Java language has provided a keyword called "extends" through which we can acheive "Reusability"

The slogan of Java is WORA (Write Once Run Anywhere)

Difference between Strongly(Statically) vs Loosly(Dynamically):-
----
The languages in which data type is compulsory before initialization of a variable is called Strongly or Statically typed langauge
Ex:- C,C++, JAVA
Where as on the other hand the languages where data type is not compulsory and it is optional then it is called Loosly or Dynamically Typed language.
Ex:- Python, JavaScript, VisualBasic

Why java becomes so popular:-

C and C++ are platform dependent language so the .exe file generated in one machine will not be executed on another machine if the configuration of the machine is changed.

That is the reason c and c++ languages are not suitable for web development.

The data which are tightly coupled or strongly associated with the function are called encapsulation where as the function which are not associated with the data and it is written inside the class are the function of class.

Inheritance :-

Deriving a new class from existing class in such a way that the new class will acquire all the features and properties (except private) is called inheritance.

Polymorphism :-

Poly means "many" and morphism means "forms", It is a Greek word whose meaning is Same object having different behavior.

Description of main():-

Any program execution always starts from main() as well as execution of the program always ends with main() only

public :-

Our main method must be declared as public because JVM is responsible to execute our java program via main method, if the main method is not declared as public, JVM can't access our main method so the program will not be executed.

In order to call a method/function in Object oriented programmimg, First of all we need to create an object.

static :-

In Java our main method is declared as static, so in order to call the main method we need not to create an Object.
A static method can be directly call with the help of class name.

void :-

It is a data type. we apply void before the name of the method so the method will not return any value. void means empty or no return type.

Command Line Argument :-

Whenever we pass an argument to the main method then it is called Command Line Argument.

In the statement,   String [] args

String :- It is a predefined class available in java.lang package
[] args :- It is an array variable of type String.

System.out.println() :-

It is an output statement in java through which we can print the variable value, String constant and any general statement.

In this statement System is a predefined class available in java.lang package, out is a reference variable and println() is predefined method of PrintStream class.

Command Line Argument :-

Whenever we pass any argument to the main method then it is called Command line argument.

By using command line argument we can pass some value at runtime.

The benefit of command line argument is single time compilation and number of time execution.

How to convert a String into Integer :-

If we want to convert a String into Integer then java software people has provided a predefined class called Integer available in java.lang package
This Integer class contains a predefined method called parseInt(), through which we can convert a String value to an integer value.

The return type of parseInt() is int as well as it is a static method so we can directly call this method with the help of class name.

Number System :-

It is a technique to describe the quantity.
Eg:- 100
Octal Literal :-

Any integral literal prefix with '0' is called octal Literal. As we know the range of octal number is 0 to 7 so we can accept any digits in between 0 to 7 only.

```
Eg:- int x = 017; valid
       int y = 018; Invalid
```

Hexadecimal Literal :-

Any integral literal prefix with 0X(zero capital X) or 0x (zero small x) is called Hexadecimal Literal. As we know the range of hexadecimal literal is 0 to F so we can any digits in between 0 to F only.

```
Eg:  int x = 0Xadd; //valid
```

```
       int y = 0Xage; //Invalid
```

Binary Literal :-

It came from java 1.7 onwards. Any integral literal prefix with 0B(zero capital B) or 0b(zero small b) is called Binary Literal. As we know the range of binary literal from 0 to 1 so we can accept only two digits i.e 0 and 1

```
Eg:    int x = 0B111; valid
         int y = 0b112; invalid
```

boolean Literal :-
----
1) boolean literal contains only two values i.e true or false.

2) It accepts only one bit of memory.

