own()
        {
                System.out.println("Own Properties");
        }
        public static void main(String args[])
        {
                Child c=new Child();
                c.Take();
                c.own();
                System.out.println(c.Currency1);
                System.out.println(c.Gold);
                System.out.println(c.Properites);
        }
}
```

Output:

```
D:\java_prog>java Child
Currency+ Gold+Properties
Own Properties
20000
2KG
Building
```

Example2:

```
class A
{
        private int x;
        public int y;
        protected int z;
        int t;
        public void set ()
        {
                x = 1;
                x = 2;
                z = 3;
                t = 4;
        }
}
class B extends A
{
```

```
        void show ()
        {
                System.out.println("X:"+x);
                 // x cannot be accessed if in same directory
        }
        public static void main (string args [])
        {
                B   bob = new B ();
                bob.set();
                bob.show ();
                bob.x =10; // x cannot be accessed
                bob.y = 20;
                bob.z = 30;
                bob.t = 40;
                bob.show ();
        }
}
```

super() vs this()

The 1st line inside every constructor should be either super() or this() if we are not writing anything compiler will always generate super().

Case 1: We have to take super() (or) this() only in the 1st line of constructor. If we are taking anywhere else we will get compile time error.

Example:

```
class Test
{
        Test()
        {
                System.out.println("constructor");
                super();
        }
}
```

Output:

Compile time error.

Call to super must be first statement in constructor.

Case 2: We can use either super() (or) this() but not both simultaneously.

Example:

```
class Test
{
        Test()
        {
                super();
                this();
        }
}
```

Output:

Compile time error.

Call to this must be first statement in constructor

Case 3: We can use super() (or) this() only inside constructor. If we are using anywhere else we will get compile time error.

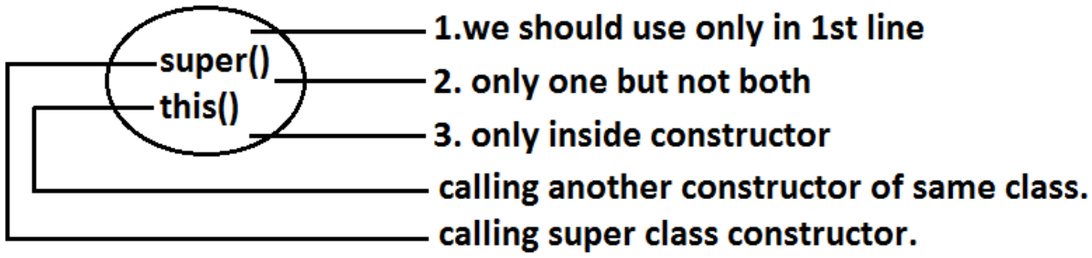
Example:

```
class Test
{
        public void methodOne()
        {
                super();
        }
}
```

Output:

Compile time error.

Call to super must be first statement in constructor




```
class A
{
    A(){
       System.out.println("A's no argument constructor");
      }

    A(String args){
        System.out.println("A's one argument constructor");
     }
}

class B extends A
{
      B(){
            this("");                    // calling one arg constructor of class B
            System.out.println("B's no argument constructor");
       }

      B(String args){
              super("");                          // calling one argument constructor of class A
            System.out.println("B's one argument constructor");
       }
}

// Test Class and Output
  public class Test
  {
            public static void main(String args[]) {
            B b = new B();
   }
}
```

Output:

```
A's one argument constructor
B's one argument constructor
B's no argument constructor
```

Example:

```
class A
{
        int x;
        A(int p)
        {
                x=p;
        }
        void show()
        {
                System.out.println("Class A");
                System.out.println("X= "+x);
        }
}
class B extends A
{
        int y;
        B(int p,int q)
```

```
        {
                super(p);
                y=q;
        }
        public void display()
        {
                System.out.println("Class B");
                System.out.println("Y= "+y);
        }
}
class C extends B
{
        int z;
        C(int p,int q,int r)
        {
                super(p,q);
                z=r;
        }
        public void message()
        {
                System.out.println("Class C");
                System.out.println("Z= "+z);
        }
}
class Inheritance_Demo
{
        public static void main(String args[])
        {
                C c=new C(10,20,30);
                c.show();
                c.display();
                c.message();
        }
}
```

Output:

```
D:\java_prog>java Inheritance_Demo
Class A
X= 10
Class B
Y= 20
Class C
Z= 30
```

Method Overriding

In inheritance, if both parent class and child class having same method signatures, then child class methods will not allow the parent class methods to be executed. This is called "Member over-riding".

1. This is used to execute its own behaviour by hiding the implementations of parent class.
2. But in some cases, there is a need to call parent class methods through child class. Then we have to use 'super' keyword before the parent class member method.
3. It refers to the parent class up to 1 level back because java supports single-level inheritance only.
4. 'super 'keyword cannot be applied with in static methods.
5. We cannot use the objects to call super keyword.

Example1:

```
class A
{
        int x;
        A()
        {
                System.out.println("A Constructor called");
                x=100;
        }

}
class B extends A
{
        int x;
        B(int x)
        {       super();
                this.x=x;
        }
        public void show()
        {
                System.out.println("x from A class is: "+super.x);
                System.out.println("x from B class is: "+x);
        }
}
class super_Demo
{
  public static void main(String args[])
        {
          B b=new B(200);
          b.show();
        }
}
```

Output:

```
D:\java_prog>java super_Demo
A Constructor called
x from A class is: 100
x from B class is: 200
```

Example2:

```
import java.io. *;
class Number
{
        protected int x;
        BufferedReader b = new BufferedReader (new InputStreamReader (System.in));
        public void getNo ()throws Exception
        {
                System.out.println("Enter a No");
                x = Integer.parseInt(b.readLine ());
        }
```

```
        void show ()
        {
                System.out.println("Decimal:"+x);
        }
        public static void main (String args [])
        {
                Number n = new number ();
                Hexa h = new Hexa ();
                Octal o = new Octal ();
                n.getNo ();
                h.getNo ();
                o.getNo ();
                n.show ();
                h.show ();
                o.show ();
        }
}
class Hexa extends Number
{
        void show ()
        {
                super.show ();
                System.out.println("Hexa:"+Integer.toHexString(x));
        }
}
class Octal extends Number
{
        void show()
        {
                super.show();
                System.out.println("Octal:" + Integer.toOctalString(x));
        }
}
```

final keyword

1. To create constants in java. (Just like #define in c)
   Eg: final float pi = 3.14f;

2. To avoid Member Over-Riding, we have to use final keyword before the parent class member method.

```
class parent
{
        final void show () { }
}
class child extends parent
{
        void show ()                    // error is displayed
        {
                }
}
```

3. To make a class as terminal class i.e. if you use final keyword before the class name, we cannot create child class to that class.

```
final class parent
{

}
class child extends parent              //error can not be inherited
{

 }
```

Dynamic method dispatching:

1. In the inheritance, if both parent and child classes having the same function signature then parent class object is able to call those methods in both parent and child parent object is able to store instance of child classes.
2. Single parent class object is overloaded with different child class instance at different times, and it will invoke the related methods according to child class instance, which is held in that object at runtime. This is called "**Dynamic Method Dispatching**".
3. The purpose of dynamic method dispatching is used to hide the some of the services of child class to provide the protection to the child class.
4. Parent class object cannot invoke the methods, which do not have same signature in parent and child.
5. To know the instance which is held in the parent class object, we have to use "**instanceof**" keyword.
6. If we want to check the instance, the parent class instance condition must be last condition.
7. Dynamic polymorphism is also called as dynamic bind or late binding or run-time poly morphism.

Example:

```
class Number
{       protected int x;
        BufferedReader b = new BufferedReader (new InputStreamReader (System.in));
        public void getNo()throws Exception
        {
                System.out.println("Enter a No");
                x = Integer.parseInt(b.readLine ());
        }
        void show()
        {
                System.out.println("Decimal:"+x);
        }
        public static disp(Number N)throws Exception
        {
                N.get no();
                If (N instanceof Hexa)
                System.out.println("Hexa Instance");
                else if (N instanceof Octal)
                System.out.println ("Octal Instance");
                else
                System.out.println("Number Instance");
                N.show ();
        }
        public static void main(String args [])throws Exception
        {
                Disp(new Number ());
                Disp(new Hexa ());
                Disp(new Octal ());
        }
}

class Hexa extends Number
{
        void show ()
        {
                super.show ();
                System.out.println("Hexa:"+Integer.toHexString(x));
        }
}
```

```
class Octal extends Number
{
        void show()
        {
                super.show();
                System.out.println("Octal:" + Integer.toOctalString(x));
        }
}
```

	Instance	o/p	I/p
N	Number	13	13
N	Hexa	15	15
N	Octal	8	10

abstract class

1 In some of the cases, it is not possible to give definition to the some of the methods.
2 Even though you are trying to give definition to the methods, it is meaningless. Such types of methods are called **abstract methods**.
3 The class, which contains these abstract methods, will be treated as **abstract class**.
4 The abstract methods only specify what to do, but not how to do. Abstract class may or may not have non-abstract or concrete methods
5 We cannot create instance to the abstract class. But we can-create objects to the abstract class and it is able to store instances of child class
6 So abstract classes are always treated as parent classes
7 We have to use '**abstract**' keyword to make the class or method as abstract.

Example1:

```
import java.io.*;
abstract class Shape
{
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        float s1,s2,a;
        final float pi=3.14f;
        public abstract void get_Input()throws Exception;
        public abstract void Cal_Area();
        public void show_Area()
        {
                System.out.println("Area is:"+a);
        }
}
class Rect extends Shape
{
        public void get_Input()throws Exception
        {
                System.out.println("Enter L and B values");
                s1=Float.parseFloat(br.readLine());
                s2=Float.parseFloat(br.readLine());
        }
        public void Cal_Area()
        {
                a=s1*s2;
        }
}
class Circle extends Shape
{
        public void get_Input()throws Exception
        {
                System.out.println("Enter radius of the Circle");
                s1=Float.parseFloat(br.readLine());
        }
        public void Cal_Area()
        {
                a=pi*s1*s1;
        }
}
class Mainclass
{
        public static void main(String args[])throws Exception
        {
                Shape s;

                s=new Rect();
                System.out.println("Rectangle:");
                s.get_Input();
                s.Cal_Area();
                s.show_Area();
                System.out.println("============");
```

```
                s=new Circle();
                System.out.println("Circle:");
                s.get_Input();
                s.Cal_Area();
                s.show_Area();
                System.out.println("==============");
        }
}
```

Example2:

```
abstract class Vehicle
{
        public abstract int get_No_Wheels();
        public abstract int seating_Capacity();
}
class Bike extends Vehicle
{
        public int get_No_Wheels()
        {
                return 2;
        }
        public int seating_Capacity()
        {
                return 2;
        }
}
class Auto extends Vehicle
{
        public int get_No_Wheels()
        {
                return 3;
        }
        public int seating_Capacity()
        {
                return 4;
        }
}
class Car extends Vehicle
{
        public int get_No_Wheels()
        {
                return 4;
        }
        public int seating_Capacity()
        {
                return 5;
        }
}
class Mainclass
{
        public static void main(String args[])
        {
                Vehicle v;
                int w,c;
                v=new Bike();
                System.out.println("==============");
                System.out.println("Bike:");
                w=v.get_No_Wheels();
                c=v.seating_Capacity();
                System.out.println("No of Wheels: "+w);
```

```
                System.out.println("Seating Capacity: "+c);
                System.out.println("==============");

                v=new Auto();
                System.out.println("==============");
                System.out.println("Auto:");
                w=v.get_No_Wheels();
                c=v.seating_Capacity();
                System.out.println("No of Wheels: "+w);
                System.out.println("Seating Capacity: "+c);
                System.out.println("==============");

                v=new Car();
                System.out.println("==============");
                System.out.println("Car:");
                w=v.get_No_Wheels();
                c=v.seating_Capacity();
                System.out.println("No of Wheels: "+w);
                System.out.println("Seating Capacity: "+c);
                System.out.println("==============");
        }
}
```

Interface

1 Interface is a mechanism to provide the security to the server-side application by hiding some of the services and gives only those services which are required to the end-users.
2 Interface itself is an abstract and all the methods in interface are public abstract by default
3 We cannot create instance to the interface but we can create objects to the interface
4 We have to implement the methods of the interface by using a class. The class which implements interface must implement all the methods in the interface otherwise we have to make the class as abstract
5 Interface object is able to store the instance of the class, which implements it.
6 Interface is used to provide multiple inheritance type of mechanism so that we can implement more than one interface by using a class.
7 One interface may be extended from more than one interface.
8 Interface object is able to call only those methods, which are having same method signature. Interfaces don't have concrete methods and variables.
9 Interface only contains constants.
10 'super' keyword cannot be applied to the interface methods.

Syntax:

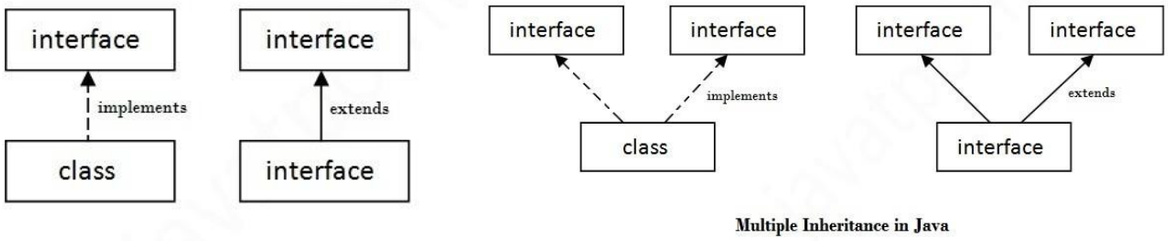
```
interface <interface-name>
{
    declaration of methods
}
class <class-name> implements <interface names>
{
    definition of methods in the interface
    definition of its own methods
}
```

Example:

```
interface FactInf
{
    int fact (int x);
}
class Dev1 implements FactInf          //iteration way of calculating
{
    int fact (int x)
    {
            Int I,f = 1;
            For (I = 1;I< = x;I++)
            F * = I;
            Return f;
    }
}
class Dev2 implements FactInf                  //recurrence way of calculating
{
    int fact (int x)
    {
            int f = 1;
            if (x> = 1)
                    f = x*fact (x-1);
            return f;
    }
}
class MainClass
{
    public static void main (String args [])
    {
            FactInf fi;
            fi = new Dev1();
            System.out.println(fi.fact(5));
            Fi =new Dev2 ();
            System.out.println(fi.fact(8));
    }
}
```

Interface hides other functions in class from the end-user. He can access only some functions in the class as per his need. So it's better to create object for the interface rather than the class.

Realationship between interface and class:



Multiple Inheritance in Java

Example2:

```
interface Inf1
{
        void method1();
        void method2();
}
class Provider implements Inf1
{
        public void method1()
        {
                System.out.println("Hello!");
        }
        public void method2()
        {
                System.out.println("Good Morning to All");
        }
}
class Mainclass
{
        public static void main(String args[])
        {
                Inf1 i1;

                i1=new Provider();
                i1.method1();
                i1.method2();
        }
}
```

Example3:

```
interface Inf1
{
        void m1();
        void m2();
}
interface Inf2
{
        void m2();
        void m3();
}
class Service_Provider implements Inf1,Inf2
{
        public void m1()
        {
                System.out.println("Hello!");
        }
        public void m2()
        {
                System.out.println("Welcome to ");
        }
        public void m3()
```

```
        {
                System.out.println("Java Programming");
        }
}
class Mainclass
{
        public static void main(String args[])
        {
                Inf1 i1;
                Inf2 i2;
                i1=new Service_Provider();
                i1.m1();
                i1.m2();
                i1.m3();                        // Error m3() is not available in Inf1

                i2=new Service_Provider();
                i2.m1();                        // Error m1() is not available in Inf2
                i2.m2();
                i2.m3();
        }
}
```

Example4:

```
interface Inf1
{
        void m1();
}
interface Inf2 extends Inf1
{
        void m2();
}
class Service_Provider implements Inf2
{
        public void m1()
        {
                System.out.println("Method m1() called");
        }
        public void m2()
        {
                System.out.println("Method m2() called");
        }
}
class Mainclass
{
        public static void main(String args[])
        {
        Inf2 i;

                i=new Service_Provider();
                i.m1();
                i.m2();
        }
}
```

Example5:

```
interface Inf1
{
        void m1();
}
interface Inf2
{
        void m2();
```

```
}
interface Inf3 extends Inf1,Inf2
{
	void m3();
}

class Service_Provider implements Inf3
{
	public void m1()
	{
		System.out.println("Method m1() called");
	}
	public void m2()
	{
		System.out.println("Method m2() called");
	}
	public void m3()
	{
		System.out.println("Method m3() called");
	}
}
class Mainclass
{
	public static void main(String args[])
	{
		Inf3 i;

		i=new Service_Provider();
		i.m1();
		i.m2();
		i.m3();
	}
}
```

Example6:

```
import java.io.*;
interface AcctInf
{
	void open(String acno, char type,double amt);
	void deposit(String acno, double amt);
	void withdraw(String acno, double amt);
	void transfer(String sacno, String dacno, double amt);
	double getBal(String acno);
	void showAccounts();
}
class Account
{
	private String acno;
	private char type;
	private double bal;
	public void setAcno(String ano)
	{
		acno = ano;
	}
	public void setType(char c)
	{
		type=c;
	}
	public void setBal(double bal)
	{
		this.bal=bal;
	}
```

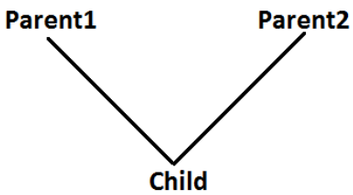
```
        String getAcno()
        {
                return acno;
        }
        double getBal()
        {
                return bal;
        }
        char getType()
        {
                return type;
        }
}

class Bank implements AcctInf
{
        int i= 0,j,found=0;
        static final int nr =50;
        Account ac[] = new Account[Bank.nr];

        public void open(String acno, char type,double amt)
        {
                ac[i]=new Account();

                ac[i].setAcno(acno);
                ac[i].setBal(amt);
                ac[i].setType(type);
                i++;
        }

        public double getBal(String acno)
        {
                for(j=0; j<i; j++)
                        if (ac[j].getAcno().equals(acno))
                                return ac[j].getBal();
                System.out.println("Invalid Acno");
                return -1;
        }

        public void deposit(String acno, double amt)
        {   found=0;
            for(j=0;j<i;j++)
                {       if(ac[j].getAcno().equals(acno))
                        {   found=1;
                                break;
                        }
                }
                if(found==1)
                        ac[j].setBal(ac[j].getBal()+amt);
                else
                        System.out.println("Invalid Account Number");

        }

        public void withdraw(String acno, double amt)
        {
                found=0;
           for(j=0;j<i;j++)
                {       if(ac[j].getAcno().equals(acno))
                        {   found=1;
                                if(ac[j].getBal()-amt>=0)
                                        ac[j].setBal(ac[j].getBal()+amt);
```

```
                        else
                                System.out.println("Insufficient Balance in your account");

                        break;
                }
        }
        if(found==0)
                System.out.println("Invalid Account Number");

}

public void transfer(String sacno, String dacno, double amt)
{
  int found1=-1,found2=-1;
  for(j=0;j<i;j++)
        {
          if(ac[j].getAcno().equals(sacno))
                found1=j;
          else if(ac[j].getAcno().equals(dacno))
                found2=j;

          if(found1!=-1 && found2!=-1)
                break;
        }
        if(found1==-1 || found2==-1)
                System.out.println("Invalid Account numbers");
        else if(ac[found1].getBal()-amt>=0)
        {       ac[found1].setBal(ac[found1].getBal()-amt);
                ac[found2].setBal(ac[found2].getBal()+amt);
        }
        else
                System.out.println("Insufficient amount to transfer");
}

public void showAccounts()
{
        for(j=0;j<i;j++)
        {
                System.out.println("Account"+(j+1)+" details");
                System.out.println(ac[j].getAcno());
                System.out.println(ac[j].getType());
                System.out.println(ac[j].getBal());
                System.out.println("==========================");
        }
}

public static void main(String args[])throws Exception
{
        AcctInf acc=new Bank();
        String acno,dacno;
        char type;
        double bal;
        int choice;
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

        while(true)
        {
                System.out.println("1. Open Account\n2. Deposit\n3. WithDraw\n4. Transfer
                Amount\n5.Check Balance\n6. Show Accounts\n7. Exit");
                choice=Integer.parseInt(br.readLine());
                switch(choice)
                {
```

```
                case 1:
                        System.out.println("Enter Account Number, Type of Account and
                        balance");
                        acno=br.readLine();
                        type=(char)br.read();
                        br.readLine();
                        bal=Double.parseDouble(br.readLine());
                        acc.open(acno,type,bal);
                        break;
                case 2:
                        System.out.println("Enter Account number and Balance to Deposit");
                        acno=br.readLine();
                        bal=Double.parseDouble(br.readLine());
                        acc.deposit(acno,bal);
                        break;
                case 3:
                        System.out.println("Enter Account number and Balance to Withdraw");
                        acno=br.readLine();
                        bal=Double.parseDouble(br.readLine());
                        acc.withdraw(acno,bal);
                        break;
                case 4:
                        System.out.println("Enter Source ACNO, Destination ACNO and
                        Balance to transfer");
                        acno=br.readLine();
                        dacno=br.readLine();
                        bal=Double.parseDouble(br.readLine());
                        acc.transfer(acno,dacno,bal);
                        break;
                case 5:
                        System.out.println("Enter the acno to get the balance");
                        acno=br.readLine();
                        System.out.println("Balance is: "+acc.getBal(acno));
                        break;

                case 6:
                        acc.showAccounts();
                        break;
                case 7:
                        System.exit(0);
                        }
                }

        }
}
```

Multiple Inheritance in JAVA

Having more than one Parent class at the same level is called multiple inheritance.



Any class can extends only one class at a time and can't extends more than one class simultaneously hence java won't provide support for multiple inheritance.

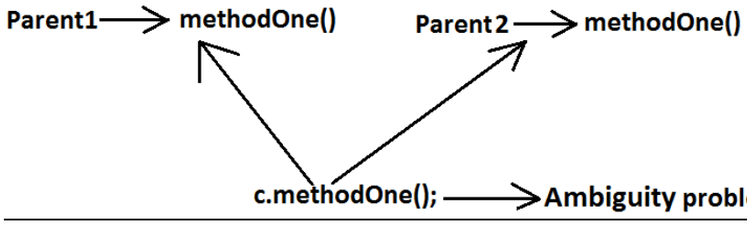
```
class A{}
class B{}
class C extends A,B
{}
```
(invalid)

But an interface can extends any no. Of interfaces at a time hence java provides support for multiple inheritance through interfaces.