3) Unlike C and C++ we can't assign intger value to boolean

```
   Ex:- boolean b = 0; XXX(Invalid)
        boolean c = 1; XXX(Invalid)
```

4) We can't assign the following String type to boolean data type

```
        boolean d = "true";  XXX(Invalid)
        boolean e = "false"; XXX(Invalid)
```

char Literal :-

There are various ways to represent char literal

1) Single character enclosed with single quotes.

```
    eg:-  char ch = 'A';
```

2) We can assign integral literal to char literal to represent UNICODE values. The range of UNICODE value is 0-65535.
C and C++ are using ASCII value, the Range of ASCII is 0-255 but to represent all characters, java uses UNICODE values so we can assign any value in between 0-65535.

```
     eg:- char ch = 65535;
```

3) We can represent UNICODE Values in 4 digit hexadecimal number. The format is '\uXXXX'

```
      minimum value :-  '\u0001'
      maximum value:-  '\uffff'      (Hexadecinal so range is 0-f)
```

4) By using char literal we can represent escape sequences.

String Literal in java:-

We can create a String in java using various ways :-

1) By using character array

```
   Ex:-  char ch[] = {'H','E','L','L','O'};
```

```
2) By using String Literal
   String str = "Hello";

3) By using new keyword
   String x = new String("Hyderabad");
```

Whenever we create String object using String literal then the String object will be created in a special memory called String constant pool as a part of Heap memory.
Once the String object is created we can't modify and delete the String object.

Now Whenever we create a String object using new keyword then the String object will be created inside the heap memory but not inside the String constant pool.
The String object created at heap memory can be deleted but can't be modified.

Facts about String memory :-

Whenever we create a String object, first of all JVM will verify that the String object is already available in the String constant pool or not.
If it is already available then JVM will not create any new String Object, It will simply assign the new reference variable to the old String object.

Why Strings are immutable :-

Strings are immutable (unchanged) why because as we know String object can be reffered by multiple reference variables and if any of the reference variable modifies the value of String then It would be very difficult for the other reference variable to get the same value what they have defined earlier.
That is the reason Strings are immutable in java.

public char charAt(int index):-

It is a predefined method available in String class. The main purpose of this method to extract or fetch a single character from the given String. The return type of this method is char, why because it returns a single character.
public String concat(String x):-

It is a predefined method of String class. The main purpose of this method is to join or append or concatenate two Strings. The return type of this method is String.
We can also use '+' operator to join two Strings.

public boolean equals(String x) :-

It is a predefined method of String class. It is used to compare two String and verify whether both the Strings are equal or not, if both the Strings are equal then It will return true otherwise It will return false. It is a case sensitive method.

public boolean equalsIgnoreCase(String x) :-
===========================
It is a predefined method of String class. It is used to compare two Strings by ignoring their case that means 'a' and 'A' both are same. This method will also return boolean value.

public int length() :-

It is a predefined method of String class. It is used to find out the length of the given String. The return type of this method is int. The counting length of a String always starts from 1.

public String replace(char old, char new)
public String replace(String old, String new)

It is a predefined method in String class. By using replace method we can replace a particular character or a token of String from the existing String. The return type of this methos is String.

public int compareTo(String s) :-

It is a predefined method in String class. If we want to compare two Strings character by character based on the UNICODE values then it is called lexicographical comparison.

comapareTo() of String class uses lexicographical comparison to compare two String. Based on the comparison It will return
0 or +ve or -ve

public String substring(int startIndex) :-
public String substring(int startIndex, int endIndex) :-

This method is used to fetch the part of the main String so called as substring(). Both the parameters are inclusive but startIndex will start from 0 whereas endIndex will start from 1.
The return type of this method is String. If both the parameters are same , It will not print anything and if first parameter is greater than 2nd parameter then It will generate an exception i.e StringIndexOutOfBoundsException
We can't pass negative parameter in substring method.

StringBuffer :-

It is a prdefined class available in java.lang package from jdk 1.0 version. It is a mutable class that means we can easily modify the object of StringBuffer class.

All the methods defined inside the StringBuffer class is synchronized methods so if multiple threads want to access the method of StringBuffer class then it is not possible.

Only one thread can enter inside the method of StringBuffer class, hence performance wise it is slow.

Drawback of if condition :-
----
while working with if condition we need to check the condition again and again so there will be extra burdon on CPU. to avoid this problem we introduced switch-case statement.

In switch case whatever the value we will pass, with that value the appropriate case will be executed if the case is not available then default statement will be executed.

Note :- break keyword is used to break the switch statement.

While working with switch case we can't pass long,float and double value.

Loop in Java :-

A loop is nothing but repeatation of statement that means by using loop we can

repeat statement of statements 'n' number of times.

Java supports 4 kinds of loop

1) do-while loop

2) while loop

3) for loop

4) for-each loop

Working with class and object :-

Whenever we write a class in java and if we have not defined any kind of constructor to the class then automatically the compiler will add one default constructor to the class.

Why compiler adds default constructor to the class? :-

Every java class has a constructor either written by user explicitly, or added by the compiler implicitly.

The compiler adds default constrctor to our class otherwise object creation is not possible in java.

The main purpose of constructor (added by the compiler) to provide default values for variables declared inside a class.
Ex:-
int -> 0
float -> 0.0
String-> null and so on

How to take input from the user :-

There is a predefined class available in java.util package called Scanner, This class is used to take the input from the user.

The following command describes how to create an object for Scanner class

```
Scanner sc = new Scanner(System.in);
```

static variables of System class :-

System is a predefined class available in java.lang package.It contains 3 static variables

1) System.out :- It is used to print normal message on the screen

2) System.err :- It is used to print the error messages on the scrren

3) System.in :- It is used to accept input from the keyboard

Methods of Scanner class :-
----
next() :- used to read a single word

nextLine() :- Used to read multiple word or complete line

nextInt() :- Used to read int value

nextFloat() :- Used to read float value

nextDouble() :- Used to read double value

nextLong() :- Used to read long value

nextBoolean() :- Used to read boolean value

Operators :-
----
1) Binary operators :- It is also known as Arithmetic Operator. It works with two oprands that is the reason it is called Binary Operators.
Eg:

+ , - , *, / , %

Division Operator :-

While working with integeral literal when we divide a number by zero i.e (10/0 = Infinity and 0/0=Undefined) then there is no way to reprsent this Infinity and Undefined so in both the cases we will get java.lang.ArithmeticException and the execution sequence of our program will be stopped.

On the other hand if we divide a number by 0.0 i.e floating point literal then java.lang.Float and java.lang.Double, both classes have been defined POSITIVE_INFINITY , NEGATIVE_INFINITY and NaN (Not a number) constants to represent positive infinity, negative infinity and undefined respectively.

Unary Operator :-

It works on single operand. we have 3 kinds of unary operator.

1) Unary minus(-)

2) Increment operator (++)

3) Decrement Operator (--)

Local varibale :-

The variables which are declared inside a method either as a method body or as a method parameter are called Local / Stack / Temporary / Automatic variable.

As per as its scope is concerned It can be accessible within the same method only.

A local variable must be initialized as well as we can't use access modifier (except final) on local variable.

Assignment Operator :-

It is used to assign right hand side value to left hand side. Both the values must be compatible with each other.

Relational Operator :-

These operators are used to compare the values. We have total 6 relational operator

1) > (Greater than)

2) >= (Greater than or equal)
3) <   (less than)
4) <= (less than or equal)
5) == (double equal to)
6) != (Not equal to)

Note :- These operators always return boolean type.

If condition :-

It is used to check condition and based on the condition, it will execute the statement. The return type of if condition is always boolean value.

```
if(condition)
{
   statement1;
}
else
{
   statement2;
}
```

Nested if :-
----
if we place an if condition inside another if condition then it is called Nested if condition.

```
if(condition) //outer if
{
   if(condition) //Nested if condition (Inner if)
   {
   }
   else
   {
   }
}
else
{
}
```

Logical Operator :-

It is used to join or compound two or more statements in a single condition.
we have three kind of logical opertors

1) && AND Operator

2) || OR Operator

3) ! Not Operator

&& -> all the conditions must be true

|| -> at least one must be true

! -> It is an inverter so it makes true as a false and false as a true

Boolean Operators :-

These operators will work with boolean values.

& Boolean AND operator

| Boolean OR operator

! Boolean Not operator

& :- All the conditions must be true

| :- At least one condition must be true

! :- It is an inverter

Bitwise Operator :-

These operators are used to work with bit by bit operation. we have 3 kinds of bitwise operator

& Bitwise AND Operator

| Bitwise OR Operator

^ Bitwise X-OR Operator

& -> all conditions must be true

| -> at least one condition must be true

^ -> It will return true if both the arguments are alternate to each other.