```
interface A{}
interface B{}
interface C extends A,B{}
```

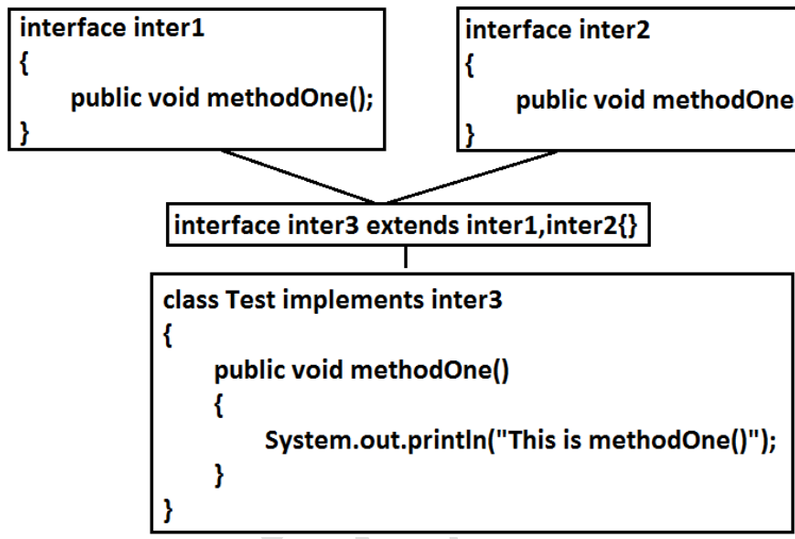
Why java won't provide support for multiple inheritance?

There may be a chance of raising ambiguity problems.



Why ambiguity problem won't be there in interfaces?

Interfaces having dummy declarations and they won't have implementations hence no ambiguity problem.



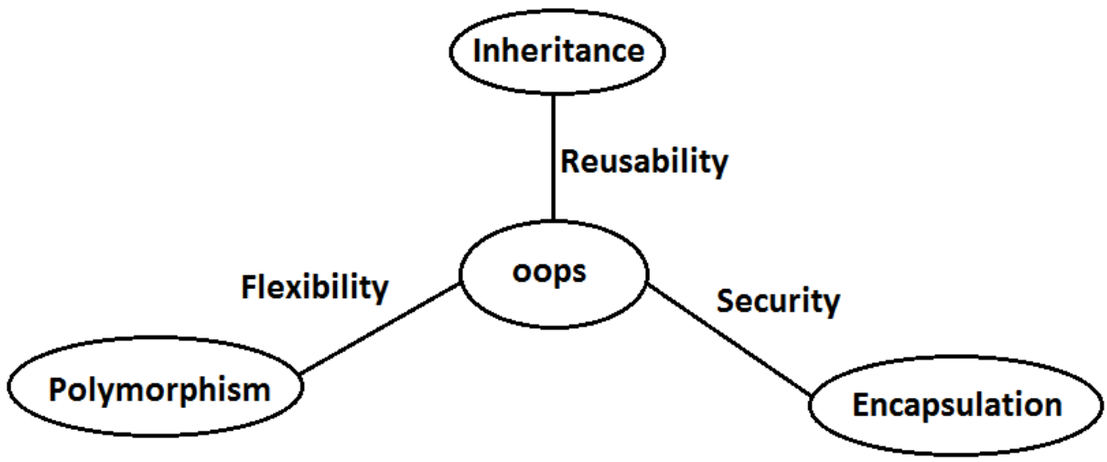
```
interface inter1
{
    public void methodOne();
}
```

```
interface inter2
{
    public void methodOne();
}
```

```
interface inter3 extends inter1,inter2{}
```

```
class Test implements inter3
{
    public void methodOne()
    {
        System.out.println("This is methodOne()");
    }
}
```

3 Pillars of OOPS:



PACKAGE

1. Package is a collection of java classes, interfaces and sub packages. It is used to separate the set of classes according to their category and usage.
2. If you unite more than one class in a single package and if you want to use only some of the classes in that package, then at the time of importing the package will increase the memory wastage and compilation time so that it is better to distribute set of classes among different packages.
3. It is used to provide the security.

Rules to create a package:

1. We have to write the classes with the "package" statement as a first statement to those classes which are required to place into packages.
2. We have to make the class as public, to access that class outside of the package.
3. We can write any number of classes with in a file but only one class must be the public.
4. If you want to make more than one class as public we have to place each and every class in separate files.
5. File name must be the name of the public class.
6. A class many have more than one method, but at least one method must be public.

Types of Packages:

1. **Built in packages**

   Java has already defined some packages and included that in java software, these packages are known as built-in packages or predefined packages. These packages contain a large number of classes and interfaces useful for java programmers for different requirements.
   There are many built-in packages such as

   java.lang – language fundamentals package
   java.io – contains classes working with input and output handling.
   java.util – contains classes working with collections
   java.sql – contains classes working with database
   java.net – contains classes working with network application
   java.awt – deals with graphical elements in java

Example - 1:

```
import java.lang.*;
class MathDemo
{
        public static void main(String args[])
        {
                 int x=27,y=25;

                // max and min method
                System.out.println("Maximum is: "+Math.max(x,y));
                System.out.println("Minimum is: "+Math.min(x,y));

                // sqrt(), cbrt()
                System.out.println("Square root: "+Math.sqrt(y));
                System.out.println("Cube root: "+Math.cbrt(x));

               // ceil() - nearest integer which is greater than given number


                System.out.println("Ceil value: "+Math.ceil(5.3));
              // nearest integer which is less than given number
                System.out.println("Floor value: "+Math.floor(5.3));

               // log10(), log()
                System.out.println("Log Value is:"+Math.log(2));      // 1
                System.out.println("Log10 Value is:"+Math.log10(10));  // 1

                // pow(x,y) => x^y
                System.out.println("pow(3,4) is:"+Math.pow(3,4));

               // round(float) ==> rounded integer value
```

```
                System.out.println("round(2.689f): "+Math.round(2.689f)); //
                System.out.println("round(2.489f): "+Math.round(2.489f));

            // abs(int)
             System.out.println("abs(-39): "+Math.abs(-39));   // 39

             System.out.println(Math.rint(100.99));
             System.out.println(Math.random());
             System.out.println(Math.floorDiv(5,2));
             System.out.println(Math.floorDiv(5,3));
      }
}
```

Example – 2:

```
import java.util.Date;
class DateDemo
{
    public static void main(String args[])
    {
        Date d1=new Date();

        long millis=System.currentTimeMillis();
        java.sql.Date d2=new java.sql.Date(millis);

        System.out.println("Date in util package is: "+d1);
        System.out.println("Date in sql package is: "+d2);
    }
}
```

```
D:\Java_Programs>java DateDemo
Date in util package is: Fri Oct 23 08:19:23 IST 2020
Date in sql package is: 2020-10-23
```

2. **User defined packages**

   User-defined packages are those which are developed by users in order to group related classes, interfaces and sub packages. With the help of an example program, let's see how to create packages, compile Java programs inside the packages and execute them.

   Steps involved in user defined package creation:

   1. Creation of user defined package file
   2. Compilation of user defined package file
   3. Setting of class path
   4. Importing of user defined package in another application.

Step 1: package creation

Filename: Display.java

```
1  package aec.ece;
2  public class Display
3  {
4     String msg;
5     public void sent_Message(String value)
6     {
7        msg=value;
8        System.out.println("Messeage Sent: "+msg);
9     }
10    public void receive_Message()
11    {
12       System.out.println("Message Received: "+msg);
13    }
14
15    public void wish(String name)
16    {
17       System.out.println("Welcome to ECE Department");
18       System.out.println("======"+name+"======");
19    }
20 }
```

Step 2: compilation of package file

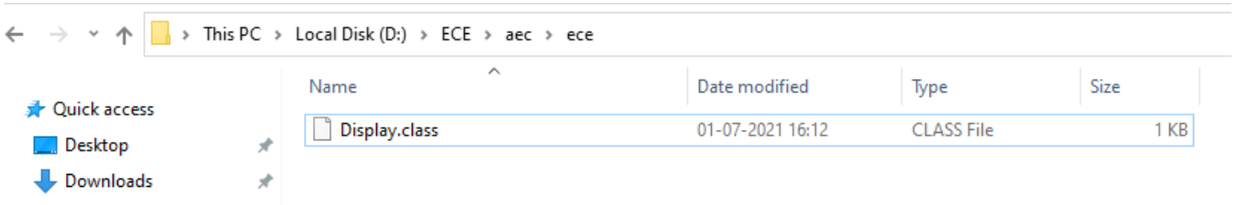
Syntax:

javac –d <path> filename.java

<path> => where to store your package in your computer.

```
D:\ECE-A\package>javac -d D:\ECE Display.java

D:\ECE-A\package>
```

Compile successfully and package aec.ece created in D:\ECE folder. So Display.class was created in D:\ECE\aec\ece folder as show below.



Step – 3: Setting of class path

classpath is a parameter in the **Java** Virtual Machine or the **Java** compiler that specifies the location of user-defined classes and packages. The parameter may be set either on the command-line, or through an environment variable. If you not set the class path properly then the compiler generates an error like package does not exist.

Syntax:

set classpath=%classpath%;.;<path>;

<path> => specify the path where your package was stored.

```
D:\ECE-A\package>set classpath=%classpath%;.;D:\ECE;

D:\ECE-A\package>
```

Step – 4: How to import a user defined package in another application

```
1  import java.util.*;          // importing built-in package
2  import aec.ece.Display;      // importing user defined package
3  class Sample
4  {
5      public static void main(String args[])
6      {
7          Scanner sc=new Scanner(System.in);
8          Display d=new Display();
9          String info;
10
11         d.wish("Srinu_M");
12
13         System.out.println("Enter any information");
14         info=sc.nextLine();
15         d.sent_Message(info);
16
17         d.receive_Message();
18     }
19 }
```

Output:

```
D:\ECE-A\package>javac Sample.java

D:\ECE-A\package>java Sample
Welcome to ECE Department
=====Srinu_M=====
Enter any information
Hello How r you?
Messeage Sent: Hello How r you?
Message Received: Hello How r you?
```

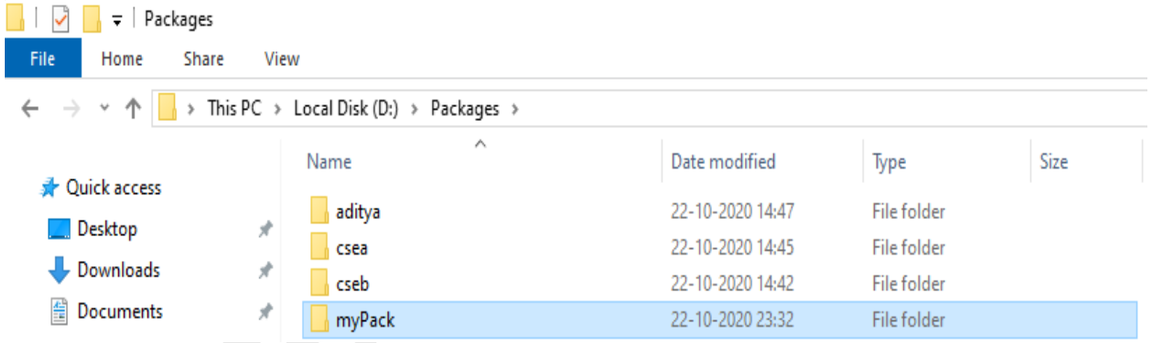
Example – 2:

Step – 1:

```
package myPack;
public class Compare
{
    public int getMax(int n, int m)
    {
        if(n>m)
            return n;
        else
            return m;
    }
    public int getMin(int n,int m)
    {
        if(n<m)
            return n;
        else
            return m;
    }
    public void getEqual(int n,int m)
    {
        if(n==m)
            System.out.println("Equal");
        else
            System.out.println("Not Equal");
    }
}
```

Step – 2:

```
D:\Java_Programs>javac -d D:\Packages Compare.java

D:\Java_Programs>
```



Step – 3 & 4:

```
1  import java.util.Scanner;       // built in package
2  import myPack.Compare;     // user defined package
3  class NumberDemo
4  {
5      public static void main(String args[])
6      {
7          int n1,n2;
8          Scanner sc=new Scanner(System.in);
9          Compare c=new Compare();
10         System.out.println("Enter any two numbers");
11         n1=sc.nextInt();
12         n2=sc.nextInt();
13
14         int max=c.getMax(n1,n2);
15         int min=c.getMin(n1,n2);
16         c.getEqual(n1,n2);
17
18         System.out.println("Maximu: "+max);
19         System.out.println("Minimum: "+min);
20     }
21 }
```

```
D:\Java_Programs>set classpath=;.;D:\Packages;%classpath%;

D:\Java_Programs>javac NumberDemo.java

D:\Java_Programs>java NumberDemo
Enter two numbers
25
36
Not Equal
Maximum value is:36
Minimum value is:25
```

Example – 3:

package aec.math;

```
public class Arithmetic
{
        public int add(int x,int y)
         {
                return (x+y);
         }
         public int sub(int x,int y)
        {
                 return (x-y);
        }
        public int mul(int x,int y)
        {
                return (x*y);
        }
        public int div(int x,int y)
        {
                return x/y;
        }
        public int mod(int x,int y)
        {
                return x%y;
        }
}
```

Compilation:

javac –d D:\ECE Arithmetic.java

```
1  import java.util.Scanner;
2  import aec.math.Arithmetic;
3  class Operations
4  {
5     public static void main(String args[])
6     {
7        Scanner sc=new Scanner(System.in);
8        Arithmetic m=new Arithmetic();
9        int n1,n2;
10       System.out.println("Enter any two integers");
11       n1=sc.nextInt();
12       n2=sc.nextInt();
13
14       int s=m.add(n1,n2);
15       int diff=m.sub(n1,n2);
16       int p=m.mul(n1,n2);
17       int d=m.div(n1,n2);
18       int m_val=m.mod(n1,n2);
19
20       System.out.println("Sum = "+s);
21       System.out.println("Diff = "+diff);
22       System.out.println("Product = "+p);
23       System.out.println("Div = "+d);
24       System.out.println("Mod = "+m_val);
25    }
26 }
```

```
D:\ECE-A\package>set classpath=%classpath%;.;D:\ECE;

D:\ECE-A\package>javac Operations.java

D:\ECE-A\package>java Operations
Enter any two integers
16
12
Sum = 28
Diff = 4
Product = 192
Div = 1
Mod = 4
```

Access Modifiers:

The access modifier in Java specifies the accessibility or scope of a field, method, constructor, or class.

There are four types of Java access modifiers:

Private: The access level of a private modifier is only within the class. It cannot be accessed from outside the class. **[class level]**

Default: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default. **[package level]**

Protected: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package. **[default + child classes in other packages]**

Public: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package. **[Everywhere]**

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Private: The private access modifier is accessible only within the class.

Example:

```
1  class A
2  {
3     private int data=40;
4     private void msg()
5        {
6           System.out.println("Hello java");
7        }
8  }
9
10 class Simple
11 {
12    public static void main(String args[])
13       {
14          A obj=new A();
15          System.out.println(obj.data);//Compile Time Error
16          obj.msg();//Compile Time Error
17       }
18 }
```

```
D:\ECE-A\package>javac Private_Demo.java
Private_Demo.java:15: error: data has private access in A
                        System.out.println(obj.data);//Compile Time Error

                                                   ^
Private_Demo.java:16: error: msg() has private access in A
                        obj.msg();//Compile Time Error
                           ^
2 errors
```

Default: If you don't use any modifier, it is treated as default by default. The default modifier is accessible only within package. It cannot be accessed from outside the package. It provides more accessibility than private. But, it is more restrictive than protected, and public.

Example – 1:

```
1 package pack1;
2 public class A
3 {
4    void m1()
5    {
6       System.out.println("m1 method");
7    }
8 }
```

```
1 package pack1;
2 public class D
3 {
4    public static void main(String args[])
5    {
6       A a=new A();
7       a.m1();
8    }
9 }
```

```
D:\ECE-A\package>javac -d D:\ECE A.java

D:\ECE-A\package>javac -d D:\ECE D.java

D:\ECE-A\package>java pack1.D
m1 method
```

Example – 2:

```
1 package pack1;
2 public class A
3 {
4    void m1()
5    {
6       System.out.println("m1 method");
7    }
8 }
```

```
1 package pack2;
2 import pack1.A;
3 public class B
4 {
5    public static void main(String args[])
6    {
7       A a=new A();
8       a.m1();
9    }
10 }
```

```
D:\ECE-A\package>javac -d D:\ECE A.java

D:\ECE-A\package>javac -d D:\ECE B.java
B.java:14: error: m1() is not public in A; cannot be accessed from outside
 package
                a.m1();
                 ^
1 error
```

Protected: The protected access modifier is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

Example -1:

```
1 package pack1;
2 public class A
3 {
4    protected void m1()
5    {
6       System.out.println("protected method in A");
7    }
8 }
```

```
1 package pack2;
2 import pack1.A;
3 public class B extends A
4 {
5    public static void main(String args[])
6    {
7       B b=new B();
8       b.m1();
9    }
10 }
```

```
D:\ECE-A\package>javac -d D:\ECE A.java

D:\ECE-A\package>javac -d D:\ECE B.java

D:\ECE-A\package>java pack2.B
protected method in A
```

Example – 2:

```
1  package pack1;
2  public class A
3  {
4     protected void m1()
5     {
6        System.out.println("protected method in A");
7     }
8  }
```

```
1  package pack2;
2  import pack1.A;
3  public class B
4  {
5     public static void main(String args[])
6     {
7        A a=new A();
8        a.m1();
9     }
10 }
```

```
D:\ECE-A\package>javac -d D:\ECE A.java

D:\ECE-A\package>javac -d D:\ECE B.java
B.java:8: error: m1() has protected access in A
                a.m1();
                 ^
1 error
```

public: The public access modifier is accessible everywhere. It has the widest scope among all other modifiers.

```
1  package pack1;
2  public class A
3  {
4     public void m1()
5     {
6        System.out.println("public method in A");
7     }
8  }
```

```
1  package pack2;
2  import pack1.A;
3  public class B
4  {
5     public static void main(String args[])
6     {
7        A a=new A();
8        a.m1();
9     }
10 }
```

```
D:\ECE-A\package>javac -d D:\ECE A.java

D:\ECE-A\package>set classpath=%classpath%;.;D:\ECE;

D:\ECE-A\package>javac -d D:\ECE B.java

D:\ECE-A\package>java pack2.B
public method in A
```

```
//Product.java
package com.aditya.cse;
public class Product
{
	private String pno;
	private int qoh;
	private double cost;
	public void setPno(String pno)
	{
		this.pno=pno;
	}
	public String getPno()
	{
		return pno;
	}
	public void setCost(double cost)
	{
		this.cost =cost;
	}
	public double getCost()
	{
		return cost;
	}
	public void setQoh(int qoh)
	{
		this.qoh =qoh;
	}
	public int getQoh()
	{
		return qoh;
	}
}
// ProdInf.java
package com.aditya.cse;
import com.aditya.cse.Product;
public interface ProdInf
{
	void addProd (Product p);
	Product getProd (String pno);
}
//ProdImp.java
package com.aditya.cse;
import com.aditya.cse.Product;
import com.aditya.cse.ProdInf;
public class ProdImp implements ProdInf
{
	final int np=10;
	Product list[] = new Product[np];
	int i=0,j;
	public void addProd (Product p)
	{
		list [i]=p;
		i++;
	}
	public Product getProd(String pno)
	{
		for(j=0;j<i;j++)
		if (list[j].getPno().equals(pno))
			return list[j];
		System.out.println("Invalid pno");
		return null;
	}
```

```
}
// Shop.java
import com.aditya.cse.Product;
import com.aditya.cse.ProdInf;
import com.aditya.cse.ProdImp;
import java.io.*;
class Shop
{
	public static void main(String args[]) throws Exception
	{
		int ch;
		BufferedReader b=new BufferedReader(new InputStreamReader(System.in));
		Product p;
		ProdInf pi=new ProdImp();
		while(true)
		{
			System.out.println("1.Add\n2.GetProd\n3.Exit");
			System.out.println("Enter ur choice:");
			ch= Integer.parseInt(b.readLine());
			switch (ch)
			{
				case 1:
					System.out.println("Enter pno, cost, qoh");
					p=new Product();
					p.setPno(b.readLine());
					p.setCost(Double.parseDouble(b.readLine()));
					p.setQoh(Integer.parseInt(b.readLine()));
					pi.addProd(p);
					break;

				case 2:
					System.out.println("Enter pno to find");
					p = pi.getProd(b.readLine());
					if(p!=null)
					{
						System.out.println("Pno:"+p.getPno());
						System.out.println("Cost:"+p.getCost());
						System.out.println("Qoh:"+p.getQoh());
					}
				break;
				case 3:
					System.exit(0);
			}
		}
	}
}
```

Compilation:

```
set classpath = .;d:\test; %class path%;
javac -d d:\test  Product java
javac -d d:\test  ProdInf.java
javac -d d:\test  ProdImp.java
javac  Shop. Java
```

Run:

```
java Shop
```

If more than one package have same class name, we should specify the package name explicitly.
Eg: sample.Test t1 = new sample.Test();
dummy.Test t2 = new dummy.Test();

Where d:
Test

Sample

Test. class

Dummy
Test. Class

Compilation of one of the tests
Java c –d d:\test Test. java

```
import sample.test;
import dummy.test;
class mainclass
{
        public static void main (string args [])
        {
                sample.test t1 = new sample.test();
                dummy. Test t2 = new dummy.test();
        }

}
```

To create the jar file:

Jar file is a collection of package, image files, html files, jsp files. It will compress all files in a single file so that using and transmitting is easy. It is like WinZip but WinZip will only work on windows platform, but jar is a platform independent file so that we can place the jar file on any platform and to work with the packages in the jar file, we have to set the class path to the jar file.

Creating the jar file:

F:\Test
Mainclass.class
Sample
First-class
Second-class
Third-class
F:\test> **jar - cvf TestFile. jar <list of files>**
Eg: jar -cvf Test.jar *.java *.class
jar -cvf Test.jar

To list out the contents of jar file
jar -tvf File.jar
t - table
jar -tvf Test.jar

To extract the jar file:
jar-xvf File.jar
set classpath = e:\proj\Test.jar

To append the new files to the existing files:

jar -uvf Test.jar *.html

EXCEPTION HANDLING

In programming languages there is a possibility to generate 3 types of errors:

1 **Compile-time errors**: These are generated because of programming syntaxes.
2 **Run-time errors**: These will be generated at the time of execution.
3 **Logical errors**: These will be generated because of the programmer's mistakes.

Exception:

This is a run-time error; it will be generated at the time of program execution. This will make the program to be terminated abnormally. Without handling of these exceptions, we cannot provide safety to the application. If any language having proper exception handling facilities will be treated as strictly written language. If any application is developed using such type of languages will be treated as '**Robust**' applications.

It is highly recommended to handle exceptions. The main objective of exception handling is graceful (normal) termination of the program.

What is the meaning of exception handling?

Exception handling doesn't mean repairing an exception. We have to define alternative way to continue rest of the program normally. This way of defining alternative is nothing but exception handling.

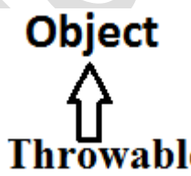
Example: Suppose our programming requirement is to read data from remote file locating at London. At runtime if London file is not available then our program should not be terminated abnormally.

We have to provide a local file to continue rest of the program normally. This way of defining alternative is nothing but exception handling.

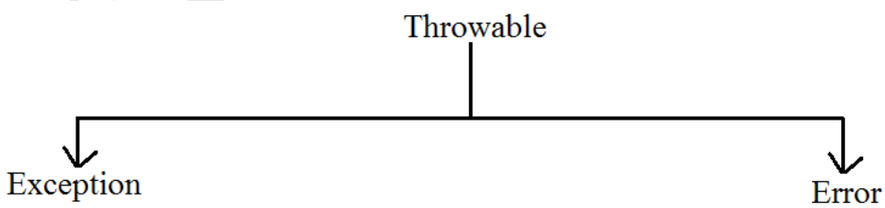
Example:

```
try
{
        read data from London file
}
catch(FileNotFoundException e)
{
        use local file and continue rest of the program normally
}
```

Exception Hierarchy:



Throwable acts as a root for exception hierarchy. Throwable class contains the following two child classes.



Exception:

Most of the cases exceptions are caused by our program and these are recoverable.

Ex : If FileNotFoundException occurs then we can use local file and we can continue rest of the program execution normally.

Error:

Most of the cases errors are not caused by our program these are due to lack of system resources and these are non-recoverable.

Ex : If OutOfMemoryError occurs being a programmer we can't do anything the program will be terminated abnormally. System Admin or Server Admin is responsible to raise/increase heap memory.

Checked Vs Unchecked Exceptions:

The exceptions which are checked by the compiler whether programmer handling or not, for smooth execution of the program at runtime, are called checked exceptions.

1. HallTicketMissingException
2. PenNotWorkingException
3. FileNotFoundException

The exceptions which are not checked by the compiler whether programmer handing or not, are called unchecked exceptions.

1. BombBlastException
2. ArithmeticException
3. NullPointerException

Note: RuntimeException and its child classes, Error and its child classes are unchecked and all the remaining are considered as checked exceptions.

Note: Whether exception is checked or unchecked compulsory it should occurs at runtime only and there is no chance of occurring any exception at compile time.

Fully checked Vs Partially checked:

A checked exception is said to be fully checked if and only if all its child classes are also checked.

Example:

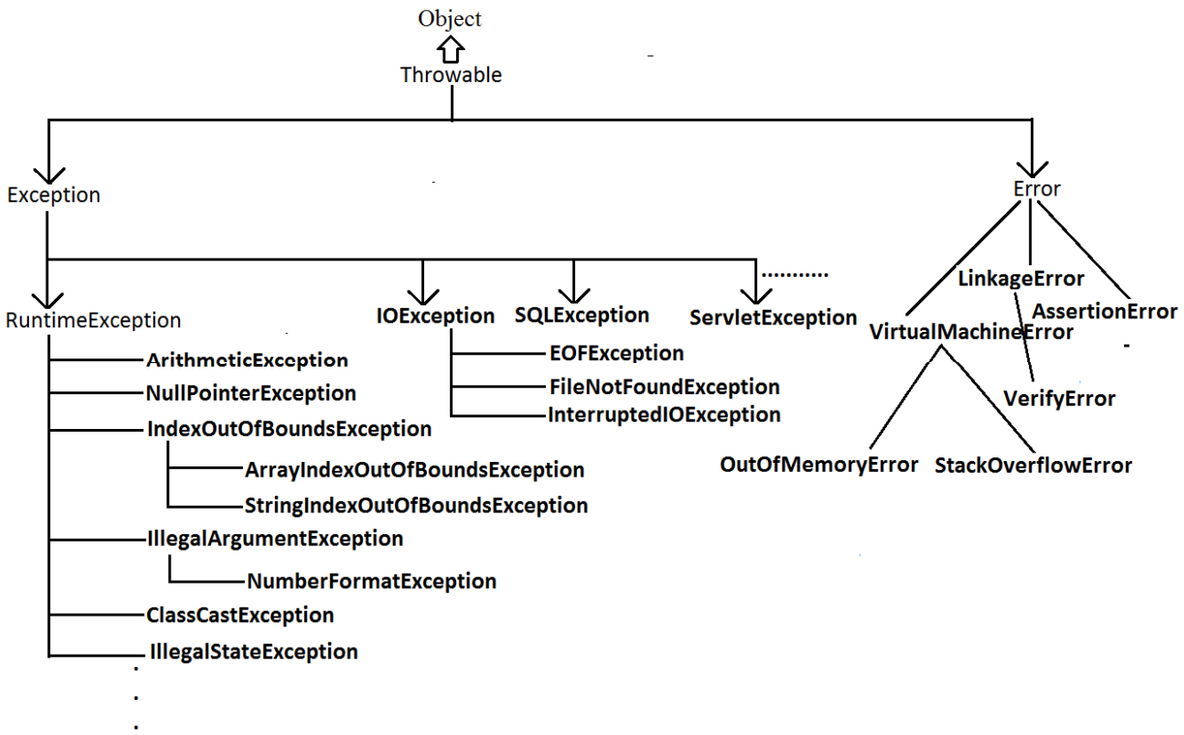
1) IOException
2) InterruptedException

A checked exception is said to be partially checked if and only if some of its child classes are unchecked.

Example: Exception

Note: The only possible partially checked exceptions in java are:

1. Throwable.
2. Exception.



Some built-in Exceptions:

1. **Arithmetic exception:** If attempt to divide with zero.
2. **NumberFormatException**: if conversion from string to number contains alphabets.
3. **ArrayIndexOutOfBoundsException**: if the index exceeds the size of the array.
4. **NullPointerException**: if you are trying to access members of a class through the object with out allocating the memory to that object.
5. **ClassCastException**: improper conversion from one class to other class-type.
6. **InterruptedException**: if one thread interrupts the other thread.
7. **IllegalThreadStateException**: if any thread enters into undefined state.
8. **UnsupportedOperationException**: improper use of methods.
9. **IOException**: if any problem occurred in the reading and writing from and to devise and memory or networks.
10. **FileNotFoundException**: if you are trying to do any operation on a file, which does not exist.
11. **EOFException**: if there are no more records.
12. **MalformedURLException**:URL-Uniform Resource Locater
    If the format of the url is wrong
    eg: correct url:-http://localhost:8080/mail/index.jsp
    http://www.yahoomail.com/index.html
13. **UnknownHostException**: if the give host-name is invalid.

Customized Exception handling by using try-catch:

- It is highly recommended to handle exceptions.
- In our program the code which may raise exception is called risky code, we have to place risky code inside try block and the corresponding handling code inside catch block.

Example:

```
try
{
        Risky code
}
catch(Exception e)
{
        Handling code
}
```

Without try catch

```
class Test
{
public static void main(String[] args){
System.out.println("statement1");
System.out.println(10/0);
System.out.println("statement3");
}
}
output:
statement1
RE:AE:/by zero
at Test.main()

Abnormal termination.
```

With try catch

```
class Test{
public static void main(String[] args){
System.out.println("statement1");
try{
System.out.println(10/0);
}
catch(ArithmeticException e){
System.out.println(10/2);
}
System.out.println("statement3");
}}
Output:
statement1
5
statement3

Normal termination.
```

Control flow in try---catch:

```
try{
        statement1;
        statement2;
        statement3;
}
catch(X e)
{
        statement4;
}
statement5;
```

Case 1:
if there is no exception.
1, 2, 3, 5 normal termination.

Case 2:
if an exception raised at statement 2 and corresponding catch block matched
1, 4, 5 normal termination.

Case 3:
if an exception raised at statement 2 but the corresponding catch block not matched
1 followed by abnormal termination.

Case 4:
if an exception raised at statement 4 or statement 5 then it's always abnormal termination of the program.

Note:

1. Within the try block if anywhere an exception raised then rest of the try block won't be executed even
   though we handled that exception. Hence we have to place/take only risk code inside try block and length of the try block should be as less as possible.
2. If any statement which raises an exception and it is not part of any try block then it is always abnormal termination of the program.
3. There may be a chance of raising an exception inside catch and finally blocks also in addition to try block.

Various methods to print exception information:

Throwable class defines the following methods to print exception information to the console.

printStackTrace():

This method prints exception information in the following format.
Name of the exception: description of exception, Stack trace

toString():

This method prints exception information in the following format.
Name of the exception: description of exception

getMessage():

This method returns only description of the exception.
Description.

Example:

```
class Test
{
public static void main(String[] args){
try
{
System.out.println(10/0);
}
catch(ArithmeticException e)
{
e.printStackTrace();
System.out.println(e);
System.out.println(e.getMessage());
}}}
```

```
java.lang.ArithmeticException: / by zero
    at Test.main(Test.java:6)
```

```
java.lang.ArithmeticException: / by zero
```

```
/ by zero
```

Note: Default exception handler internally uses printStackTrace() method to print exception information to the console.

Try with multiple catch blocks:

The way of handling an exception is varied from exception to exception. Hence for every exception type it is recommended to take a separate catch block. That is try with multiple catch blocks is possible and recommended to use.

```
try
{
.
.
.
.
}
catch(Exception e)
{
    default handler
}
```

This approach is not recommended because for any type of Exception we are using the same catch block.

```
try
{
.
.
.
.
catch(FileNotFoundException e)
{
  use local file
}
catch(ArithmeticException e)
{
  perform these Arithmetic operations
}
catch(SQLException e)
{
  don't use oracle db, use mysqldb
}
catch(Exception e)
{
  default handler
}
```

This approach is highly recommended because for any exception raise we are defining a separate catch block.

Note:

If try with multiple catch blocks present then order of catch blocks is very important. It should be from child to parent by mistake if we are taking from parent to child then we will get Compile time error saying

finally block:

- It is not recommended to take clean up code inside try block because there is no guarantee for the execution of every statement inside a try.
- It is not recommended to place clean up code inside catch block because if there is no exception then catch block won't be executed.
- We require some place to maintain clean up code which should be executed always irrespective of whether exception raised or not raised and whether handled or not handled. Such type of best place is nothing but finally block.
- Hence the main objective of finally block is to maintain cleanup code.

```
try
{
risky code
}
catch(x e)
{
handling code
}
finally
{
cleanup code
}
```

The speciality of finally block is it will be executed always irrespective of whether the exception raised or not raised and whether handled or not handled.

return Vs finally:

Even though return statement present in try or catch blocks first finally will be executed and after that only return statement will be considered. i.e finally block dominates return statement.

Example:

```
class Test
{
	public static void main(String[] args)
	{
		try
		{
			System.out.println("try block executed");
			return;
		}
		catch(ArithmeticException e)
		{
			System.out.println("catch block executed");
		}
		finally
		{
			System.out.println("finally block executed");
		}
	}
}
```

Output:

```
try block executed
Finally block executed
```

finally vs System.exit(0):

There is only one situation where the finally block won't be executed is whenever we are using System.exit(0) method.

When ever we are using System.exit(0) then JVM itself will be shutdown , in this case finally block won't be executed.

i.e., System.exit(0) dominates finally block.

Example:

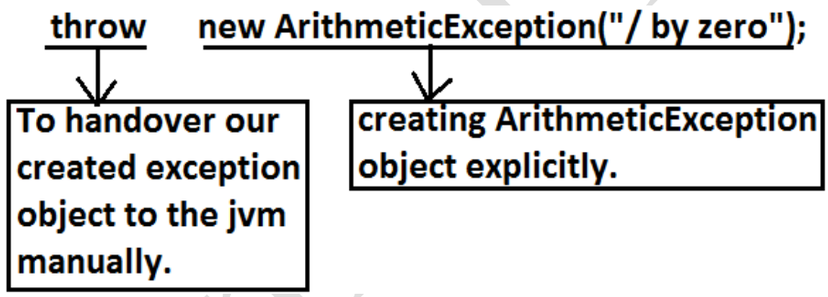
```
class Test
{
	public static void main(String[] args)
	{
		try
		{
			System.out.println("try");
			System.exit(0);
		}
		catch(ArithmeticException e)
		{
			System.out.println("catch block executed");
		}
		finally
		{
			System.out.println("finally block executed");
		}
	}
}
```

Output:

try

throw statement:

Sometimes we can create Exception object explicitly and we can hand over to the JVM manually by using throw keyword.



The result of following 2 programs is exactly same.

```
class Test
{
   public static void main(String[] args)
    {
         System.out.println(10/0);
     }
}
```

In this case creation of ArithmeticException object and handover to the jvm will be performed automatically by the main() method.

```
class Test
{
       public static void main(String[] args)
      {
            throw new ArithmeticException("/by
zero");
      }
}
```

In this case we are creating exception object explicitly and handover to the JVM manually.

Throws Statement:

In our program if there is any chance of raising checked exception then compulsory we should handle either by try catch or by throws keyword otherwise the code won't compile.

Example:

```
import java.io.*;
class Test3
{
	public static void main(String[] args)
	{
		PrinterWriter out=new PrintWriter("abc.txt");
		out.println("hello");
	}
}
```

CompileTime Error :

Unreported exception java.io.FileNotFoundException; must be caught or declared to be thrown.

Example:

```
class Test3
{
        public static void main(String[] args)
        {
                Thread.sleep(5000);
        }
}
```

Compiletime Error:

Unreported exception java.lang.InterruptedException; must be caught or declared to be thrown.

We can handle this compile time error by using the following 2 ways.

By using try catch	By using throws keyword

class Test3
{
public static void main(String[] args)
{
try{
Thread.sleep(5000);
}
catch(InterruptedException e){}
}
}
Output:
Compile and running successfully
	We can use throws keyword to delegate the responsibility of exception handling to the caller method. Then caller method is responsible to handle
	that exception.
	class Test3
	{
	public static void main(String[] args)throws InterruptedException
	{
	Thread.sleep(5000);
	}
	}
	Output:
	Compile and running successfully

Exception handling keywords summary:

1. **try:** To maintain risky code.
2. **catch:** To maintain handling code.
3. **finally:** To maintain cleanup code.
4. **throw:** To handover our created exception object to the JVM manually.
5. **throws:** To delegate responsibility of exception handling to the caller method.

Customized Exceptions (User defined Exceptions):

Sometimes we can create our own exception to meet our programming requirements. Such type of exceptions are called customized exceptions (user defined exceptions).

Example1:

1. InSufficientFundsException
2. TooYoungException
3. TooOldException

Program:

```
class TooYoungException extends RuntimeException
{
        TooYoungException(String s)
        {
                super(s);
        }
}
class TooOldException extends RuntimeException
{
        TooOldException(String s)
        {
                super(s);
        }
}
```

```
class CustomizedExceptionDemo
{
    public static void main(String[] args)
    {
        int age=Integer.parseInt(args[0]);
        if(age>60)
        {
                throw new TooYoungException("please wait some more time.... u will get best
match");
        }
        else if(age<18)
        {
                throw new TooOldException("u r age already crossed....no chance of getting
        married");
        }
        else
        {
                System.out.println("you will get match details soon by e-mail");
        }
     }
}
```

Output:

```
1)D:\Programs>java CustomizedExceptionDemo 61
Exception in thread "main" TooYoungException:
please wait some more time.... u will get best match
at CustomizedExceptionDemo.main(CustomizedExceptionDemo.java:21)

2) D:\Programs>java CustomizedExceptionDemo 27
You will get match details soon by e-mail

3) D:\Programs>java CustomizedExceptionDemo 9
Exception in thread "main" TooOldException:
u r age already crossed....no chance of getting married
at CustomizedExceptionDemo.main(CustomizedExceptionDemo.java:25)
```

Example2:

```
    class UserException extends Exception
    {
        String mesg;
        public UserException ()
        {
                msg="unknown exception";
        }
        public UserException (String err)
        {
                msg = err;
        }
        public String toString()
        {
        return "Exception Caught:"+msg;
        }
    }
    class InvalidLoginException extends UserException
    {
        public InvalidLoginException()
        {
                super ();
        }
        public InvalidLoginException(string msg)
        {
                super (msg);
        }
    }
```

```
import java.io. *;
class Account
{
    public static void main (String args [])
    {
        BufferedReader b = new BufferedReader(new InputStreamReader(System.in));
        try
        {
            String user, pass;
            System.out.println("Enter user & password");
            user=b.readLine ();
            pass=b.readLine ();
            Login l=new Login(user, pass);
            System.out.println("Login Success");
            //If exception is generated, this statement will not be displayed.
            //Control will go directly to catch block
        }catch (Exception e)
        {
            System.out.println(e);
        }
    }
}
```

MULTI THREADING

1. In the multitasking environment there is a chance to create more than one process is created it requires some set of resources.
2. Every process, which is created, is stored in different memory locations, so that if you want to shift from one process to another process, takes so much of time.
3. Switching from one process to another process is called context switch.
4. It is very expensive in terms of time.
5. In some of the cases one process may have more than one sub-process, if you make the sub-process as main process again it requires separate resources.
6. So to avoid such type of extra allocation we have to run the sub process with in the some Process memory area.
7. It will increase the performance of the system.
8. A sub-process which is placed in the unit of memory location which can be run in unit of time and is treated as unit of execution. This is called a thread.
9. It doesn't require any extra resources.
10. Switching form one thread to another thread is called thread switching.
11. It is inexpensive, when compared with the context switch.

Advantage:

If the CPU is capable to do more than one task at a time, then if you give only one task to the CPU, utilization CPU is very poor. But in multithreading, Executing several tasks simultaneously where each task is a separate independent part of the same program, it will increase the throughput of the CPU and utilization of CPU is high. By assign more than one task; we can speed up the processes

The main important application areas of multithreading are:

1. To implement multimedia graphics.
2. To develop animations.
3. To develop video games etc.
4. To develop web and application servers

Whether it is process based or Thread based the main objective of multitasking is to improve performance of the system by reducing response time.

Thread life cycle:

A thread can be in one of the five states.

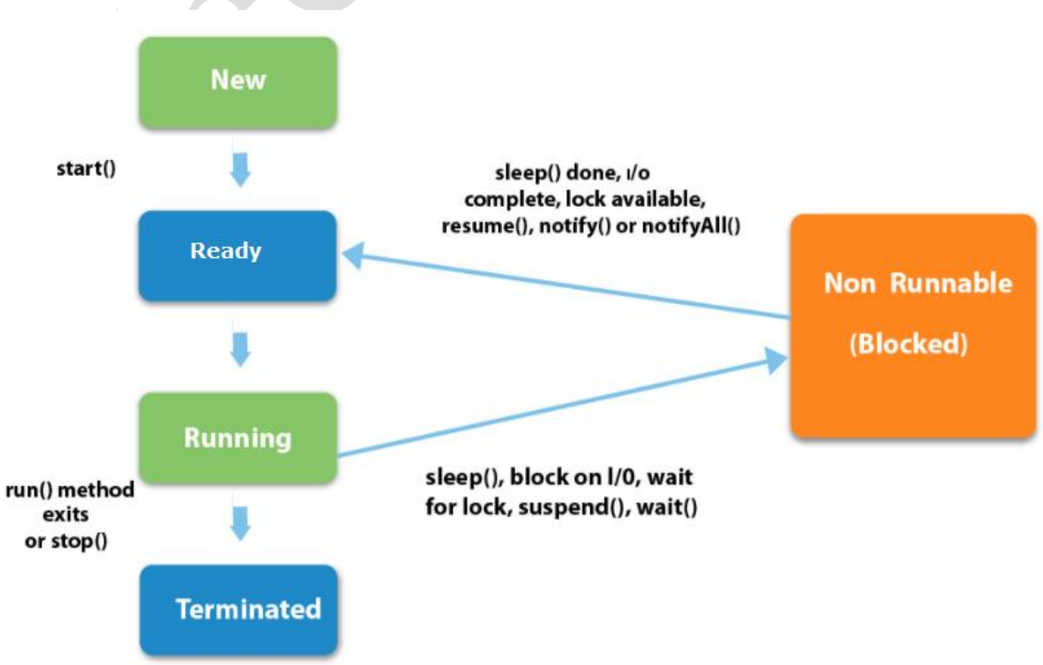
1) New: The thread is in new state if you create an instance of Thread class but before the invocation of start() method.

2) Ready: The thread is in ready state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

3) Running: The thread is in running state if the thread scheduler has selected it.

4) Non-Runnable (Blocked): This is the state when the thread is still alive, but is currently not eligible to run.

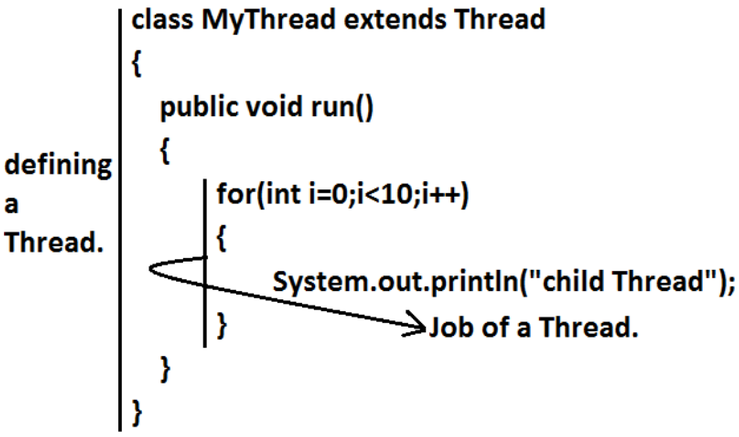
5) Terminated: A thread is in terminated or dead state when its run() method exits.



Methods to implement threads:

1. **Runnable interface**: This interface contains only one method called run(). We have to implement our own class by using Runnable interface.
2. **Thread class** in **java.lang** package: This Thread class is already implemented Runnable interface, so that we have to write our own class and make that class as child class to the Thread class and override run method.It contains some set of life cycle methods

Defining a Thread by extending "Thread class":




```
class ThreadDemo
{
        public static void main(String[] args)
        {
                MyThread t=new MyThread();    //Instantiation of a Thread
                t.start();                    //starting of a Thread
                for(int i=0;i<5;i++)
                {
                        System.out.println("main thread");
                }
        }
}
```

Thread Scheduler:

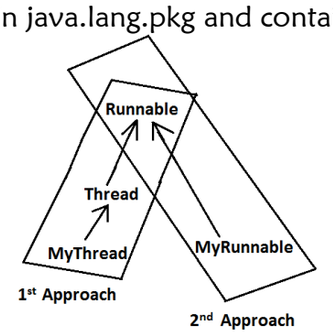
If multiple Threads are waiting to execute then which Thread will execute 1st is decided by "Thread Scheduler" which is part of JVM. Which algorithm or behaviour followed by Thread Scheduler we can't expect exactly it is the JVM vendor dependent hence in multithreading examples we can't expect exact execution order and exact output.

The following are various possible outputs for the above program.

p1	p2	p3
main thread	main thread	main thread
main thread	main thread	main thread
main thread	main thread	main thread
main thread	main thread	main thread
main thread	main thread	main thread
child thread	child thread	child thread
child thread	child thread	child thread
child thread	child thread	child thread
child thread	child thread	child thread
child thread	child thread	child thread
child thread	child thread	child thread
child thread	child thread	child thread
child thread	child thread	child thread
child thread	child thread	child thread
child thread	child thread	child thread

Defining a Thread by implementing Runnable interface:

We can define a Thread even by implementing Runnable interface also. Runnable interface present in java.lang.pkg and contains only one method run().



Example:

```
class MyRunnable implements Runnable
{
    public void run()
    {
        for(int i=0;i<10;i++)
        {
            System.out.println("child Thread");
        }
    }
}
```

defining a Thread

job of a Thread

```
class ThreadDemo
{
    public static void main(String[] args)
    {
        MyRunnable r=new MyRunnable();
        Thread t=new Thread(r);//here r is a Target Runnable
        t.start();
        for(int i=0;i<10;i++)
        {
            System.out.println("main thread");
        }
    }
}
```

Output:

```
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
child Thread
child Thread
child Thread
child Thread
child Thread
child Thread
child Thread
child Thread
child Thread
child Thread
```

We can't expect exact output but there are several possible outputs.

Best approach to define a Thread:

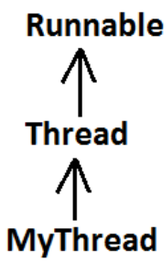
- Among the 2 ways of defining a Thread, implements Runnable approach is always recommended.

- In the 1st approach our class should always extends Thread class there is no chance of extending any other class hence we are missing the benefits of inheritance.
- But in the 2nd approach while implementing Runnable interface we can extend some other class also.

Hence implements Runnable mechanism is recommended to define a Thread.

Thread class constructors:

1. Thread t=new Thread();
2. Thread t=new Thread(Runnable r);
3. Thread t=new Thread(String name);
4. Thread t=new Thread(Runnable r,String name);
5. Thread t=new Thread(ThreadGroup g,String name);
6. Thread t=new Thread(ThreadGroup g,Runnable r);
7. Thread t=new Thread(ThreadGroup g,Runnable r,String name);
8. Thread t=new Thread(ThreadGroup g,Runnable r,String name,long stackSize);



Getting and setting name of a Thread:

- Every Thread in java has some name it may be provided explicitly by the programmer or automatically generated by JVM.
- Thread class defines the following methods to get and set name of a Thread.

Methods:

1. public final String getName()
2. public final void setName(String name)

Example:

```
class MyThread extends Thread
{

}
class ThreadDemo
{
        public static void main(String[] args)
        {
                System.out.println(Thread.currentThread().getName());        //main
                MyThread t=new MyThread();
                System.out.println(t.getName());                             //Thread-0
                Thread.currentThread().setName("Aditya Thread");
                System.out.println(Thread.currentThread().getName());        //Aditya Thread
        }
}
```

Note: We can get current executing Thread object reference by using Thread.currentThread() method.

Thread Priorities:

Every Thread in java has some priority it may be default priority generated by JVM (or) explicitly provided by the programmer.

The valid range of Thread priorities is 1 to 10[but not 0 to 10] where 1 is the least priority and 10 is highest priority.

Thread class defines the following constants to represent some standard priorities.

1. Thread. MIN_PRIORITY----------1
2. Thread. MAX_PRIORITY----------10
3. Thread. NORM_PRIORITY--------5

Thread scheduler uses these priorities while allocating CPU.

The Thread which is having highest priority will get chance for 1st execution.

If two Threads having the same priority then we can't expect exact execution order it depends on Thread scheduler whose behavior is vendor dependent.

We can get and set the priority of a Thread by using the following methods.

1. public final int getPriority()
2. public final void setPriority(int newPriority); //the allowed values are 1 to 10

The allowed values are 1 to 10 otherwise we will get runtime exception saying "IllegalArgumentException".

Default priority:

The default priority only for the main Thread is 5. But for all the remaining Threads the default priority will be inheriting from parent to child. That is whatever the priority parent has by default the same priority will be for the child also.

Example 1:

```
class MyThread extends Thread
{

}
class ThreadPriorityDemo
{
	public static void main(String[] args)
	{
		System.out.println(Thread.currentThread().getPriority());		//5
		Thread.currentThread().setPriority(9);
		MyThread t=new MyThread();
		System.out.println(t.getPriority());				//9
	}
}
```

Example 2:

```
class MyThread extends Thread
{
	public void run()
	{
		for(int i=0;i<10;i++)
		{
			System.out.println("child thread");
		}
	}
}
class ThreadPriorityDemo
{
	public static void main(String[] args)
	{
		MyThread t=new MyThread();
		//t.setPriority(10); //----> 1
		t.start();
		for(int i=0;i<10;i++)
		{
			System.out.println("main thread");
		}
	}
}
```

If we are commenting line 1 then both main and child Threads will have the same priority and hence we can't expect exact execution order.

If we are not commenting line 1 then child Thread has the priority 10 and main Thread has the priority 5 hence child Thread will get chance for execution and after completing child Thread main Thread will get the chance in this the output is:

Output:

```
child thread
child thread
child thread
```

```
child thread
child thread
child thread
child thread
child thread
child thread
child thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
```

Some operating systems(like windowsXP) may not provide proper support for Thread priorities. We have to install separate bats provided by vendor to provide support for priorities.

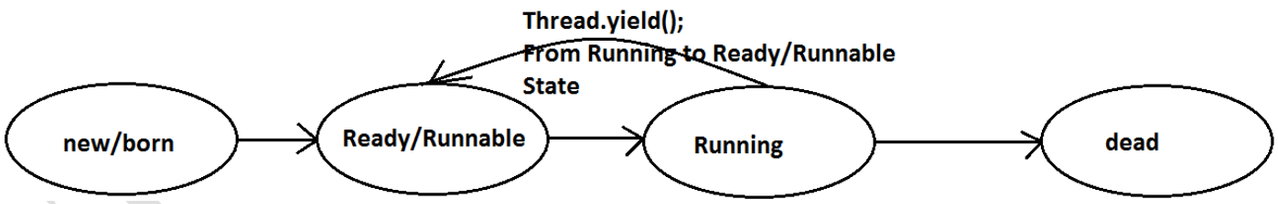
The Methods to Prevent a Thread from Execution:

We can prevent(stop) a Thread execution by using the following methods.

1. yield();
2. join();
3. sleep();

yield():

1. yield() method causes "to pause current executing Thread for giving the chance of remaining waiting
   Threads of same priority".
2. If all waiting Threads have the low priority or if there is no waiting Threads then the same Thread will be
   continued its execution.
3. If several waiting Threads with same priority available then we can't expect exact which Thread will get
   chance for execution.
4. The Thread which is yielded when it get chance once again for execution is depends on mercy of the
   Thread scheduler.
5. public static native void yield();



Example:

```
class MyThread extends Thread
{
        public void run()
        {
                for(int i=0;i<5;i++)
                {
                        Thread.yield();
                        System.out.println("child thread");
                }
        }
}
class ThreadYieldDemo
{
        public static void main(String[] args)
        {
```

```
                MyThread t=new MyThread();
                t.start();
                for(int i=0;i<5;i++)
                {
                        System.out.println("main thread");
                }
        }
}
```

Output:

```
main thread
main thread
main thread
main thread
main thread
child thread
child thread
child thread
child thread
child thread
```

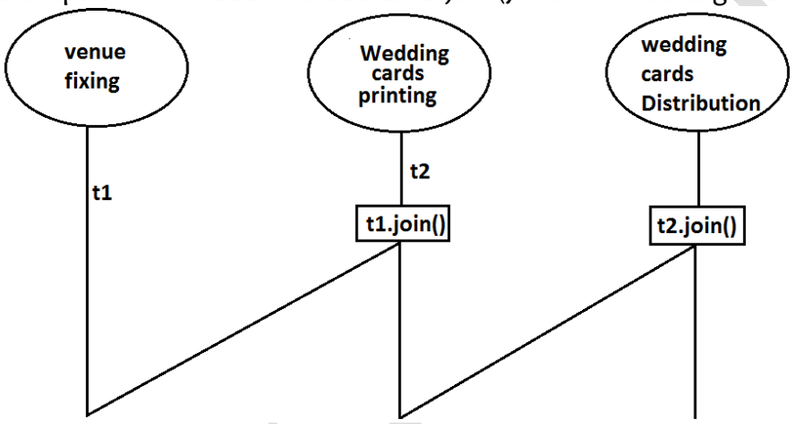
In the above program child Thread always calling yield() method and hence main Thread will get the chance more number of times for execution. Hence the chance of completing the main Thread first is high.

Note: Some operating systems may not provide proper support for yield() method.

join():

If a Thread wants to wait until completing some other Thread then we should go for join() method.

Example: If a Thread t1 executes t2.join() then t1 should go for waiting state until completing t2.



1. public final void join()throws InterruptedException
2. public final void join(long ms) throws InterruptedException
3. public final void join(long ms,int ns) throws InterruptedException

Every join() method throws InterruptedException, which is checked exception hence compulsory we should handle either by try catch or by throws keyword. Otherwise we will get compiletime error.

Example:

```
class MyThread extends Thread
{
        public void run()
        {
                for(int i=0;i<5;i++)
                {
                        System.out.println("Sita Thread");
                        try
                        {
                                Thread.sleep(2000);
                        }
                        catch (InterruptedException e){}
                }
        }
}
```

```
class ThreadJoinDemo
{
	public static void main(String[] args)throws InterruptedException
	{
		MyThread t=new MyThread();
		t.start();
		//t.join(); //--->1
		for(int i=0;i<5;i++)
		{
			System.out.println("Rama Thread");
		}
	}
}
```

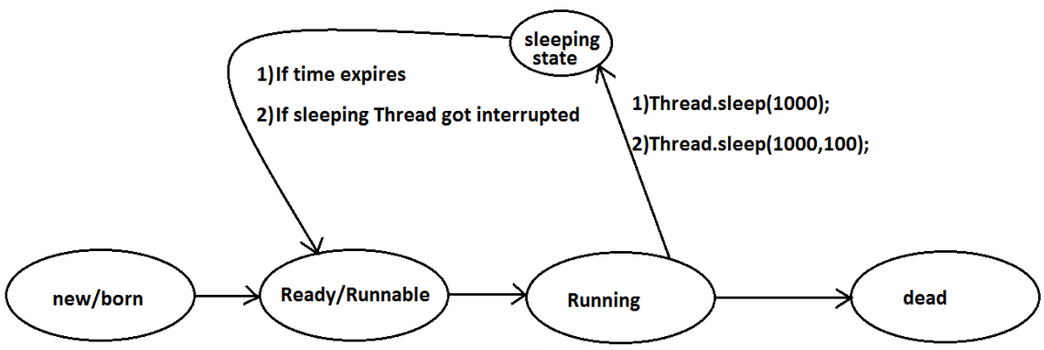
If we are commenting line 1 then both Threads will be executed simultaneously and we can't expect exact execution order.

If we are not commenting line 1 then main Thread will wait until completing child Thread in this the output is sita Thread 5 times followed by Rama Thread 5 times.

sleep() method:

If a Thread don't want to perform any operation for a particular amount of time then we should go for sleep() method.

1. public static native void sleep(long ms) throws InterruptedException
2. public static void sleep(long ms,int ns)throws InterruptedException



Example:

```
class ThreadJoinDemo
{
	public static void main(String[] args)throws InterruptedException
	{
		System.out.println("M");
		Thread.sleep(3000);
		System.out.println("E");
		Thread.sleep(3000);
		System.out.println("G");
		Thread.sleep(3000);
		System.out.println("A");
	}
}
```

Output:

```
M
E
G
A
```

Examples:

```
class Test implements Runnable
{
	long stime;
	Thread t;
	int i=1;
	public Test()
	{
		t = new thread (this, "Default");
		System.out.println("Name:"+ t.getName());
```

```
                stime=1000;
                t.start ();
        }
        public Test(String tname)
        {
                t=new Thread(this, tname);
                stime=500;
                System.out.println(t);
                t.start ();
        }
        public test (String tname, long time)
        {
                stime=time
                t = new Thread(this,name);
                System.out.println(t);
                t.start ();
        }

        public void run ()
        {
                try
                {
                        while (true)
                        {
                        System.out.println("Message From:"+ t.getName()+":"+i);
                        i++;
                        Thread.sleep(stime);
                        }
                }

                catch (Exception e)
                {
                        System.out.println(e);
                }
        }

        public static void  main (String args[])
        {
                Test t1 = new Test();
                Test t2 = new Test("One");
                Test t3 = new Test ("Two",250);
        }
}
```

Thread Synchronization:

In the multithreading environment, there is a chance to access a common area by more than one thread at a time. The threads, which are trying to access this common area, will be treated as cooperative threads. And the common area is called **critical section**. The threads are can share nothing will be treated as independent threads. More than one thread must not be enter into critical section at a time, it leads to deadlock [or] loss of data. So to overcome this problem we have to make the critical section in such a way so that only one thread is able to enter into critical section at a time. The first thread is trying to access critical section it must lock that section, then only it continue with its work. If any other thread is passing request at the same time, it has to wait, until resource will release by the first thread. This is called **Synchronization**.

1. Synchronized is the keyword applicable for methods and blocks but not for classes and variables.
2. If a method or block declared as the synchronized then at a time only one Thread is allow to execute that method or block on the given object.
3. The main advantage of synchronized keyword is we can resolve data inconsistency problems. But the main disadvantage of synchronized keyword is it increases waiting time of the Thread and effects performance of the system.

4. Hence if there is no specific requirement then never recommended to use synchronized keyword.
5. Internally synchronization concept is implemented by using lock concept. Every object in java has a unique lock. Whenever we are using synchronized keyword then only lock concept will come into the picture.
6. If a Thread wants to execute any synchronized method on the given object 1st it has to get the lock of that object. Once a Thread got the lock of that object then it's allow to execute any synchronized method on that object. If the synchronized method execution completes then automatically Thread releases lock.
7. While a Thread executing any synchronized method the remaining Threads are not allowed execute any synchronized methods on that object simultaneously. But remaining Threads are allowed to execute any non-synchronized method simultaneously. [lock concept is implemented based on object but not based on method].

Example:

```
class Display
{
	public synchronized void wish(String name)
	{
		for(int i=0;i<5;i++)
		{
			System.out.print("good morning:");
			try
			{
				Thread.sleep(1000);
			}
			catch (InterruptedException e)
			{}
			System.out.println(name);
		}
	}
}
class MyThread extends Thread
{
	Display d;
	String name;
	MyThread(Display d,String name)
	{
		this.d=d;
		this.name=name;
	}
	public void run()
	{
		d.wish(name);
	}
}
class SynchronizedDemo
{
	public static void main(String[] args)
	{
		Display d1=new Display();
		MyThread t1=new MyThread(d1,"dhoni");
		MyThread t2=new MyThread(d1,"yuvaraj");
		t1.start();
		t2.start();
	}
}
```

If we declare wish()method as synchronized then the Threads will be executed one by one that is until completing the 1st Thread the 2nd Thread will wait in this case we will get output as:

Output:
good morning:dhoni
good morning:dhoni
good morning:dhoni
good morning:dhoni
good morning:dhoni
good morning:yuvaraj
good morning:yuvaraj
good morning:yuvaraj
good morning:yuvaraj
good morning:yuvaraj

If we are not declaring wish() method as synchronized then both Threads will be executed simultaneously and we will get irregular output.

Output:
good morning:good morning:yuvaraj
good morning:dhoni
good morning:yuvaraj
good morning:dhoni
good morning:yuvaraj
good morning:dhoni
good morning:yuvaraj
good morning:dhoni
good morning:yuvaraj
dhoni

Inter Thread communication (wait(),notify(), notifyAll()):

- Two Threads can communicate with each other by using wait(), notify() and notifyAll() methods.
- The Thread which is required updation it has to call wait() method on the required object then immediately the Thread will entered into waiting state. The Thread which is performing updation of object, it is responsible to give notification by calling notify() method. After getting notification the waiting Thread will get those updations.

1. **wait ()**: it is used to specify the methods commonly be waiting to access a resource.
2. **notify()**: to give the notification to a particular thread, which is waiting to access that resource.
3. **notifyAll()**: this is used to give notification to all those threads, which are waiting to access that resource.

All the above methods must be written in synchronized block.
The above concept will be treated as producer and consumer problem, and the operations involved in that problem will be as follows.

Producer consumer problem:

Producer(producer Thread) will produce the items to the queue and consumer(consumer thread) will consume the items from the queue. If the queue is empty then consumer has to call wait() method on the queue object then it will entered into waiting state. After producing the items producer Thread call notify() method on the queue to give notification so that consumer Thread will get that notification and consume
items.

Source Code:

```
import java.io.*;
//critical section: buffer object is passed into both producer and consumer threads
class Buffer
{
        String data;
        boolean avail=false;
        public synchronized void put(String data)//used by producer
        {
                while (avail==true)
                {
                        try
```

```
                {
                        wait ();
                }
                catch(Exception e)
                {System.out.println(e);}
        }
        this.data=data;
        System.out.println("Produced:"+data);
        avail = true;
        notify();
    }
    public synchronized String get()  // used by consumer
    {
        while(avail==false)
        {
            try
            {
                wait (); // it throws InterruptedException
            }
            catch (exception e)
            {System.out.println(e);}
        }
        avail = false;
        notify ();
        return data;
    }
}

class Producer extends Thread
{
    String data;
    BufferedReader b = new BufferedReader(new InputStreamReader(System.in));
    Buffer buf;
    public Producer(Buffer buf)
    {
        super("Producer");
        this.buf=buf;
        start();
    }
    public void run ()
    {
        try
        {
            while (true)
            {
                System.out.println("Enter data");
                data=b.readLine ();
                buf.put(data);
                Thread.sleep(500); //InterruptedException
            }
        }catch (Exception e)
        {System.out.println(e);}
    }
}

class Consumer extends Thread
{
    Buffer buf;
    public Consumer(buffer buf)
    {
        super ("Consumer");
        this.buf=buf;
```

```
                start ();
        }
        public void run ()
        {
                try
                {
                        while (true)
                        {
                                System.out.println("Consumed:"+ buf.get());
                                Thread.sleep (500);
                        }
                }catch(Exception e)
                {System.out.println(e);}
        }
}
class Product
{
        public static void main (String args [])
        {
                Buffer buf = new Buffer ();
                Producer p = new Producer (buf);
                Consumer c = new Consumer (buf);
        }
}
```

Daemon Threads:

The Threads which are executing in the background are called daemon Threads. The main objective of daemon Threads is to provide support for non-daemon Threads like main Thread.

Example:

Garbage collector

Whenever the program runs with low memory the JVM will execute Garbage Collector to provide free memory. So that the main Thread can continue it's execution.

We can check whether the Thread is daemon or not by using isDaemon() method of Thread class.

```
public final boolean isDaemon();
```

We can change daemon nature of a Thread by using setDaemon () method.

```
public final void setDaemon(boolean b);
```

But we can change daemon nature before starting Thread only. That is after starting the Thread if we are trying to change the daemon nature we will get R.E saying *IllegalThreadStateException*.

Default Nature: Main Thread is always non daemon and we can't change its daemon nature because it's already started at the beginning only. Main Thread is always non daemon and for the remaining Threads daemon nature will be inheriting from parent to child that is if the parent is daemon child is also daemon and if the parent is non daemon then child is also non daemon.

Whenever the last non daemon Thread terminates automatically all daemon Threads will be terminated.

Example:

```
class MyThread extends Thread
{
        public void run()
        {
                for(int i=0;i<10;i++)
                {
                        System.out.println("lazy thread");
                        try
                        {
                                Thread.sleep(2000);
                        }
                        catch (InterruptedException e)
                        {}
                }
        }
```

```
}
class DaemonThreadDemo
{
	public static void main(String[] args)
	{
		MyThread t=new MyThread();
		t.setDaemon(true); //-->1
		t.start();
		System.out.println("end of main Thread");
	}
}
```

Output:

End of main Thread

If we comment line 1 then both main & child Threads are non-Daemon, and hence both threads will be executed untill there completion.

If we are not comment line 1 then main thread is non-Daemon and child thread is Daemon. Hence when ever main Thread terminates automatically child thread will be terminated.

ThreadGroup:

Based on functionality we can group threads as a single unit which is nothing but ThreadGroup. ThreadGroup provides a convenient way to perform common operations for all threads belongs to a particular group.

We can create a ThreadGroup by using the following constructors

```
ThreadGroup g=new ThreadGroup(String gName);
```

We can attach a Thread to the ThreadGroup by using the following constructor of Thread class

```
Thread t=new Thread(ThreadGroup g, String name);

ThreadGroup g=new ThreadGroup("Printing Threads");
MyThread t1=new MyThread(g,"Header Printing");
MyThread t2=new MyThread(g,"Footer Printing");
MyThread t3=new MyThread(g,"Body Printing");
----------
g.stop();
```

Class Runtime

```
class RunTimeDemo
{
	public static void main(String args[])
	{
	Runtime r=Runtime.getRuntime();
	System.out.println(r.freeMemory ());
	System.out.println(r.totalMemory ());
	r.gc (); // (or) system.gc(); Grabage Collection
	r.exec("notepad");//Executes the specified string command in a separate process.
	r.exec("mspaint")
	r.exec("calc");
	}
}
```

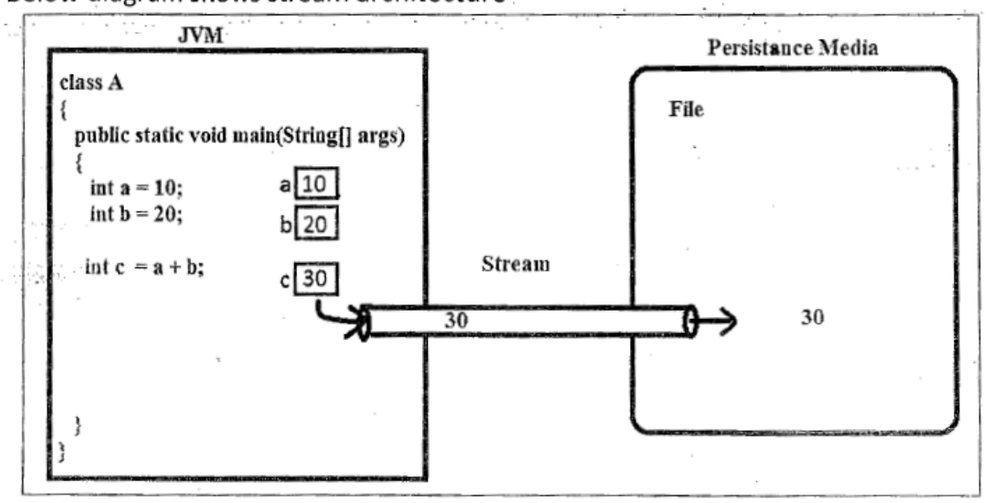
IO Streams or File Handling

Java.io package contains classes to perform input and output operations. By using java.io package we are performing file handling java.

Stream:

Stream is a logical connection between java program and a file. Stream can be defined as "it is continuous flow of data between java program and persistence media.

Below diagram shows stream architecture



Types of Streams:

Generally streams are divided into two types based on data flow direction.

InputStream: the stream that allows data to come into the java application from the persistent media is called InputStream.

OutputStream: The stream that allows data to send out from the java application to be stored into the persistent media is called OutputStream.

I/O Streams:

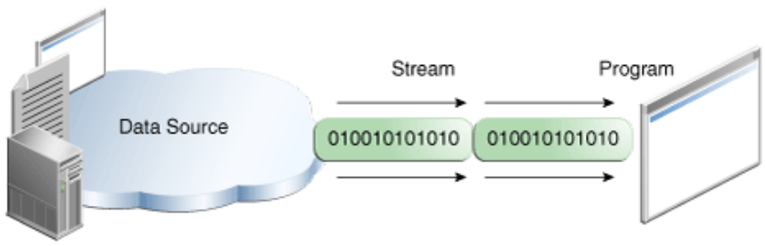
An I/O stream represents input source or an output destination. Streams represent many kind of source and destination like disk files & devices & memory arrays.

Streams support different kind of data

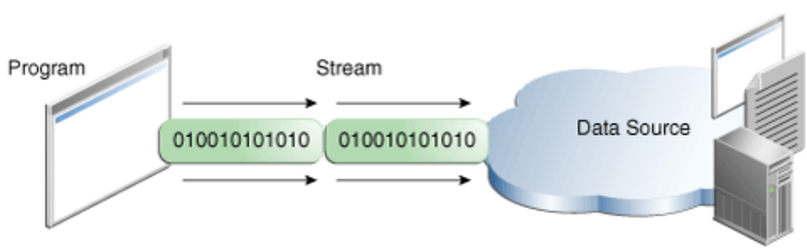
- Simple bytes.
- Primitive data types
- Localized characters
- Objects …..etc

Stream is a communication channel between source and destination & a stream is a sequence of data.

Input stream:- Program uses Input stream to read the data from a source one item at a time.



Output stream:- Program uses output stream to write the data to a destination one item at a time.



In java we are allowed to send data through streams only either in the format of bytes or characters.

Byte Streams:

The streams which read and write data in the format of bytes called Binary Streams.

Character Streams:

The streams which read and write data in the format of character is called character streams.

File class constructors:

1. File f=new File(String name);
   Creates a java File object that represents name of the file or directory in current working directory.
2. File f=new File(String subdirname,String name);
   Creates a File object that represents name of the file or directory present in specified sub directory.
3. File f=new File(File subdir,String name);

Programs:

Example1:

```
import java.io.*;
class FileDemo
{
	public static void main(String[] args)throws IOException
	{
		File f=new File("cricket.txt");
		System.out.println(f.exists());			//false
		f.createNewFile();
		System.out.println(f.exists());			//true
	}
}
```

Example2:

A java File object can represent a directory also.

```
import java.io.*;
class FileDemo
{
	public static void main(String[] args)throws IOException
	{
		File f=new File("cricket123");
		System.out.println(f.exists());//false
		f.mkdir();
		System.out.println(f.exists());//true
	}
}
```

Example3: Write code to create a file named with demo.txt in current working directory.

```
import java.io.*;
class FileDemo
{
	public static void main(String[] args)throws IOException
	{
		File f=new File("demo.txt");
		f.createNewFile();
	}
}
```

Example4: Write code to create a directory named with SaiCharan123 in current working directory and create a file named with abc.txt in that directory.

```
import java.io.*;
class FileDemo
{
	public static void main(String[] args)throws IOException
	{
		File f1=new File("SaiCharan123");
		f1.mkdir();
		File f2=new File("SaiCharan123","abc.txt");
		f2.createNewFile();
	}
}
```

Example5: Write code to create a file named with demo.txt present in c:\saicharan folder.

```
import java.io.*;
class FileDemo
{
	public static void main(String[] args)throws IOException
	{
		File f=new File("c:\\saiCharan","demo.txt");
		f.createNewFile();
	}
}
```

Example6: Write a program to display the names of all files and directories present in c:\\charan_classes.

```
import java.io.*;
class FileDemo
{
	public static void main(String[] args)throws IOException
	{
		 int count=0;
		File f=new File("c:\\charan_classes");
		String[] s=f.list();
		for(String s1:s) {
			count++;
			System.out.println(s1);
			}
		System.out.println("total number : "+count);
	}
}
```

Example7: Write a program to display only file names

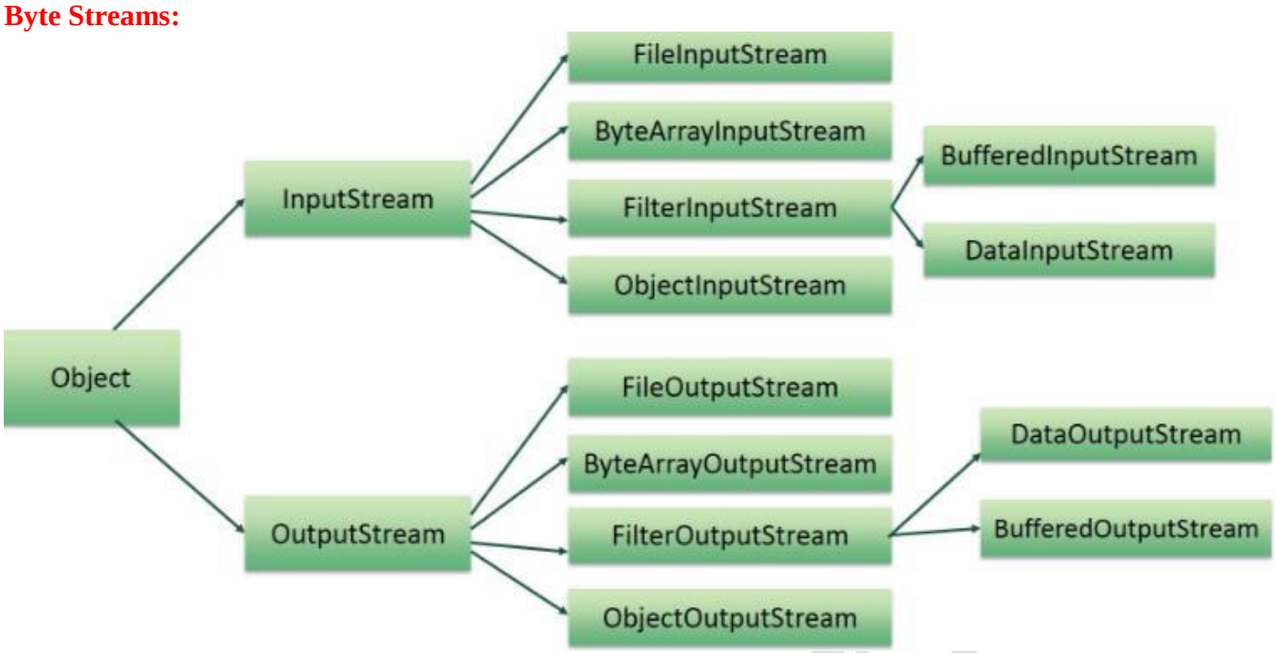
```
import java.io.*;
class FileDemo
{
	public static void main(String[] args)throws IOException
	{
		 int count=0;
		File f=new File("c:\\charan_classes");
		String[] s=f.list();
		for(String s1:s) {
			File f1=new file(f,s1);
			if(f1.isFile()) {
				count++;
				System.out.println(s1);
				}
			}
		System.out.println("total number : "+count);
	}
}
```

Example8: Write a program to display only directory names

```
import java.io.*;
class FileDemo
{
	public static void main(String[] args)throws IOException
	{
		int count=0;
		File f=new File("c:\\charan_classes");
		String[] s=f.list();
		for(String s1:s)
		{
			File f1=new file(f,s1);
			if(f1.isDirectory())
			{
				count++;
				System.out.println(s1);
```

```
                }
            }
            System.out.println("total number : "+count);
        }
}
```

Byte Streams:



Program uses byte stream to perform input & output of byte data. All byte stream classes developed based on InputStream & OutputStream. Program uses byte stream to perform input and output of 8-bit bytes.

To demonstrate how the byte stream works file I/O provided two main classes

FileInputStream

It is used to read the data from source one item at a time.

To read the data from source use read() method of FileInputStream class.

public int read() throws java.io.IOException;

read() method returns first character Unicode value in the form of integer value.

FileOutputStream

It is used to write the data to destination one item at a time.

To write the data to destination use write() method of FileOutputStream class.

public void write(int unicode) throws java.io.IOException;

write() method is taking Unicode value of the character as a parameter.

Example8:

```
import java.io.*;
class Test
{
    public static void main(String[] args)throws FileNotFoundException,IOException
    {
        //Byte oriented channel creation
        FileInputStream fis = new FileInputStream("abc.txt");
        FileOutputStream fos = new FileOutputStream("xyz.txt");
        int c;
        while((c=fis.read())!=-1)
        {
            System.out.print((char)c);
            fos.write(c);
        }
        System.out.println("read() & write operatoins are completed");

        //stream closing operations
        fis.close();
        fos.close();
    }
}
```

Limitation of byte streams:

It reads the data only one byte at a time hence it takes more time to copy.
Must close the streams always.
These streams always hitting hard disk to retrieve the data it reduces the performance.

Character streams:

Program uses character stream to perform input & output of character data. All character stream classes developed based on Reader & Writer classes.

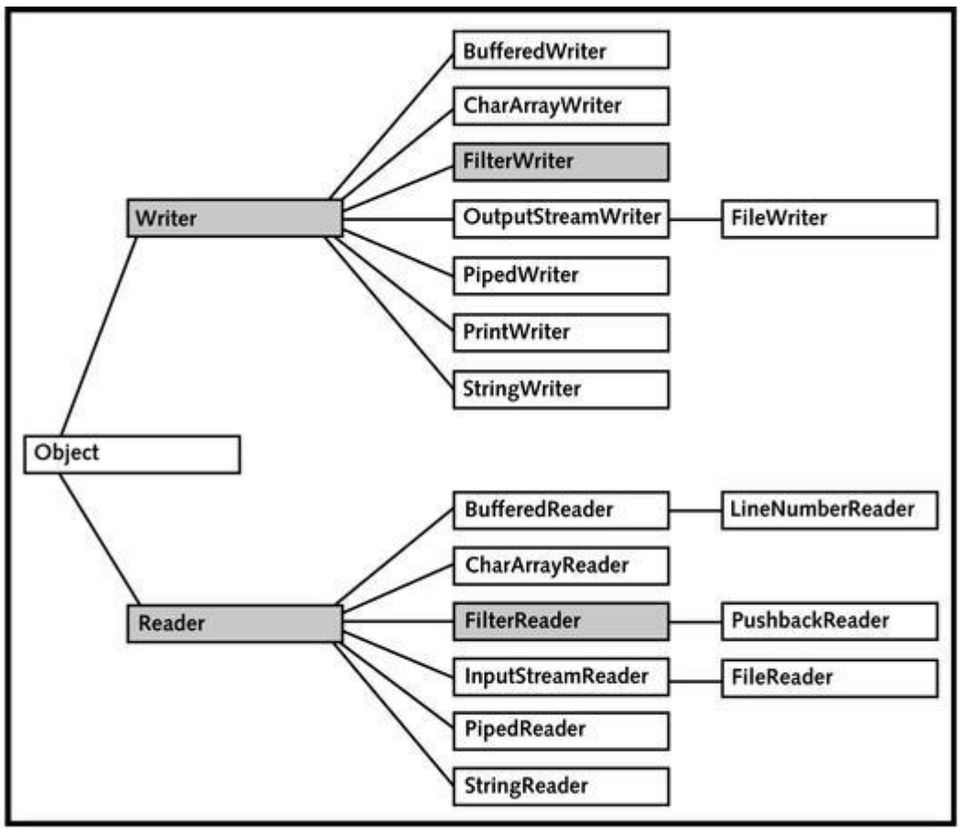


Figure 8-2 Character stream classes

To demonstrate how the character stream works file I/O provided two main classes

FileReader

It is used to read the data from source one item at a time.
To read the data from source use read() method of FileInputStream class.
public int read() throws java.io.IOException;
read() method returns first character Unicode value in the form of integer value.

FileWriter

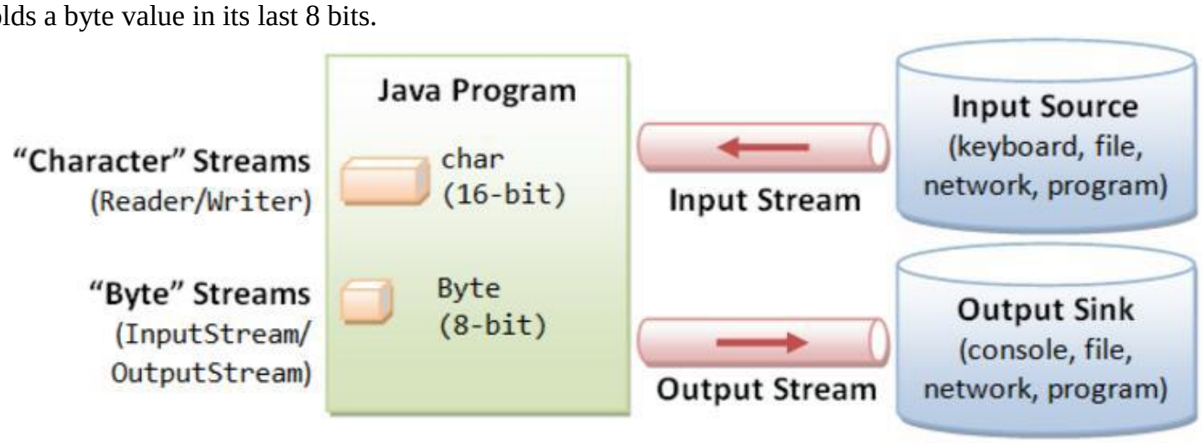
It is used to write the data to destination one item at a time.
To write the data to destination use write() method of FileOutputStream class.
public void write(int unicode) throws java.io.IOException;
write() method is taking Unicode value of the character as a parameter.

Example9:

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
public class Test
{
	public static void main(String[] args) throws IOException
	{
		FileReader inputStream = null;
		FileWriter outputStream = null;
		try {
			inputStream = new FileReader("abc.txt");
			outputStream = new FileWriter("characteroutput.txt");
			int c;
			while ((c = inputStream.read()) != -1)
			{
				outputStream.write(c);
			}
		}
```

```
                finally
                {
                        if (inputStream != null)
                        {
                                inputStream.close();
                        }
                        if (outputStream != null)
                        { outputStream.close();
                        }
                }
        }
}
```

Note:
In CopyCharacters, the int variable holds a character value in its last 16 bits; in CopyBytes, the int variable holds a byte value in its last 8 bits.



Buffered Streams:
In previous examples we are using un-buffered I/O .This means each read and write request is handled directly by the underlying OS. In normal streams each request directly triggers disk access it is relatively expensive & performance is degraded.

To overcome above limitations use buffered streams.

- Bufferd input stream read the data from buffered memory and it interacting with hard disk only when buffered memory is empty.
- Buffered output stream write the data to buffer memory.

There are four buffered stream classes.

Buffered byte streams,

1. BufferedInputStream
2. BufferedOutputStream

A program can convert an un buffered stream into buffered streams.

new BufferedInputStream(new FileInputStream("xanadu.txt"));
new BufferedOutputStream(new FileOutputSream("byteoutput.txt"));

Buffered Character streams,

3. BufferedReader
4. BufferedWriter

A program can convert an un buffered stream into buffered streams.

new BufferedReader(new FileReader("ratan.txt"));
new BufferedWriter(new FileWriter("characteroutput.txt"));

Example10:

```
import java.io.*;
class Test
{
        public static void main(String[] args)
        {
                BufferedReader br;
                BufferedWriter bw;
                try{
                        br=new BufferedReader(new FileReader("Test1.java"));
                        bw=new BufferedWriter(new FileWriter("States.java"));
                        String str;
                        while ((str=br.readLine())!=null)
```

```
                {
                        bw.write(str);
                }
                br.close();
                bw.close();
        }
        catch(Exception e)
        {
                System.out.println("getting Exception");
        }
    }
}
```

Example11:

```
import java.io.*;
class Test
{
    public static void main(String[] args)
    {
        BufferedInputStream bis;
        BufferedOutputStream bos;
        try{
            bis=new BufferedInputStream(new FileInputStream("abc.txt"));
            bos=new BufferedOutputStream(new FileOutputStream("xyz.txt"));
            int str;
            while ((str=bis.read())!=-1)
            {
                bos.write(str);
            }
            bis.close();
            bos.close();
        }
        catch(Exception e)
        {
            System.out.println(e);
            System.out.println("getting Exception");
        }
    }
}
```

Example12:Program to Write the Data into the File using FileWriter

```
import java.io.*
class FileWrite
{
    public static void main(String args [])
    {
        try
        {
        BufferedReader b=new BufferedReader(new InputStreamReader(System.in));
        File f=new File(args[0]);
        String data;
        BufferedWriter fout=new BufferedWriter(new FileWriter(f));
        while(true)
        {
            System.out.println("Enter some lines of text (end-stop):");
            data=b.readLine();
            if(data.equals("end"))
                break;
            fout.write(data,0,data.length());
        }
        fout.close ();
        }catch(Exception e)
        {
            System.out.println(e);
        }
```

```
        }
    }
```

Example13: Program to Write the Data into the File using DataOutputStream

```
import java.io.*;
class DataWrite
{
    public static void main(String args[])
    {
        try
        {
            int eno;
            String ename;
            double sal;
            BufferedReader b;
            b=new BufferedReader(new InputStreamReader(System.in));
            DataOutputStream fout;
            while (true)
            {
                fout=new DataOutputStream(new FileOutputStream(args[0],true));
                System.out.println("Enter No,Name & Sal");
                eno = Integer.parseInt(b.readLine());
                ename = b.readLine();
                sal = Double.parseDouble(b.readLine());
                fout.writeInt(eno);
                fout.writeBytes ("\n"+ename+"\n");
                fout.writeDouble (sal);
                fout.writeBytes("\n");
                System.out.println("Do you want to continue: \n");
                int ch=System.in.read();        //similar to getch ()
                if(ch=='y' || ch=='Y')
                    break;
            }
            fout.close ();
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

Example14: Program to Read the Data from the File using DataInputStream

```
import java.io. *;
class DataRead
{
public static void main(String args[])
{
    DataInputStream fin;
    try
    {
        int eno;
        String ename;
        double sal;
        File f = new File(args[0]);
        if(!f.exists())
        {
            System.out.println("File does not exist");
            System.exit(0);
        }
        fin=new DataInputStream(new FileInputStream(f));
        while (true)
        {
            eno= fin.readInt();
            ename= fin.readLine();
```

```
                    sal=fin.readDouble();
                    System.out.println("No:"+eno);
                    System.out.println("Name:"+ename);
                    System.out.println ("Sal:"+sal);
                }
            }catch(EOFException e)
            {       fin.close ();
            }
            catch(Exception e)
            {System.out.println(e);}
        }
    }
```

Example15: StreamTokenizers

```
    import java.io.*;
    class TokenDemo
    {
        public static void main(String args[])
        {
            try
            {
                File f=new File(args[0]);
                FileInputStream fin=new FileInputStream(f);
                StreamTokenizer st=new StreamTokenizer(fin);
                int token;
                while(true)
                {
                    token=st.nextToken();
                    if(token==StreamTokenizer.TT_WORD)
                    System.out.println(st.sval);
                    else if(token==StreamTokenizer.TT_NUMBER)
                    System.out.println(st.nval):
                    else if (token==StreamTokenizer.TT_EOF)
                    break:
                }
                fin.close();
            }catch(Exception e)
            {
                System.out.println (e);
            }
        }
    }
```

Application: compiler designing, spelling check etc

Example16: Program to Write the Data into the File using PrintStream

```
        import java.io. *;
        class PrintDemo
        {
            public static void main (String args[])
            {
                try
                {
                BufferedReader b;
                b=new BufferedReader(new InputStreamReader(System.in));
                FileOutputStream fout=new FileOutputStream(args[0],true);
                PrintStream ps=new PrintStream(fout);
                int eno;
                String ename;
                double sal;
                System.out.println("Enter No,Name & Sal");
                eno=Integer.parseInt(b.readLine());
                ename=b.readLine();
```

```
                    sal=Double.parseDouble(b.readLine ());
                    ps.println(eno);
                    ps.println(ename);
                    ps.println(sal);
                    ps.close();
                    }catch(Exception e)
                        {System.out.println(e);}
                }
}
```

Abstract Window ToolKit (AWT)

Concepts of AWT:

AWT stands for Abstract Window Toolkit.

It is API for develop GUI (or) window based Applications.

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 AWT contains large no. of classes & methods that allow you to create and manage windows.

The AWT classes are contained in the java.awt package.

Class	Description
AWTEvent	Encapsulate AWT events.
Button	Creates a push button.
Canvas	A blank free window
Checkbox	Creates a check box control.
Choice	Creates a pop-up list.
Color	Manages color in a portable, platform-independent fashion.
Cursor	Encapsulates a bit mapped cursor.
Dialog	Creates a dialog window.
Font	Encapsulates a type font
Frame	Creates a standard window that has a title bar, resize corners, menu bar.
Graphics	Encapsulates the graphics context.
Image	Encapsulate graphical images.
Label	Creates a label that displays a string.
List	Creates a list from which the user can choose.
Menu	Creates a pull down menu.
Scrollbar	Creates a scroll bar
TextArea	Creates a multiline edit control.
point	Encapsulates a coordinate pair, stored in x, y-axis.
Event	Encapsulate events.
Component	An abstract super class for various Awt components.
Container	A sub class of component that can hold other components.
Panel	The simplest sub class of container.
Window	Creates a window with no frame, no menu bar, and no title.
Toolkit	Abstract class implemented by the Awt.

AWT Component Hierarchy:

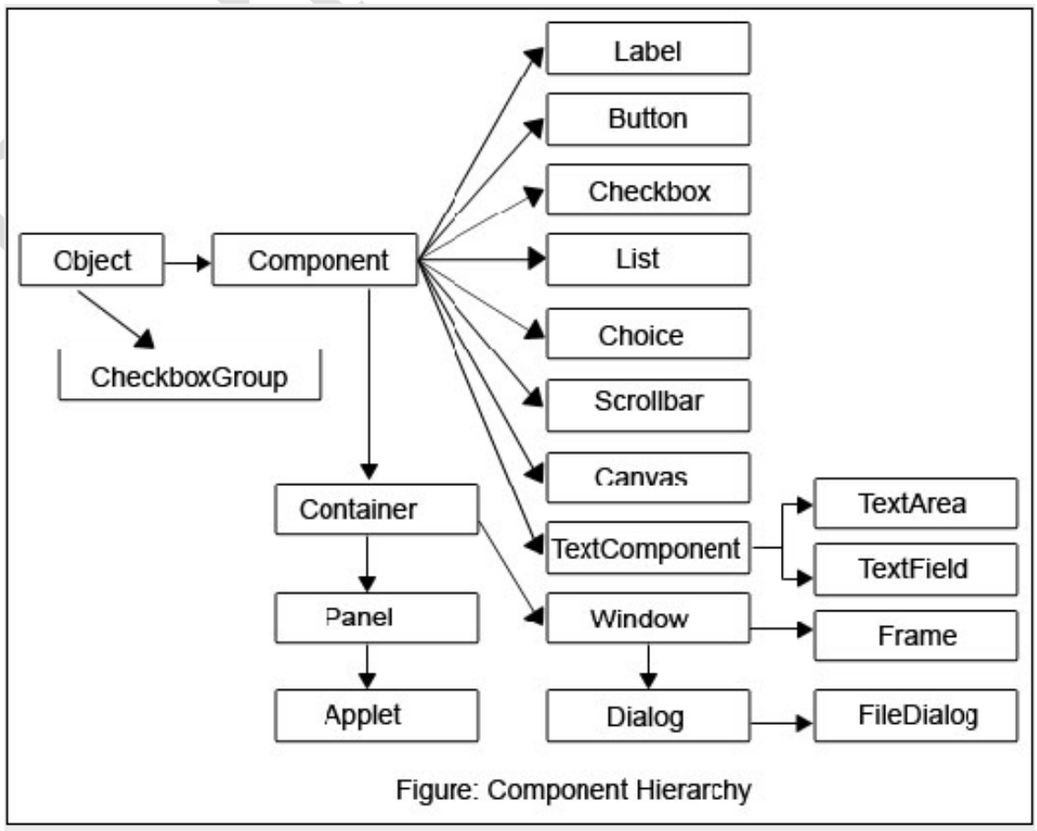


Figure: Component Hierarchy

Window fundamentals:

The AWT defines windows according to a class hierarchy that adds functionality and specificity with each level. The two most common windows are those derived from Panel, which is used by applets, and those derived from Frame, which creates a standard window. Much of the functionality of these windows is derived from their parent classes.

Component:

1. At the top of the AWT hierarchy is the Component class.
2. Component is an abstract class that encapsulates all of the attributes of a visual component.
3. All user interface elements that are displayed on the screen and that interact with the user are subclasses of Component.
4. Component is an object which is displayed pictorially on the screen.
   Ex:- Button,Label,TextField.....etc

Container:

1. The Container class is a subclass of Component.
2. It has additional methods that allow other Component objects to be nested within it.
3. Other Container objects can be stored inside of a Container (since they are themselves instances of Component). This makes for a multileveled containment system.
4. A container is responsible for lay out (that is, positioning) any components that it contains.
5. The classes that extends container classes those classes are containers such as **Frame, Dialog and Panel.**

Panel:

1. The Panel class is a subclass of Container. It doesn't add any new methods; it simply implements Container.
2. Panel is the super class for Applet. A Panel is a window that does not contain a title bar, menu bar, or border. When you run an applet using an applet viewer, the applet viewer provides the title and border.
3. Other components can be added to a Panel object by it's add ( ) method inherited from Container.
4. Once these components have been added, you can position and resize them manually using the setLocation( ), setSize( ), or setBounds( ) methods defined by Component.

Window:

Window class creates a top-level window. A top-level window is not contained within any other object; it sits directly on the desktop. Generally, we don't create Window objects directly. Instead, we use a subclass of Window called Frame.

Frame:

Frame encapsulates what is commonly thought of as a "window." It is a subclass of Window and has a title bar, menu bar, borders, and resizing corners.

Canvas:

It is not part of the hierarchy for applet or frame windows. Canvas encapsulates a blank window upon which you can draw.

Discuss working with Frame Windows:

Frame: A frame represents a window with some title & borders.

Creating a Frame:

There are 3 ways to create a frame window:

1. Create a frame class object:

   Frame f=new Frame (); - creates a standard window that doesn't contain a title.

2. Create a frame class object and pass its title also:

   Frame f= new Frame ("My Frame");

3. Create a subclass MyFrame to the Frame class & create an object to the subclass:

   Class MyFrame extends Frame
   MyFrame f=new MyFrame ();

Frame methods:

1.setSize() - this method is used to set the dimensions of the window.
Syntax: void setSize (int width, int height);

2.getSize() - this method is used to obtain the current size of the window.

3.setTitle(String title) - this method is used to change the title in a frame window.
Syntax: void setTitle (String title);

4.setVisible (boolean flag) – After a frame window has been created , it will not be visible until you call setVisible() is set true.

Closing a Frame:

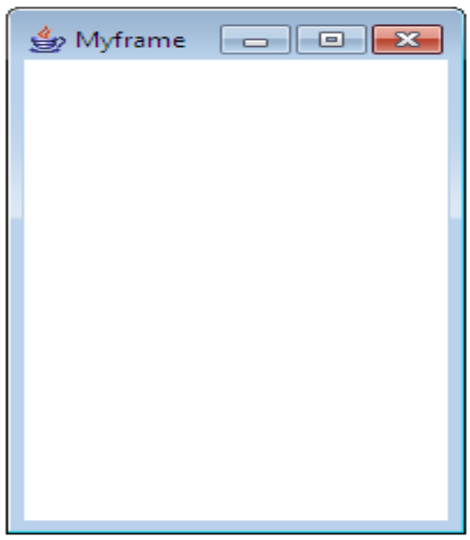
Remove the window frame from the screen when it is closed, by calling setVisible(false);

Close the window event; you must implement the windowClosing method of windowListener interface.

Example1:

```
import java.awt.*;
public class FrameDemo extends Frame
{
        public static void main(String[] poojitha)
        {
                FrameDemo framedemo=new FrameDemo();
                framedemo.setTitle("Myframe");
                framedemo.setSize(200,300);
                framedemo.setVisible(true);
        }
}
```

Output:



Use of a Frame:

Once frame is created,we can use it for any purposes.

1.To draw some graphical shapes like dots,lines,rectangles etc..

2.To display some text,pictures or images in the frame.

3.To display components like push buttons, radio buttons etc..

Graphics class:

The Graphics class defines a number of drawing function. Each shape can be draw edge-only or filled. Objects are drawn and filled in the currently selected graphics color, which is black by default. When graphics object is drawn that exceeds the dimensions of the window, output is automatically clipped.

Several drawing methods:

1. **Drawing Lines:** Lines are drawn by means of the drawLine() method.
   Syntax: void drawLine(int start X, int start Y,int end X, int end Y)
2. **Drawing Rectangles:** drawRect() & fillRect() methods display an outlined & filled rectangle, respectively.
   Syntax: void drawRect(int top,int left,int width,int heigh)
   void fillRect(int top,int left,int width,int height)
   drawRoundRect() or fillRoundRect() are used to draw a round Rectangle
   Syntax: void drawRoundRect(int top, int left,int width,int height,int x Diam, int y Diam)
   void fillRoundRect(int top, int left, int width,int height,int xDiam int ydiam)
3. **Drawing Ellipse & Circles:** 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)
4. **Drawing Arcs:** Arcs can be draw with drawArc() & fillArc().
   Syntax: void drawArc(int top,int left, int height, int width, int startAngle,int sweepAngle)
   void fillArc(int top, int left,int height,int width,int startAngle,int sweepAngle)
5. **Drawing polygons:** drawpolygon() & fillpolygon(), are used to draw the polygons.
   Syntax: void drawpolygon(int x[],int y[],int num points)
   void fillpolygon(int x[],int y[],int num points)

There are two approaches to create Frame in java

1. By creating object of Frame class.
2. By extending the Frame class.

Approach 1:- Creation of Frame by creating Object of Frame class.

```
import java.awt.*;
class Demo
{
	public static void main(String[] args)
	{
		Frame f=new Frame();
		f.setVisible(true);
		f.setSize(400,400);
		f.setBackground(Color.red);
		f.setTitle("myframe");
	}
}
```

Approach 2:- Taking user defined class by extending Frame class.

```
import java.awt.*;
class MyFrame extends Frame
{
	MyFrame()
	{
		setVisible(true);
		setSize(500,500);
		setTitle("myframe");
		setBackground(Color.green);
	}
	public static void main(String[] args)
	{
		MyFrame f=new MyFrame();
	}
}
```

Working with Color:

Java support color in a portable, device-independent fashion. The AWT color system allows you to specify any color you want. Color is encapsulated by the Color class. Color defines several constants eg: Color.black, Color.blue, Color.cyan etc..

There are two ways to set a color in AWT

1.The first way is by directly mentioning the needed color name from color class as Color.red, Color.pink etc..

2. The second way to mention any color is by combining the three primary colors: red, green & blue while creating color class object

```
Color c=new Color(r,g,b);
```

Here r,g,b values can change from 0 to 255

0 - Represents no color. 10 - Represents low intensity

200 - Represents high intensity of color

```
Color c=new Color(255,0,0);          //red color
Color c=new Color(255,255,255);      //white color
Color c=new Color(0,0,0);            //black color
```

Creating colors:

You can also create your own colors, uisng one of the color constructors. There are 3 ways mostly used.

Syntax:

1. Color(int red, int green, int blue,)
2. Color(int rgb value)
3. Color(float red, float green, float blue)

Example: new colr(255,100,100); //light red.

1. The first constructor takes three intergers that specify the color as mix of red, green & blue. These values must between 0 to 255.

2. The second color constructor takes a single integer that contains the mix of red, green,blue packed in to integer.The integer is organized with red bits-16 to 23, green 8 to 15 blue in bits 0 to 7.

Example: int newRed=(0xff000000/(0x(0<<16)/(0x00<<8)/(0x00);

```
Color darkRed=new Color(newRed)
```

3. The final constructor color(float, float, float)

Take 3 float values (between 0.0 & 1.0) that specify the relative mix of red, green & blue.

Methods:

Once you have created a color, you can use it to set the foreground & background color by using the setForeground() & setBackground() methods.

setForeground(): To set the foreground color of an window

setBackground(): To set the background color of an window

Working with Fonts:

The AWT supports multiple type fonts. Fonts have a familyname, a logical fontname, and a facename. The familyname is the general name of the font, the logicalname specifies a category of font, The facename specifies a specific font. Fonts are encapsulated by the font class.

Font methods:

String getFamily() - Returns the name of the font family to which the invoking to not belongs .

String getFontName() - Returns the font of the invoking font.

Static Font getFont(String property) - Returns the font associated with the system property specified by property. null is returned if property does not exist.

Determining the Available Fonts:

When working with fonts, offen you need to know which fonts are available on your machine.To be obtain this information,you can use the getAvailable FontFamilyNames()methods defined by the Graphics Environment class.

Syntax:

String [] getAvailableFontFamilyNames()

This method returns an array of strings that contains the name of the available font families.

The getAllFonts() method is defined by the Graphics Environment class.

Font[] getAllFonts()

Creating and selecting a Font:

To select a new font, you must first construct a Font object that describes that Font.

Syntax:

Font(String FontName, int FontStyle, int pointSize)

To use a font that you have created, you must select it using setFont() Which is defined by component.

Syntax:

void setFont(Font Fontobj) Here,fontobj is the object that contains the desired font.

Program:

```
import java.awt.*;
import java.awt.BorderLayout;
class Frame3 extends Frame
{
	public static void main(String[] args)
	{
		Frame3 t = new Frame3();
		t.setVisible(true);
		t.setSize(500,500);
		t.setTitle("My Frame");
		t.setBackground(Color.pink);
	}
	public void paint(Graphics g)
	{
		Font f=new Font("arial",Font.ITALIC,25);
		g.setFont(f);
		g.drawString("hi Ramesh how r u",100,100);
	}
}
```

Output:

Introduction of Event Handling:

- ✓ Object resides in a particular state until it is made transit to the other state. This transition occurs due to an "Event". E.g. pressing key on the key board
- ✓ The object which generates the event, i.e., the source of the event is known as the "Event Generator". E.g. button.
- ✓ The object that is responsible for performing the task when the event occurs is called the "Event Handler".
- ✓ There may be more than one event handler for one single event generated; each event handler responsible for doing a unique activity on account of that particular event.
- ✓ Now the question is, how do these handlers know that a particular event has occurred so that they can perform their activity? For this, there is a registration process, under taken by the event object, for each of its event handlers.
- ✓ This registration involves the event handler simply asking the event generator to inform about the occurrence of an event. By registering, the event generator is able to keep track of all the registered event handlers.
- ✓ Without Event handling, it is impossible to enter into the word of GUI programming.

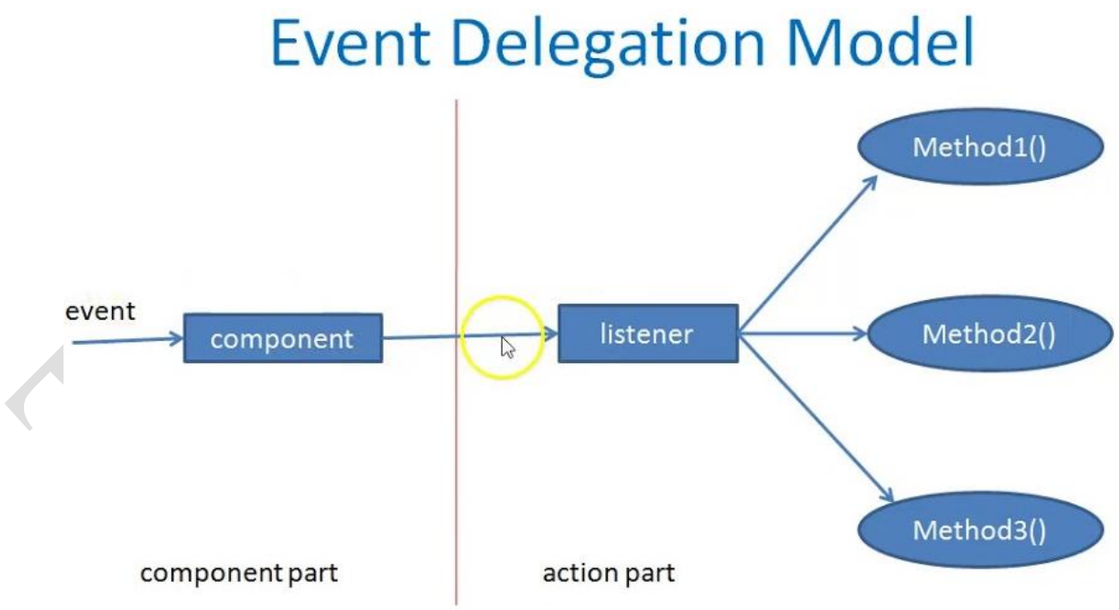
Delegation Event Model:

- ✓ Event delegation model is an approach that has been followed since java 1.0
- ✓ In the event delegation model, a source generates events which are sent to one or more listeners. The listeners are responsible for receiving the event.
- ✓ The event- generating component can be designed differently that the event- processing component.
- ✓ Actually the event generating component delegates the responsibility of performing an event based action to a separate event- performing component.
- ✓ The model has three dimensions, namely events, event sources, and event listeners

Event: an event is an object that describes a state change in the source. It may be generated directly because of interaction with the components of GUI environment or any other reason such as expiry of timer, completion of an operation etc.

Event source: is an object which generates the event. Generation of event must cause a change in the state of the source. A source can generate more than one event.

Event listeners: are the objects that get notified when an evet occurs on an event source.



Event Classes:

Java.awt.event package: The java.awt.event package provides interfaces & class for dealing with different types of events fired by AWT components. The java.awt.AWT Eevent class is the root for all AWT events. This package includes the definition of events classes, listeners' interfaces, and adapter's classes, which form the basics of event handling.

Event Class	Description
ActionEvent	Generated when a button is pressed, a list item is double-clicked, or a menu item is selected.
AdjustmentEvent	Generated when a scroll bar is manipulated.
ComponentEvent	Generated when a component is hidden, moved, resized, or becomes visible.
ContainerEvent	Generated when a component is added to or removed from a container.
FocusEvent	Generated when a component gains or loses keyboard focus.
InputEvent	Abstract superclass for all component input event classes.
ItemEvent	Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected.
KeyEvent	Generated when input is received from the keyboard.
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.
MouseWheelEvent	Generated when the mouse wheel is moved.
TextEvent	Generated when the value of a text area or text field is changed.
WindowEvent	Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

SOURCES OF EVENTS: Sources of events can be either components of GUI or any other class derived from a component (such as an applet), which can generate event like events from keyboard and mouse. Some of the components of GUI are:

Buttons, Menu item, Check box, List, Window, Scroll bar, Text components.

Event Listener Interfaces:

- ✓ Event listeners are created by implementing one or more interfaces defined by the java.awt. Event package.
- ✓ Whenever a source generates an event, it basically invokes the appropriate method defined in the listener interface.
- ✓ The method has an event object passed as an argument to it.

List of Event Listeners and its methods:

Listener (interface)	Action (methods)	Event (classes)
1. ActionListener	actionPerformed ()	ActionEvent
2. TextListener	textValueChanged ()	TextEvent
3. AdjustmentListener	adjustmentValueChanged ()	AdjustmentEvent
4. ItemListener	itemStateChanged ()	ItemEvent
5. FocusListener	focusGained () focusLost ()	FocusEvent
6. KeyListener	keyPressed () keyReleased () keyTyped ()	KeyEvent
7. MouseMotionListener	mouseMoved () mouseDragged ()	
8. MouseListener	mouseEntered () mouseExited () mousePressed () mouseReleased () mouseClicked ()	MouseEvent
9. WindowListener	windowOpened () windowClosing () windowClosed () windowIconified () windowDeiconified () windowActivated () windowDeactivated ()	WindowEvent

Each of these listener interfaces have certain number of methods, which are given in brief below.
KeyListenerInterface: This interface has three methods defined within it:
-void keypressed(keyEvent e):- is invoked when a key is pressed.
-void keyReleased(KeyEvent e):- is invoked when a key is released
-void keyTyped(KeyEvent e):- is invoked when a character is typed
MouseListenerInterface: This interface has five methods, having the signatures as follows:
-void mouseClicked(MouseEvent e):- is invoked when a mouse is clicked.
-void mouseEntered(MouseEvent e):- is invoked when mouse enters a component.
-void mousePressed(MouseEvent e):- is invoked when mouse is pressed but not released.
-void mouseReleased(MouseEvent e):- is invoked when pressed mouse is released.
-void mouseExited(MouseEvent e):- is invoked when the mouse leaves component.
MouseMotionListenerInterface: This interface has two methods having the signatures
-void MouseMoved(MouseEvent e):- is invoked when the mouse is moved from one place to another place.
-void MouseDragged(MouseEvent e):- is used when the mouse is dragged.
MouseWheelListenerInterface: This interface has only one method, having the signature,
-void MouseWheelMoved(MouseEvent e):- is invoked when the mouse wheel is moved.
ItemListenerInterface: This interface has one method having the signatures.
-void itemStateChanged(ItemEvent e):- is invoked when the state of item changes.
ActionListenerInterface: This interface has one method having the signatures.
-void actionPerformed(ActionEvent e):- is invoked when any action event is performed.
TextListenerInterface: This interface has one method having the signatures.
-void textChanges(TextEvent e):- is invoked when there is a change in text field or text area.
WindowListener interface: This interface has five methods having the signatures
-void windowActivated (WindowEvent e):- is invoked when a window is activated.
-void windowClosed (WindowEvent e):- is invoked when a window is closed.
-void windowClosing(WindowEvent e):- is invoked when a window is being closed.
-void windowDeactivated(WindowEvent e):- is invoked when a window is deactivated.
-void windowOpened(WindowEvent e):- is invoked when a window is opened.

ComponentListenerInterface: This interface has four methods having the signatures
-void component Resized (Component event e):- is invoked when a size of the component is altered.
-void componentMoved(Component event e):- is invoked when the component is moved
-void component Shown (Component event e):- is invoked when the component is shown.
-void component Hidden (Component event e):- is invoked when the component is hidden.
ContainerListenerInterface: This interface has two methods having the signatures
-void component Added(Container Event e):- is invoked when the component is added to the container.
-void component Removed(Container Event e):- is invoked when the component is removed from container.
AdjustmentListenerInterface: This interface has two methods having the signatures
-void adjustmentValueChanged(AdjustmentEvent e):- is invoked when an adjustment event class.

Working with Components:

The AWT supports the following types of controls:

* Labels	* Scrollbars
* push button	* Text Fileds
* check boxes	* Text Area
* choice lists	* menus
* Lists	* dialog boxes

These controls are subclasses of component.

Label: A label is an object of type Label, and it contains a string, which it displays.

Constructors:

Label (): creates a blank label.

Label (String str): creates a label that contains the string specified by str. string left-justified.

Label(String str , int how): creates label that contains the string specified by str an set the alignment specified by how.

The value of how must be one of these three constants:

Label . LEFT , Label .RIGHT, Label. CENTER.

Methods of the label:

setText(): you can set or change the text in a label by using the setText()method.

Syntax :

void setText(String str) – str specifies the new label.

getText(): you can obtain the current label by calling getText()

Syntax:

String getText () -- current label is returned.

setAlignment(): you can set the alignment of the String within the label by calling setAlignment (), To obtain the current alignment, call getAlignment().

Syntax: void setAlignment(int how)
int getAlignment()

Button: A push button is a component that contains a label and that generates an event when it is pressed.

Constructors:

Button ()-creates an empty button.

Button (String str) –creates a button that contains str as label

Methods: After a button has been created,

setLabel() - you can set its label by calling setLabel ()

Syntax: void setLabel(String str)

getLabel() - you can retrieve its label by calling getLabel-String getLabel()

Handling Buttons:

When, there are several butons, naturally the programmer should know which button is clicked by the user. For this purpose, getActioncommand() method of ActionEvent class is useful.

String s=ae.getActioncommand();

Represent the label of the buton clicked by the user.

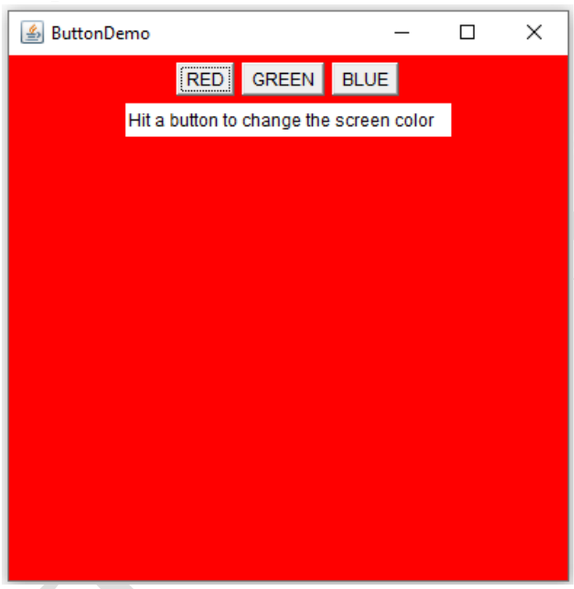
Each time a button is pressed, an Action event is generated.This is sent to any Listeners that previously Registered an interest in Receiving action event notifications from that component. Each listener implements the Action Listener interface, that interface defines the actionPerformed() method,which is called when an event occurs. An Action Event object is supplied as argument to this method.

ButtonDemo.java

```
import java.awt.*;
import java.awt.event.*;
public class ButtonDemo extends Frame implements ActionListener
{
        Button red,green,blue;
        Label hit;
        public ButtonDemo()
        {
        setLayout(new FlowLayout());
        red=new Button("RED");
        green=new Button("GREEN");
        blue=new Button("BLUE");
        hit=new Label("Hit a button to change the screen color");
        add(red);
        add(green);
        add(blue);
        add(hit);
        red.addActionListener(this);
```

```
green.addActionListener(this);
blue.addActionListener(this);
}
public void actionPerformed(ActionEvent ae)
{
    String str=ae.getActionCommand();
    if(str.equals("RED"))
    {
            setBackground(Color.red);
    }
    else if(str.equals("GREEN"))
    {
            setBackground(Color.green);
    }
    else if(str.equals("BLUE"))
    {
            setBackground(Color.blue);
    }

}
public static void main(String args[])
{
    ButtonDemo b=new ButtonDemo();
    b.setSize(400,400);
    b.setTitle("ButtonDemo");
    b.setVisible(true);
}
}
```

Output:



Checkboxes: A checkbox is a square shaped box which displays an option to the user. The user can select one or more options from a group of checkbox. To create a checkbox, we can create an object to checkbox class.

Syntax:

```
Checkbox cb =new Checkbox();
Checkbox cb =new Checkbox("label");
Checkbox cb= new checkbox("label",state);
```

Methods:

Syntax:

```
boolean  getState()
void   setState(boolean on)
String  getLabel()
void  setLabel(String str)
```

Radio Button:

AWT does not provide any predefined support to create Radio Buttons directly.
It is possible to create RadioButton by using two classes.

- o CheckboxGroup
- o Checkbox

step 1:- Create Checkbox group object. CheckboxGroup cg=new CheckboxGroup();
step 2:- pass Checkboxgroup object to the Checkbox class argument.

Checkbox cb1=new Checkbox("male",cg,true);
Checkbox cb2=new Checkbox("female",cg,false);

- To get the status of the RadioButton : String str=cb.getState();
- To get Label of the RadioButton : String str=getLabel().

CheckBoxDemo.java

```
import java.awt.*;
class CheckBoxDemo extends Frame
{
        Checkbox cb1,cb2,cb3,cb4,r1,r2;
        CheckboxGroup cg;
        Label l1,l2;
        public CheckBoxDemo()
        {
            setLayout(new FlowLayout());
            l1 = new Label("Qualifications:");
            l2 = new Label("Gender:");
            cb1=new Checkbox("SSC",true);
            cb2=new Checkbox("DEGREE");
            cb3=new Checkbox("MCA");
            cb4=new Checkbox("BTECH");
            add(l1);
            add(cb1);
            add(cb2);
            add(cb3);
            add(cb4);
            System.out.println(cb1.getLabel());
            System.out.println(cb2.getState());

            CheckboxGroup cg=new CheckboxGroup();
            Checkbox r1=new Checkbox("Male",cg,true);
            Checkbox r2=new Checkbox("Female",cg,false);
            add(l2);
            add(r1);
            add(r2);
            System.out.println(cb1.getLabel());
            System.out.println(cb1.getState());
        }
        public static void main(String[] args)
        {
            CheckBoxDemo f=new CheckBoxDemo();
            f.setVisible(true); f.setTitle("ratan");
            f.setBackground(Color.red);
            f.setSize(400,500);
            f.setLayout(new FlowLayout());