29-12-2021

Bitwise Complement Operator :-

This operator does not work with booelan. It can work with only integer. It is represented by ~

Ternary Operator or Conditional operator (?:)

This operator is mainly used to reduce the size of if condition. It is called Ternary opertaor why because It uses 3 operators.

Member Access Operator(.) :-

This operator is represented by . (dot). It is used to access the member of the class like variables and methods.

new Operator :-

This operator is used to create the object. With the help of object referenece we can invoke the method or variable of the class.

If we have an instance variable or instance method (the non-static variable and method) then in order to call the method first of all we need to create an object.

instanceof operator :-

This operator is used to verify whether a reference variable is the instance of the particular object or not. If it is the instance of particular object then it will return true otherwise it will return false.

default values of the instance variable with class and object :-

Whenever we write a class in java and if we have not written any kind of constructor then one defualt constructor would be added by the compiler.

Now the main purpose of this constructor to initialize the instance variable of the class with some meaningful value called as default value.
int -> 0
float -> 0.0f
double -> 0.0d
char -> '\u0000'
String -> null
boolean -> false

Why we pass parameter to a function ?

We pass parameter to a function for getting more information regarding the function.

As we know compiler automatically add one default constructor to the class, the purpose of this default constructor to provide default values like for int -> 0, for float 0.0 and so on.

But these default values are not useful for the user hence we need to take a seperate method (input()) to re-initialize the variable value as shown in the above program.

Instance variable :-

The non-static variables which are declared inside the class but outside of the method are called instance variable.

As per as its scope is concerned the instance variable can be accessible from anywhere but within the same class only.

The life of an instance variable starts at the time of creating the object that is the reason all the instance variables are initialized with default values at the time of creating the object.

Local Variable :-

The variables which are declared inside the method either as method body or as a method parameter are called local/stack/temporary/automatic variable.

As per as its scope is concerned the local variables are accessible within the same method only.

A local variable must be initialized as well as we can't apply any kind of access modifier like private, protected and public to the local variable.

Object Oriented programming :-

If we put everything in a single class then it is not an object oriented programming. In oder to acheive the object oriented programming classes must be loosly coupled with each other.

In java we can develop two kinds of classes
1) BLC(Business Logic class)
2) ELC(Executable Logic class)

this keyword :-

Whenever insatance variable and local variable name both are same then at the time of variable initialization our runtime environment gets confused that which one is local variable and which one is instance variable, To avoid this problem we should use this keyword.

this keyword always refers to the current object and we know instance variables are also the part of current object so JVM can easily understand about instance variables because they are associated with 'this' keyword.

What is the benefit of writing constructor in our program :-

If we write constructor in our program then compiler will not add any kind of constructor, we initialize our all instance veriables inside the constructor so variable initialization and variable re-initialization both are done in the same place as shown in the diagram below.

Constructor :-

It is a special method whose name is same as class name or in words we can say if the class name and method name both are same then it is called constructor.

The main purpose of constructor to initialize the object that initialize the instance variable of the class.

Every java class has a constructor either implicitly added by the compiler or explicitly written by the user.

A default constructor will be added by the compiler in a class to provide default values for the instance variables in case user has not written any kind of constructor.

Constructor never containing any return type including void also.

A constructor is automatically called and executed at the time of creating the object.

A constructor is called only once per object that means when we craete the object constructor will be called and executed, if we create the object again then again the constructor will be called and executed.

Java supports 4 kinds of constructor
----
1) default constructor

2) parameterized constructor

3) copy constructor

4) private constructor

default constructor :-

If no argument is passed to the constructor then it is called default constructor.

Eg:-

```
class Test
{
     Test() //default constructor
      {
      }
}
```

parameterized constructor :-

If one or more argument is passed to the constructor then it is called as parameterized constructor.

Using parameterized constructor we can take value from user.

```
class Test
{
     int a, b;

     Test(int a, int b)   //Parameterized constructor
     {
        this.a = a;
      this.b = b;
     }
}
```

Note :-

Whenever we create an object in java then a seperate copy of all the instance variables will be created with each object reference.
Copy constructor :-

Whenever we pass an object reference to the constructor then it is called copy constructor.
The main purpose of copy constructor to copy the content of one object to another object.

Eg:-

```
class Test
{
    int a,b;

    Test(int a, int b)   //parameterized constructor
     {
     }

     Test(Test t)  //copy constructor
     {
     }
}
```

private constructor :-

It is possible to declare a constructor as private. we should declare a constructor as private if we want to define all the methods as a static method so, It will restrict a user to create the object for the class but we can invoke the static methods with the help of class name.

instance block :

It is also known as instance initializer. The main purpose of instance blcok to initialize the instance variable of the class.

An instance block will be executed automatically whenever we create an object in java.

An instance block will be executed before the constructor.

If we have more than one instance block in a class then it would be executed from top to bottom order.

Inheritance :-

Deriving a new class from existing class in such a way that the new class will acquire all the features and properties of existing class is called Inheritance.

It is one of the most imporatnt feature of oops which provides "CODE REUSABILITY".

In java we provide inheritance using a keyword called 'extends'.

Using inheritance mechanism the relationship between the classes is parent and child, According to C++ the parent class is called Base class and the Child class is called Derived class whereas According to Java the parent class is called as super class and the child class is called as sub class.

Using inheritance all the features of super class is bydefault available to the sub class so the sub class need not to start the process from begning onwards.

Inheritance follows top to bottom approach, In this hierarchy if we move towards upward direction more generalized properties will occur and on the other hand if we move towards downward direction more specialized properties will occur.

Inheritace provides two kinds of relationship
IS-A Relation :- It occurs between the classes.
HAS-A Relation :- It occurs between the class and its property.

Why java does not support multiple inheritance :-

Whenever a sub class wants to inherit the properties of two or more super classes then there might be chance of getting memory duplication. This situation is known as Diamond Problem in Java.

That is reason Java does not support multiple inheritance. But we can acheive multiple inheritace in java using interface concept.

super keyword :-

super keyword can be used 3 ways in java

1) To call the super class variable

2) To call the super class method

3) To call the super class constructor

To call the super class variable :-

Whenever super class variable name and sub class variable name both are same and if we create an object of sub class then it will give more priority to its own class variable if we want to call super class variable then we should use super keyword.

super keyword always refers to its immediate super class as well as we should only use super keyword whenever the member of super class name and the member of sub class both are same.

To call the super class method :-

Whenever the super class method name and sub class method name both are same and if we create an object of sub class then it will give more priority to its own class method.
        If we want to invoke the super class method then we should use super

keyword.

To call the super class constructor:-

Whenever we write a class in java and if we have not written any kind of constructor then automatically the compiler will add one default constructor to the class.
The first line of any constructor is reserved either for super() or this(). If a developer does not specify either super() or this() to the first line of constructor then compiler will automatically add super() to the first line of constructor.

We have following four conclusions :

1) super() :- To call the default constructor of super class

2) super(9) :- To call the parameterized constructor of super class.

3) this() :- To call the default constructor of its own class.

4) this(12) :- To call the parameterized constructor of its own class.

Type casting in java :-

Type casting is nothing but converting one data type to another data type. Type casting is divided into two types.

1) Implicit Type casting (Automatic type casting)
2) Explicit Type casting (Mannual type casting)

Implicit Type casting :-

Whenever we want to assign a smaller data type value to the bigger data type then It is automatically done by the compiler and It is known as Implicit Type Casting.

byte -> short -> char -> int -> long -> float -> double

Explicit Type casting :-

Whenever we want to assign a bigger data type value to the smaller data type then by default the compiler will not allow this, if at all we want to assign then we need to mannually convert the bigger data type into smaller data type.

```
short s = 15;
byte b = s;  -> by default it is not possible
byte b = (byte)s;  //short is converted to byte
```

While working with explicit type casting there might be a chance of loss of data.

Access modifiers in java :-

An Access Modifiers describes how the member of the class (data + method) and the class it self would be accessible that means the scope of accessibility.

Java provides 4 access modifiers

1) private (Within the same class only)

  It is most restrictive access modifier in java, The members which are declared as private can be accessible within the same class only.
  In java we can't declare a class as a private.

2) default (Accessible within the same package(folder)):
    ----
    default is an access modifier which is less restrictive than private, It is such kind of access modifier whose physical existance is not available that means If a user does not specify any kind of access modifier then by default It would be default.
    As per as its scope is concerned default memebers are accessible within the same package i.e same folder only.

3) protected (Accessible within the same class, within the package and from
----
another package as well but using inheritance)

It is an access modifier which is less restrictive than default, The members which are declared as protected can be accessible within the same class, within the same package(folder) and from the another package as well but using inheritance.

4) public (No restriction , members are accessible from everywhere)
----

public access modifier does not contain any kind of restriction that means the members which are declared as public can be accessible from everywhere.

Note : - In java generally we declare data member as private, member function and classes as public.

Polymorphism :-

Poly means "many" and morphism means "form". It is a Greek word whose meaning is "SAME OBJECT HAVING DIFFERENT BEHAVIOR"

In our real life a person or a human being can perform so many task similarly in our programming languages a method or a constructor can perform so many task.

Polymorphism can be divided into two categories :

1) Compile time Polymorphism OR Static Polymorphism

2) Run time Polymorphism OR Dynamic Polymorphism

Compile time polymorphism :-

The polymorphism which exist at the time of compilation is called static polymorphism. In static polymorphism compiler has a very good idea that which method is invoked depending upon the type of parameter we have passed in the method.

This type of preplan polymorphism is called static or compile time polymorphism.

Ex:- Method Overloading

Runtime polymorphism :-

The polymorphism which exist at runtime is called dynamic polymorphism. In dynamic polymorphism, compiler does not have any idea that which method is invoked, at runtime only the JVM will decide that which method is invoked depending upon the class type.

This type of polymorphism is called dynamic or Runtime polymorphism Or dynamic method dispatch.

Eg:- Method Overriding

Method Overloading :-

Writing two or more methods in the same class in such a way that the method name must be same and argument must be different is called method overloading.

While working with method overloading we can change the return type of the method.

var-args :-

It stands for variable arguments. It is actually an array variable which can hold 0 parameter to n number of parameter of same type.

It is represented by ... (exactly 3 dots).By using this var-args conecpt, now we need to define only one method body for accepting different kinds of parameter.

var-args must be only one argument as well as last argument in the method otherwise there will be a compilation error.

Method Overriding :-

Writing two or more methods in super and sub class in such a way that method signature (Method name along with parameter) must be same is called Method Overriding.

Method Overriding is only possible with inheritance, if there is no inheritance there is no method overriding.

Generally we can't change the return type of the method while overriding a method but from JDK 1.6 onwards we can change the return type of the method by using a concept called Co-variant.

@Override Annotation :

It is a new Feature introduced in java from jdk 1.5 onwards. It is different from comment beacuse it ensures the compiler as well as User that the method is an overriden method and It must be available in the super class.

If we use @Override annotation and If we are not overriding the method from super class then compiler will generate an error.

Role of access modifer while overridng a method :-

Method Overriding is only possible with inheritance, if there is no inheritance there will not be method overriding so private accees modifer is allowed at overriding.

While overriding a method the access modifier of sub class method must be greater or equal to the access modifer of super class method.

The following statement described the access modifer from greater to less

public > protected > default

(public is greater than protected, protected is greater than default)

Co-variant in Java :

In general we can't change the return type of the method while overriding a method.

From JDK 1.6 onwards Java software people has provided a new concept called Co-variant through which we can change the return type of the method.

We can change the return type of the method using co-variant , co-variant describes the return type of the method must be in inheritance relationship.

final keyword in java :-

To provide some sort of restrictions we use final keyword. It can be used 3 ways in java.

1) To declare the class as a final

2) To declare the method as a final

3) To declare the variable as a final

To declare a class as a final :-

Whenever we declare a class as a final then inheritance is not possible for that class. To prevent inheritance we use final keyword to declare a class as final.

We should use final keyword when the logic of the class is very important and we don't want to share with another user, in that situation we should declare a class as a final.

final class means only class behavior is final it's variables and methods we can modify.

To declare the method as a final :-

Whenever method implementation is very important and we don't want to change the method implementation in the later stage of development then we should declare a method as final.

Once a method is declared as final we can't override that method in the child class.

To declare the variable as a final :-

Whenver we declare a variable as final then we can't perform re-assignment for that variable.

In java final variables are written by uppercase letter.

Class Loader subsystem with JVM Architectute :-

The three main component of JVM

1) class loader sub system

2) Runtime Data Areas

3) Execution engine

In order to load the required .class file, JVM makes a request to class loader sub system. The class loader sub system follows delegation hierarchy algorithm to load the required .class file from different area.

To load the required .class file we have 3 different kinds of class loader

1) Bootstrap class loader

2) Extension/Platform class loader

3) Application/System class loader

Bootstrap class Loader :-

It is responsible to load the required .class file from java API that means all the predfined classes .class file will be loaded by Bootstrap class loader.
It the super class of Extension class loader as well as It has the highest priority among all the class loader.

Extension/Platform class Loader :-

It is responsible to load the required .class files from ext (extension) folder. Inside the extension folder we have jar file(Java level zip file) given by some third party or user defined jar file.
It is the super class of Application class loader as well as It has more priority than Application class loader.

Application class Loader :-

It is responsible to load the required .class file from class path level i.e Environment variable. It has lowest priority as well as It is the sub class of Extension class loader.

How Delegation Hierarchy algorithm works :-

Whenever JVM makes a request to class loader sub system to load the required .class file into JVM memory, first of all class loader sub system makes a request to Application class loader , Application class loader will delegate the request to the Extension class loader, Extension class loader will also delegate the request to Bootstrap class loader.
Bootstrap class loader will load the .class file from lib folder(rt.jar) and then by pass the request to extension class loader, Extension class loader will load the .class file from ext folder(*.jar) and by pass the request to Application class loader, It will load the .class file from environment variable into JVM memory.

Note :-
If all the class loaders are unable to load the .class file into JVM memory then we will get a Runtime exception i.e java.lang.ClassNotFoundException

Linking :-
----
verify :-

It ensures the correctness the .class files, If any suspicious activity is there in the .class file then It will stop the execution immediately by throwing an exception i.e java.lang.VerifyError.

There is something called ByteCodeVerifier, responsible to verify the .class file i.e byte code. Due to this verify module JAVA is highly secure language.

Prepare :-

It will allocate the memory for all the static data member, here all the static data member will get the default values so if we have static int x = 100; then for x we will get the default value i.e 0.

Resolve :-

All the symbolic references will be converted to direct references.

Initialization :-

In Initialization, all the static data member will get their actual value as well as if any static block is available in the class then the static block will be exceuted here.

static block :-

It is a very special block in java which will be executed at the time of loading the .class file into JVM memory by class loader subsystem.

The main purpose of static block to initialize the static data member of the class.

static block will be executed only once because class loading is possible only once in java.

If we have multiple static blocks in a class then It will be executed according to order.

What is the difference between java.lang.ClassNotFoundException and
----
java.lang.NoClassDefFoundError:-
----

java.lang.ClassNotFoundException :-

It occurs when we try to load the required .class file at runtime by using Class.forName() statement and if the required .class file is not available at runtime then we will get an exception i.e java.lang.ClassNotFoundException.

java.lang.NoClassDefFoundError:-

It occurs when the class was present at the time of compilation but at runtime the required .class file is not available(mannually deleted by user) then we will get an exception i.e java.lang.NoClassDefFoundError.

What is the drawback of new keyword :-

We know 'new' keyword is used to create the objcet but It demands the class name at the begning or at the time of compilation.

new keyword is not suitable to create the object for the classes which are coming at runtime.

In order to create the Object for the classes which are coming at runtime either from database or files, we should use newInstance() method available in class called Class.

2) Runtime data areas :-

Method area :-

In this area all class level information is available. Actually the .class file is dumpped here hence we have all kinds of information related to class is available like name of the class, name of the immediate super