        }
}
```

Output:

```
D:\Java_Programs\AWT>java CheckBoxDemo
SSC
false
SSC
true
```

Choice:

List is allows to select multiple items but choice is allow to select single Item.

Choice ch=new Choice();

- To add items to the choice : add()
- To remove item from the choice based on String : choice.remove("HYD");
- To remove the item based on the index position : choice.remove(2);
- To remove the all elements : ch.removeAll();
- To inset the data into the choice based on the position : choice.insert(2,"CSE");
- To get selected item from the choice : String s=ch.getSelectedItem();
- To get the selected item index number : int a=ch.getSelectedIndex();

List:

List is providing list of options to select.

Constructor:

List l=new List(); It will creates the list by default size is four elements.

List l=new List(3); It will display the three items size and it is allow selecting the only one.

List l=new List(5,true); It will display five items and it is allow selecting the multiple items.

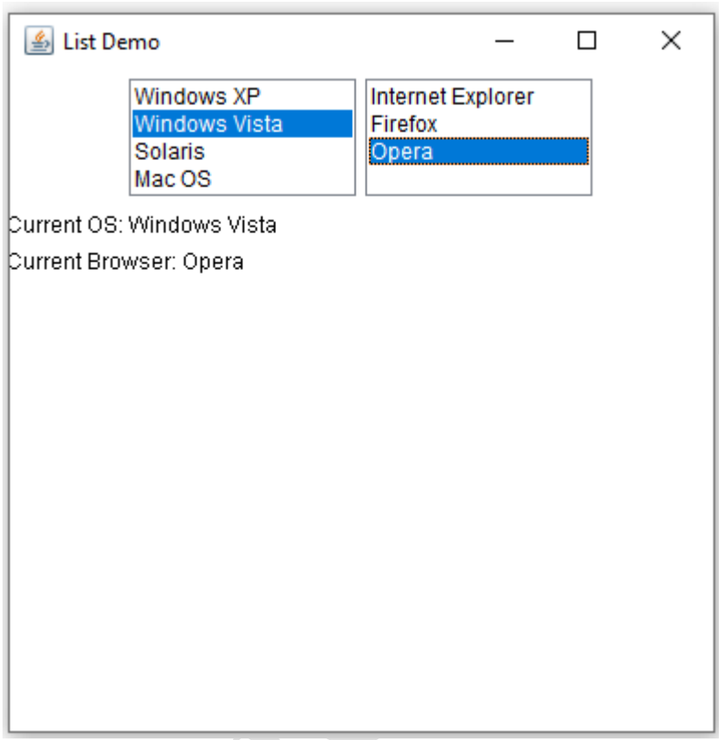
- To add the elements to the List : list.add("c");
- To add the elements to the List at specified index : list.add("CSE",0);
- To remove element from the List : list.remove("c");
- To get selected item from the List : String x=l.get SelectedItem();
- To get selected items from the List : String[] x=s.getSelectedItems()

ListDemo.java

```
import java.awt.*;
import java.awt.event.*;
public class ListDemo extends Frame implements ActionListener
{
        List os, browser;
        String msg = "";
        public ListDemo()
        {
                setLayout(new FlowLayout());
                os = new List(4,true);
                browser = new List(4,false);
                // add items to os list
                os.add("Windows XP");
                os.add("Windows Vista");
                os.add("Solaris");
                os.add("Mac OS");
                // add items to browser list
                browser.add("Internet Explorer");
                browser.add("Firefox");
                browser.add("Opera");
                browser.select(1);
                // add lists to window
                add(os);
                add(browser);
                // register to receive action events
                os.addActionListener(this);
                browser.addActionListener(this);
        }
        public void actionPerformed(ActionEvent ae)
        {
                repaint();
        }
        // Display current selections.
        public void paint(Graphics g)
        {
                int idx[];
                msg = "Current OS: ";
                idx = os.getSelectedIndexes();
                for(int i=0; i<idx.length; i++)
                msg += os.getItem(idx[i]) + " ";
                g.drawString(msg, 6, 120);
```

```
                msg = "Current Browser: ";
                msg += browser.getSelectedItem();
                g.drawString(msg, 6, 140);
        }
        public static void main(String args[])
        {
                ListDemo l=new ListDemo();
                l.setTitle("List Demo");
                l.setSize(400,400);
                l.setVisible(true);
        }
}
```

Output:



Scroll Bars: Scroll bars are used to select continuous values between a specified minimum and maximum. Scroll bars may be oriented horizontally or vertically. Scroll bars are encapsulated by the Scrollbar class.

Constructors:

Scrollbar( )
Scrollbar(int style)
Scrollbar(int style, int initialValue, int thumbSize, int min, int max)

The first form creates a vertical scroll bar. The second and third forms allow you to specify the orientation of the scroll bar.

If style is Scrollbar.VERTICAL, a vertical scroll bar is created.

If style is Scrollbar.HORIZONTAL, the scroll bar is horizontal.

Example:

```
import java.awt.*;
import java.awt.event.*;
class SBDemo extends Frame implements AdjustmentListener
{
        String msg = "";
        Scrollbar vertSB, horzSB;
        public SBDemo()
        {
                setLayout(new FlowLayout());
                int width = 250;
                int height = 250;
                vertSB = new Scrollbar(Scrollbar.VERTICAL,0, 1, 0, height);
                horzSB = new Scrollbar(Scrollbar.HORIZONTAL,0, 1, 0, width);
                add(vertSB);
                add(horzSB);
                // register to receive adjustment events
```

```
                vertSB.addAdjustmentListener(this);
                horzSB.addAdjustmentListener(this);

        }
        public void adjustmentValueChanged(AdjustmentEvent ae)
        {
                repaint();
        }
        public void paint(Graphics g)
        {
                msg = "Vertical: " + vertSB.getValue();
                msg += ", Horizontal: " + horzSB.getValue();
                g.drawString(msg, 6, 160);
                // show current mouse drag position
                g.drawString("*", horzSB.getValue(),vertSB.getValue());
        }
        public static void main(String args[])
        {
                SBDemo sb=new SBDemo();
                sb.setTitle("Scroll Bar Demo");
                sb.setSize(500,500);
                sb.setVisible(true);
        }
}
```

Output:

Text Field:

The TextField class implements a single-line text-entry area, usually called an edit control. TextField is a subclass of Text Component.

Constructors:

TextField( )
TextField(int numChars)
TextField(String str)
TextField(String str, int numChars)

Methods: Text Field provides several methods that allow you to utilize a text field.

getText( )-To obtain the string currently contained in the text field, call getText( )

void String getText( )

setText( )-To set the text, call setText( ).

void setText(String str)

Here, str is the new string.

select( )-The user can select a portion of the text in a text field. Also, you can select a portion of text under program control by using select( ).

void select(int startIndex, int endIndex)

The select ( ) method selects the characters beginning at startIndex and ending at endIndex–1.

getSelectedText( )-obtain the currently selected text by calling getSelectedText( ).

String getSelectedText( )

getSelectedText( ) returns the selected text.

setEditable( )-whether the contents of a text field may be modified by the user by calling setEditable( ).

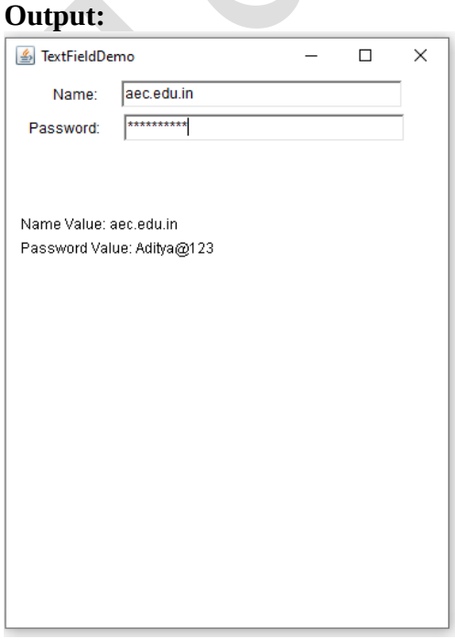
boolean isEditable( )

void setEditable(boolean canEdit).

Example:

```
import java.awt.*;
import java.awt.event.*;
class TextFieldDemo extends Frame implements TextListener
{
	TextField name, pass;
	Label l1,l2,l3;
	public TextFieldDemo()
	{
		setLayout(new FlowLayout());
		l1 = new Label("Name: ");
		l2 = new Label("Password: ");
		l3=new Label("");
		name = new TextField(30);
		pass = new TextField(30);
		pass.setEchoChar('*');
		add(l1);
		add(name);
		add(l2);
		add(pass);
		add(l3);
		// register to receive action events
		name.addTextListener(this);
		pass.addTextListener(this);
	}
	public void textValueChanged(TextEvent te)
	{
		repaint();
	}
	public void paint(Graphics g)
	{
		g.drawString("Name Value: " + name.getText(), 20, 160);
		g.drawString("Password Value: " + pass.getText(), 20, 180);
	}
	public static void main(String args[])
	{
		TextFieldDemo tf=new TextFieldDemo();
		tf.setSize(500,500);
		tf.setTitle("TextFieldDemo");
		tf.setVisible(true);
	}
}
```

Output:



Text Area: TextArea is a subclass of TextComponent .the AWT includes a simple multiline editor called TextArea.

Constructors:

TextArea( )
TextArea(int numLines, int numChars)
TextArea(String str)
TextArea(String str, int numLines, int numChars)
TextArea(String str, int numLines, int numChars, int sBars)

Methods:

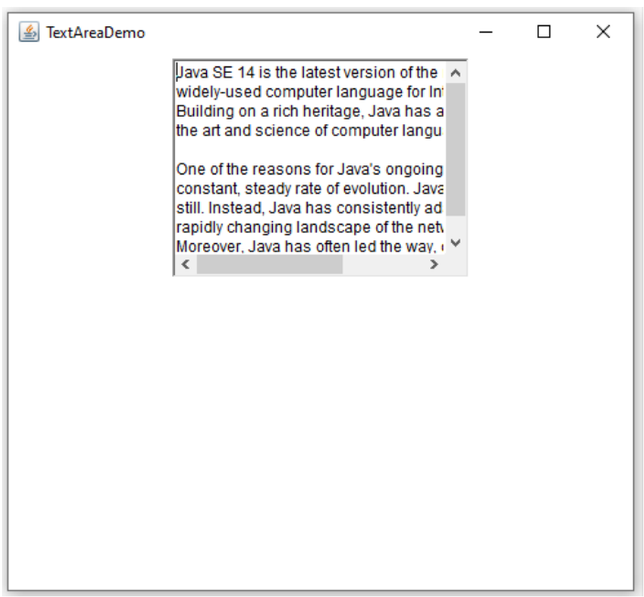
Therefore, it supports the getText( ), setText( ), getSelectedText( ), select( ), isEditable( ), and setEditable( ) methods.

void append(String str)
void insert(String str, int index)
void replaceRange(String str, int startIndex, int endIndex)

Example:

```
import java.awt.*;
class TextAreaDemo extends Frame
{
	TextArea text;
	public TextAreaDemo()
	{
		setLayout(new FlowLayout());
		String val =
		"Java SE 14 is the latest version of the most\n" +
		"widely-used computer language for Internet programming.\n" +
		"Building on a rich heritage, Java has advanced both\n" +
		"the art and science of computer language design.\n\n" +
		"One of the reasons for Java's ongoing success is its\n" +
		"constant, steady rate of evolution. Java has never stood\n" +
		"still. Instead, Java has consistently adapted to the\n" +
		"rapidly changing landscape of the networked world.\n" +
		"Moreover, Java has often led the way, charting the\n" +
		"course for others to follow.";

		text = new TextArea(val, 10, 30);
		add(text);
	}
	public static void main(String args[])
	{
		TextAreaDemo ta=new TextAreaDemo();
		ta.setSize(500,500);
		ta.setTitle("TextAreaDemo");
		ta.setVisible(true);
	}
}
```

Output:



Menus:

A top-level window can have a menu bar associated with it. A menu bar displays a list of top-level menu choices. Java uses 3 classes for creating menus.

1.MenuBar
2.Menu
3.MenuItem

A menu bar contains one or more Menu objects. Each Menu object contains a list of MenuItem objects. Each MenuItem object represents something that can be selected by the user. Since Menu is a subclass of MenuItem. create instances of Menu that will define the selections displayed on the bar.

Constructors:

Menu( )
Menu(String optionName)
Menu(String optionName, boolean removable)

Here, optionName specifies the name of the menu selection. If removable is true, the menu can be removed and allowed to float free. Individual menu items are of type MenuItem.

Constructors:

MenuItem( )
MenuItem(String itemName)
MenuItem(String itemName, MenuShortcut keyAccel)

Here, itemName is the name shown in the menu, and keyAccel is the menu shortcut for this item. You can disable or enable a menu item by using the setEnabled( ) method.

Methods:

void setEnabled(boolean enabledFlag) - the If argument enabledFlag is true, the menu item is enabled. If false, the menu item is disabled.
boolean isEnabled( )
setLabel( ) - change the name of a menu item by calling setLabel( ). void setLabel(String newName)
getLabel( ) -You can retrieve the current name by using getLabel( ).
add( ) - created a menu item, you must add the item to a Menu object by using add( )
void add(MenuItem item)

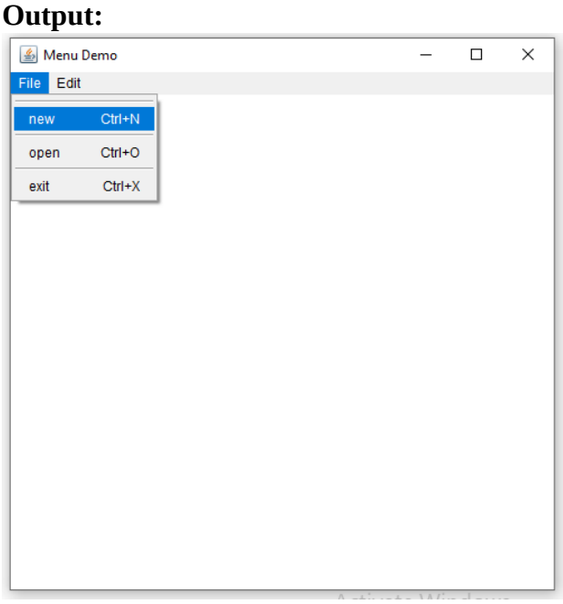
Here, item is the item being added. Items are added to a menu in the order in which the calls to add( ) take place.

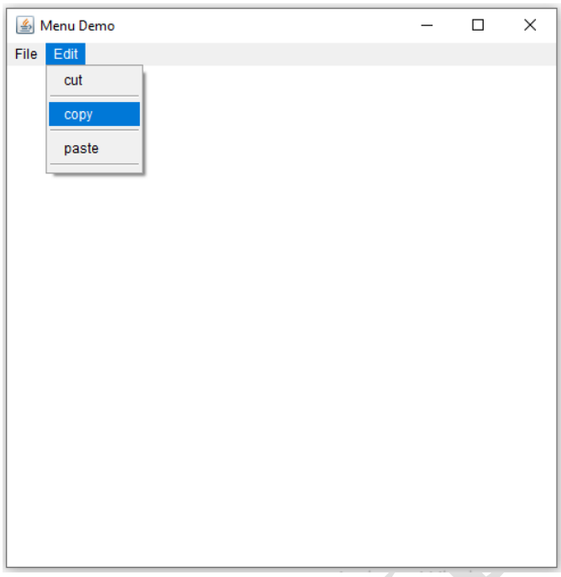
Example:

```
import java.awt.*;
import java.awt.event.*;
public class MenuDemo extends Frame implements ActionListener
{
	public void menuDemo()
	{
		MenuBar mb=new MenuBar();
		setMenuBar(mb);
		MenuShortcut n=new MenuShortcut(KeyEvent.VK_N);
		MenuShortcut o=new MenuShortcut(KeyEvent.VK_O);
		MenuShortcut x=new MenuShortcut(KeyEvent.VK_X);
			Menu filemenu=new Menu("File");
			Menu editmenu=new Menu("Edit");
				MenuItem newAction=new MenuItem("new",n);
				MenuItem openAction=new MenuItem("open",o);
				MenuItem exitAction=new MenuItem("exit",x);
				MenuItem cutAction=new MenuItem("cut");
				MenuItem copyAction=new MenuItem("copy");
				MenuItem pasteAction=new MenuItem("paste");
	newAction.addActionListener(this);
	openAction.addActionListener(this);
	exitAction.addActionListener(this);
	filemenu.addSeparator();
	filemenu.add(newAction);
	filemenu.addSeparator();
	filemenu.add(openAction);
	filemenu.addSeparator();
	filemenu.add(exitAction);
		mb.add(filemenu);
	cutAction.addActionListener(this);
```

```
        copyAction.addActionListener(this);
        pasteAction.addActionListener(this);
        editmenu.add(cutAction);
        editmenu.addSeparator();
        editmenu.add(copyAction);
        editmenu.addSeparator();
        editmenu.add(pasteAction);
        editmenu.addSeparator();
        mb.add(editmenu);

        addWindowListener(new WindowAdapter()
        {
                public void windowClosing(WindowEvent we)
                {
                        System.exit(0);
                }
        });
                  }
        public void actionPerformed(ActionEvent ae)
        {
                String action=ae.getActionCommand();
                        if(action.equals("new"))
                        {
                                System.out.println("new");
                        }
                        else if(action.equals("open"))
                        {
                                System.out.println("file");
                        }
                        else if(action.equals("exit"))
                        {
                                System.exit(0);
                        }
                        else if(action.equals("cut"))
                        {
                                System.out.println("cut");
                        }
                        else if(action.equals("copy"))
                        {
                                System.out.println("copy");
                        }
                        else if(action.equals("paste"))
                        {
                                System.out.println("paste");
                        }
        }
        public static void main(String[] poojitha)
        {
                MenuDemo md=new MenuDemo();
                md.menuDemo();
                md.setSize(500,500);
                md.setTitle("Menu Demo");
                md.setVisible(true);
        }
}
```

Output:





Dialog Boxes:

The Dialog control represents a top level window with a border and a title used to take some form of input from the user. It inherits the Window class.

Unlike Frame, it doesn't have maximize and minimize buttons.

Dialog boxes are of type Dialog and has the following constructors.

Dialog(Frame parentWindow, boolean mode)

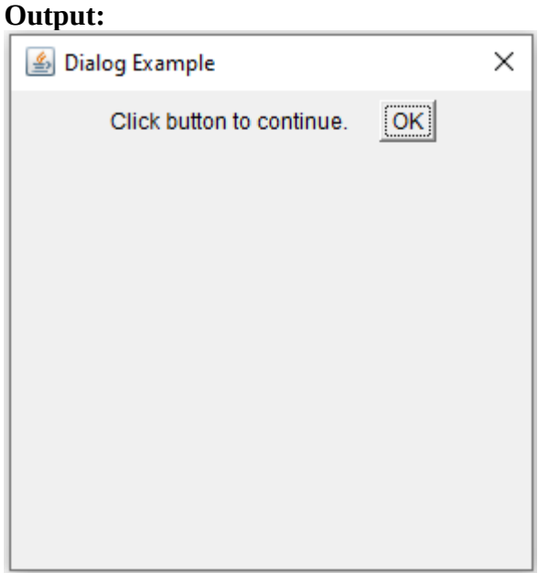
Dialog(Frame parentWindow, String title, boolean mode)

Here, parentWindow is the owner of the dialog box. If mode is true, the dialog box is modal. Otherwise, it is modeless. When the dialog box is closed, dispose( ) is called.

Example:

```
import java.awt.*;
import java.awt.event.*;
public class DialogExample extends Frame
{
   private static Dialog d;
   public DialogExample()
         {
                  Frame f=new Frame();
      d = new Dialog(f, "Dialog Example", true);
      d.setLayout( new FlowLayout() );
      Button b = new Button ("OK");
      b.addActionListener ( new ActionListener()
      {
         public void actionPerformed( ActionEvent e )
         {
            DialogExample.d.setVisible(false);
         }
      });
      d.add( new Label ("Click button to continue."));
      d.add(b);

                  d.setSize(300,300);
      d.setVisible(true);

   }
   public static void main(String args[])
   {
                  DialogExample de=new DialogExample();

   }
}
```

Output:

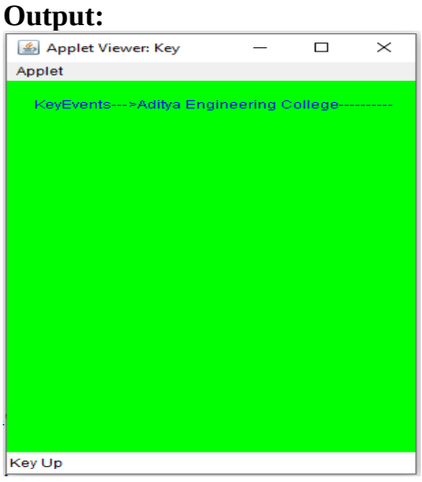


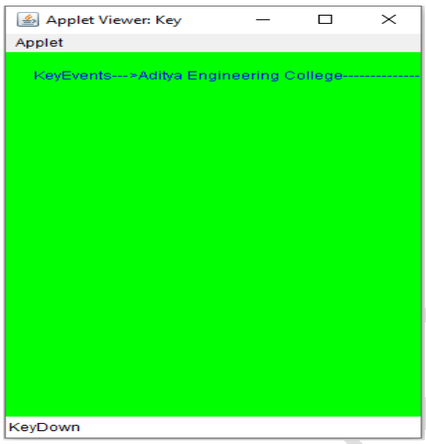
KeyEvent Handling:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*<applet code="Key" width=300 height=400></applet>*/
public class Key extends Applet implements KeyListener
{
	int X=20,Y=30;
	String msg="KeyEvents--->";
	public void init()
	{
		addKeyListener(this);
		requestFocus();
		setBackground(Color.green);
		setForeground(Color.blue);
	}
	public void keyPressed(KeyEvent k)
	{
		showStatus("KeyDown");
		int key=k.getKeyCode();
		switch(key)
		{
			case KeyEvent.VK_UP:
				showStatus("Move to Up");
				break;
			case KeyEvent.VK_DOWN:
				showStatus("Move to Down");
				break;
			case KeyEvent.VK_LEFT:
				showStatus("Move to Left");
				break;
			case KeyEvent.VK_RIGHT:
				showStatus("Move to Right");
				break;
		}
		repaint();
	}
	public void keyReleased(KeyEvent k)
	{
		showStatus("Key Up");
	}
	public void keyTyped(KeyEvent k)
	{
		msg+=k.getKeyChar();
		repaint();
	}
```

```
	public void paint(Graphics g)
	{
		g.drawString(msg,X,Y);
	}
}
```

Output:



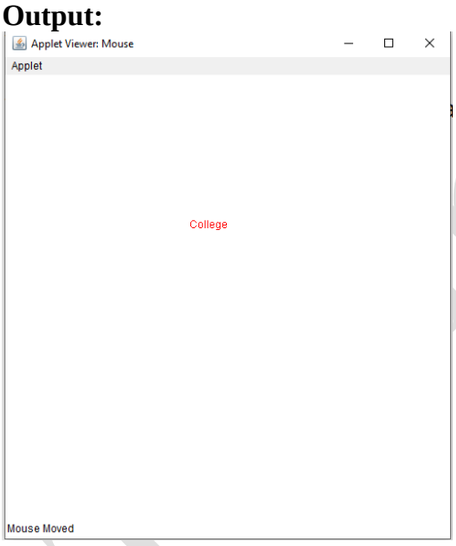


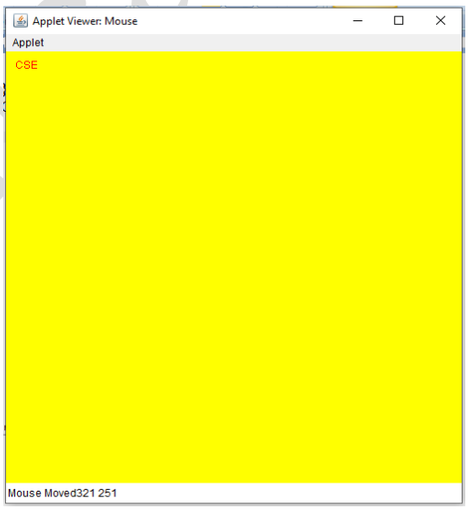
MouseEvent Handling:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*<applet code="Mouse" width=500 height=500></applet>*/
public class Mouse extends Applet
implements MouseListener,MouseMotionListener
{
	int X=0,Y=20;
	String msg="MouseEvents";
	public void init()
	{
		addMouseListener(this);
		addMouseMotionListener(this);
		setBackground(Color.black);
		setForeground(Color.red);
	}
	public void mouseEntered(MouseEvent m)
	{
		setBackground(Color.magenta);
		showStatus("Mouse Entered");
		repaint();
	}

	public void mouseExited(MouseEvent m)
	{
		setBackground(Color.black);
		showStatus("Mouse Exited");
		repaint();
	}
	public void mousePressed(MouseEvent m)
	{
		X=10;
		Y=20;
		msg="dcme3c";
		setBackground(Color.green);
		repaint();
	}
	public void mouseReleased(MouseEvent m)
	{
		X=10;
		Y=20;
		msg="advanced java programming";
```

```
                setBackground(Color.blue);
                repaint();
        }
        public void mouseMoved(MouseEvent m)
        {
                X=m.getX();
                Y=m.getY();
                msg="College";
                setBackground(Color.white);
                showStatus("Mouse Moved");
                repaint();
        }
        public void mouseDragged(MouseEvent m)
        {
                msg="CSE";
                setBackground(Color.yellow);
                showStatus("Mouse Moved"+m.getX()+" "+m.getY());
                repaint();
        }
        public void mouseClicked(MouseEvent m)
        {
                msg="Students";
                setBackground(Color.pink);
                showStatus("Mouse Clicked");
                repaint();
        }
        public void paint(Graphics g)
        {
                g.drawString(msg,X,Y);
        }
}
```

Output:





Adapter classes:

- ✓ When a listener includes many abstract methods to override, the coding becomes heavy to the programmer.
  E.g.: To close the frame, you override seven abstract methods of window Listener, in which, infact you are using only one method. To avoid this heavy coding, the designers come with another group of classes is known as "adapters".
- ✓ Adapters are abstract classes defined in java.awt.event package. Every listener that has more than one abstract method has got a corresponding adapter class.
- ✓ Java provides a special feature, called on adapter class, that can simplify the creation of event handlers in certain situations
- ✓ An adapter class provides an empty implementation of all methods in an event listener interface.
- ✓ Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface.

MouseMotionListener interface:

Example :

```
public interface MouseMotionListener
    {
       public void mouseDragged(MouseEvent e);
       public void mouseMoved(MouseEvent e);
    }
```

Corresponding adapter classes:

```
public class mouseMotionAdapter implements MouseMotionListener
{
        public void mouseDragged (MouseEvent e){}
        public  void mouseMoved(MouseEvent e){}
}
```

How to use adapter classes:

If you are not using adapter classes, your event Handler class needs to implement listener interface

```
public class Handlerclass implements MouseMotionListener.
{       _ _ _ _ _ _ _ _ _ _ _ _
}
public class Handlerclass extends MouseMotionAdapter.
{
        _ _ _ _ _ _
        _ _ _ _ _ _
}
```

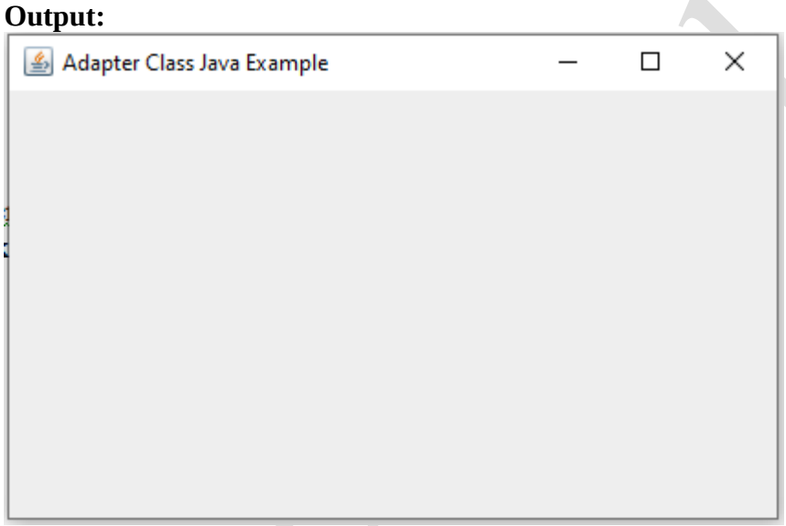
List of Adapter classes:

Listener	Adapter classes	registration method
ComponentListener	ComponentAdapters	addComponent Listener ()
ContainerListener	ContainerAdapter	addContainerListener()
FocusListener	FocusAdapter	addFocusListener ()
HierarchyBounds Listener	HierarchyBoundsAdapter	addHierarchyBounds Listener()
KeyListener	KeyAdapter	addkeyListener()
MouseListener	MouseAdapter	addkeyListener ()
MouseMotionListener	MouseMotionAdapter	addMouseMotion Listener ()
WindowListener	WindowAdapter	addWindowListener()

Example:

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
class AdapterExample extends JFrame
{
      AdapterExample()
      {
        this.addWindowListener( new WindowAdapter() {
        public void windowClosing(WindowEvent e)
                    {
                      System.exit(0);
                    }
                    });
      }
}
 class AdapterClassJavaExample
{
     public static void main(String[] args)
     {
       AdapterExample frame = new AdapterExample();
       frame.setTitle("Adapter Class Java Example");
       frame.setBounds(100,200,200,200);
       frame.setVisible(true);
     }
}
```

Output:



Layout Managers:-

When we are trying to add the components into container without using layout manager the components are overriding hence the last added component is visible on the container instead of all.

To overcome above problem to arrange the components into container in specific manner use layout manager.

Definitions:

The layout managers are used to arrange the components in a Frame in particular manner or

A layout manager is an object that controls the size and the position of components in a container.

Different layouts in java,

1) java.awt.FlowLayout
2) java.awt.BorderLayout
3) java.awt.GridLayout
4) java.awt.CardLayout
5) java.awt.GridBagLayout

java.awt.FlowLayout

The FlowLayout is used to arrange the components into row by row format. Once the first row is filled with components then it is inserted into second row. And it is the default layout of the applet.

java.awt.BorderLayout:-

The BoderLayout is dividing the frame into five areas north,south,east,west,center so we can arrange the components in these five areas.

To represent these five areas borderlayout is providing the following 5-constants

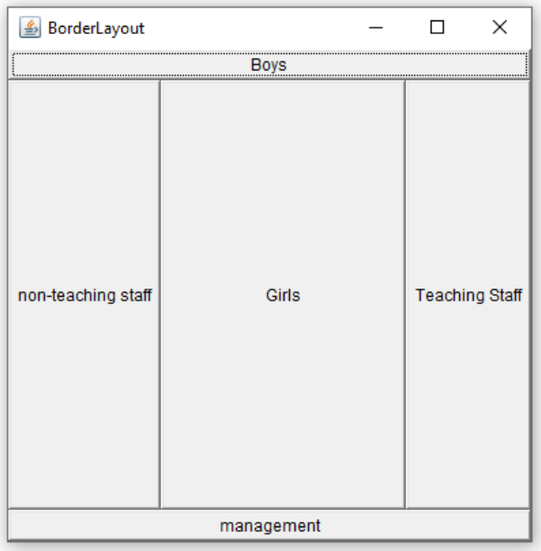
public static final java.lang.String NORTH;
public static final java.lang.String SOUTH;
public static final java.lang.String EAST;
public static final java.lang.String WEST;
public static final java.lang.String CENTER;

Example:

```
import java.awt.*;
class MyFrame extends Frame
{
	Button b1,b2,b3,b4,b5;
	MyFrame()
	{
		//this keyword is optional because all methods are current class methods only
		this.setSize(400,400);
		this.setVisible(true);
		this.setTitle("BorderLayout");
		this.setLayout(new BorderLayout());
		b1=new Button("Boys");
		b2=new Button("Girls");
		b3=new Button("management");
		b4=new Button("Teaching Staff");
		b5=new Button("non-teaching staff");
		this.add("North",b1);
		this.add("Center",b2);
		this.add("South",b3);
		this.add("East",b4);
		this.add("West",b5);
	}
}
class Demo
{
	public static void main(String[] args)
	{
		MyFrame f=new MyFrame();
	}
}
```

Output:

```
D:\Java_Programs\AWT>javac BorderLayoutDemo.java

D:\Java_Programs\AWT>java Demo
```



Grid Layout:

The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.

Constructors of GridLayout class

GridLayout(): creates a grid layout with one column per component in a row.

GridLayout(int rows, int columns): creates a grid layout with the given rows and columns but no gaps between the components.

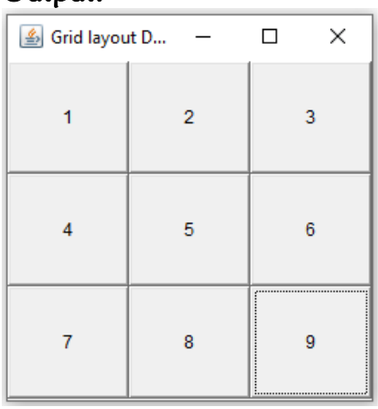
GridLayout(int rows, int columns, int hgap, int vgap): creates a grid layout with the given rows and columns alongwith given horizontal and vertical gaps.

Example:

```
import java.awt.*;
class GridLayoutDemo extends Frame
{
   Button b1,b2,b3,b4,b5,b6,b7,b8,b9;

        public GridLayoutDemo()
        {
                setLayout(new GridLayout(3,3));

                b1=new Button("1");
                b2=new Button("2");
                b3=new Button("3");
                b4=new Button("4");
                b5=new Button("5");
                b6=new Button("6");
                b7=new Button("7");
                b8=new Button("8");
                b9=new Button("9");

                add(b1);
                add(b2);
                add(b3);
                add(b4);
                add(b5);
                add(b6);
                add(b7);
                add(b8);
                add(b9);

        }
```

```
        public static void main(String args[])
        {
                GridLayoutDemo g=new GridLayoutDemo();
                g.setTitle("Grid layout Demo");
                g.setSize(200,200);
                g.setVisible(true);
        }
}
```

Output:



Card Layout:

The CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.

Constructors of CardLayout class

CardLayout(): creates a card layout with zero horizontal and vertical gap.

CardLayout(int hgap, int vgap): creates a card layout with the given horizontal and vertical gap.

Commonly used methods of CardLayout class

public void next(Container parent): is used to flip to the next card of the given container.

public void previous(Container parent): is used to flip to the previous card of the given container.

public void first(Container parent): is used to flip to the first card of the given container.

public void last(Container parent): is used to flip to the last card of the given container.

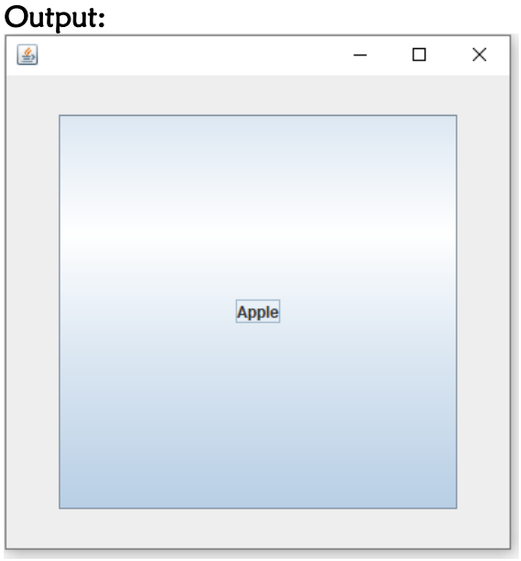
public void show(Container parent, String name): is used to flip to the specified card with the given name.

Example:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class CardLayoutExample extends JFrame implements ActionListener
{
        CardLayout card;
        JButton b1,b2,b3;
        Container c;
        CardLayoutExample(){

       c=getContentPane();
       card=new CardLayout(40,30);
      //create CardLayout object with 40 hor space and 30 ver space
       c.setLayout(card);

       b1=new JButton("Apple");
       b2=new JButton("Boy");
       b3=new JButton("Cat");
       b1.addActionListener(this);
       b2.addActionListener(this);
       b3.addActionListener(this);

         c.add("a",b1);
         c.add("b",b2);
         c.add("c",b3);
```

```
    }
    public void actionPerformed(ActionEvent e)
    {
        card.next(c);
    }

    public static void main(String[] args)
    {
        CardLayoutExample cl=new CardLayoutExample();
        cl.setSize(400,400);
        cl.setVisible(true);
        cl.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```

Output:

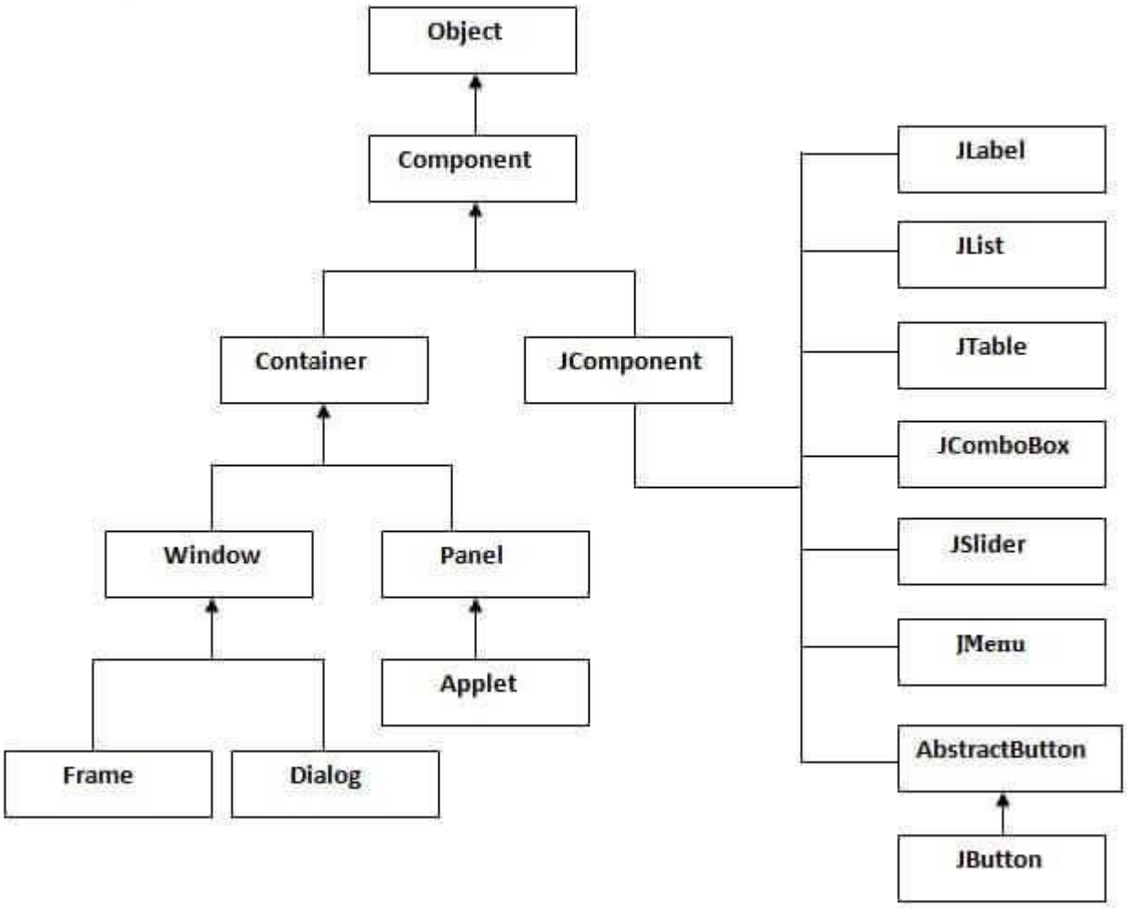


SWING

Differences between awt and swing:

S.NO	AWT	Swing
1.	Java AWT is an API to develop GUI applications in Java	Swing is a part of Java Foundation Classes and is used to create various applications.
2.	The components of Java AWT are heavy weighted.	The components of Java Swing are light weighted.
3.	Java AWT has comparatively less functionality as compared to Swing.	Java Swing has more functionality as compared to AWT.
4.	The execution time of AWT is more than Swing.	The execution time of Swing is less than AWT.
5.	The components of Java AWT are platform dependent.	The components of Java Swing are platform independent.
6.	MVC pattern is not supported by AWT.	MVC pattern is supported by Swing.
7.	AWT provides comparatively less powerful components.	Swing provides more powerful components.

Swing Hierarchy:

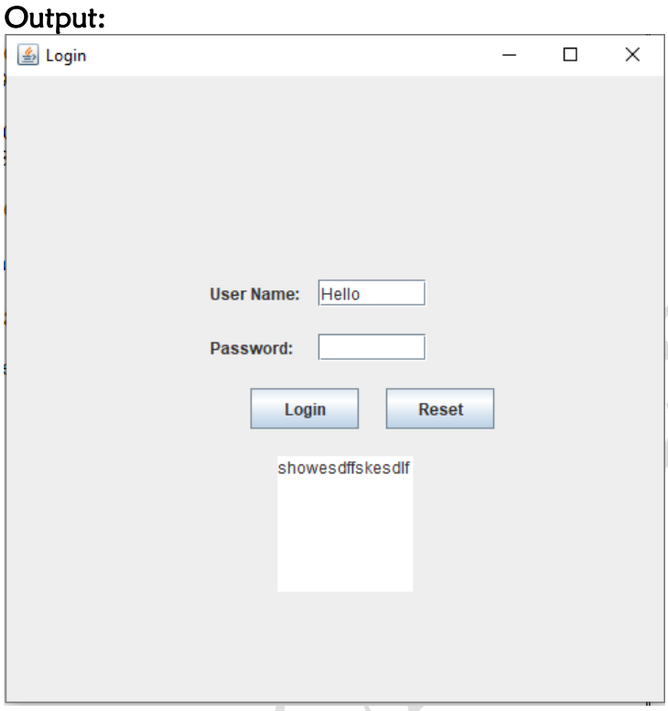


Example:

```
import javax.swing.*;
class LoginDemo extends JFrame
{
        JLabel l1,l2;
        JButton b1,b2;
        JTextField tf1;
        JPasswordField tf2;
        JTextArea ta;

        public LoginDemo()
        {
                setLayout(null);
                l1=new JLabel("User Name:");
                l2=new JLabel("Password:");
                b1=new JButton("Login");
                b2=new JButton("Reset");
                tf1=new JTextField("Hello",25);
                tf2=new JPasswordField(25);


                ta=new JTextArea("showesdffskesdlf",40,40);
                add(l1);
                add(tf1);

                add(l2);
                add(tf2);

                add(b1);
                add(b2);
                add(ta);

                l1.setBounds(150,150,80,20);
                l2.setBounds(150,190,80,20);
```

```
            tf1.setBounds(230,150,80,20);
            tf2.setBounds(230,190,80,20);

            b1.setBounds(180,230,80,30);
            b2.setBounds(280,230,80,30);

            ta.setBounds(200,280,100,100);

            System.out.println(ta.getText());
        }
        public static void main(String args[])
        {
        LoginDemo d=new LoginDemo();
                d.setTitle("Login");
                d.setSize(500,500);
                d.setVisible(true);
        }
}
```

Output:



JPanel:

The JPanel is a simplest container class. It provides space in which an application can attach any other component. It inherits the JComponents class.

It doesn't have title bar.

Commonly used Constructors:

Constructor	Description
JPanel()	It is used to create a new JPanel with a double buffer and a flow layout.
JPanel(boolean isDoubleBuffered)	It is used to create a new JPanel with FlowLayout and the specified buffering strategy.
JPanel(LayoutManager layout)	It is used to create a new JPanel with the specified layout manager.

JTree:
The JTree class is used to display the tree structured data or hierarchical data. JTree is a complex component. It has a 'root node' at the top most which is a parent for all nodes in the tree. It inherits JComponent class.

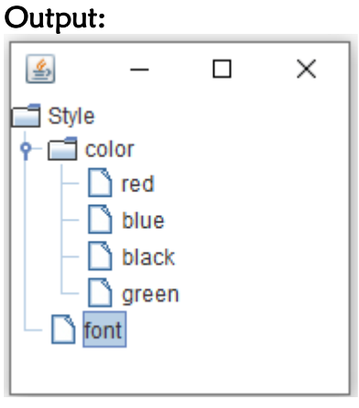
Commonly used Constructors:

Constructor	Description
JTree()	Creates a JTree with a sample model.
JTree(Object[] value)	Creates a JTree with every element of the specified array as the child of a new root node.
JTree(TreeNode root)	Creates a JTree with the specified TreeNode as its root, which displays the root node.

Example:

```
import javax.swing.*;
import javax.swing.tree.DefaultMutableTreeNode;
class TreeExample extends JFrame
{
        public TreeExample()
        {
                DefaultMutableTreeNode style=new DefaultMutableTreeNode("Style");
                DefaultMutableTreeNode color=new DefaultMutableTreeNode("color");
                DefaultMutableTreeNode font=new DefaultMutableTreeNode("font");
                style.add(color);
                style.add(font);
                DefaultMutableTreeNode red=new DefaultMutableTreeNode("red");
                DefaultMutableTreeNode blue=new DefaultMutableTreeNode("blue");
                DefaultMutableTreeNode black=new DefaultMutableTreeNode("black");
                DefaultMutableTreeNode green=new DefaultMutableTreeNode("green");
                color.add(red);
                color.add(blue);
                color.add(black);
                color.add(green);
                JTree jt=new JTree(style);
                add(jt);

        }
        public static void main(String[] args)
        {
                TreeExample f=new TreeExample();
                f.setSize(200,200);
                f.setVisible(true);
        }
}
```

Output:



JTable:
The JTable class is used to display data in tabular form. It is composed of rows and columns.

Constructor	Description
JTable()	Creates a table with empty cells.
JTable(Object[][] rows, Object[] columns)	Creates a table with the specified data.

Example:

```
import javax.swing.*;
class TableExample extends JFrame
{
    TableExample()
        {
            String data[][]={ {"101","Amit","670000"},
                              {"102","Jai","780000"},
                              {"101","Sachin","700000"}};

            String column[]={"ID","NAME","SALARY"};

            JTable jt=new JTable(data,column);
            jt.setBounds(30,40,200,300);
            JScrollPane sp=new JScrollPane(jt);
            add(sp);

        }
        public static void main(String[] args)
        {
            TableExample f=new TableExample();
            f.setSize(300,400);
            f.setVisible(true);
        }
}
```

Output: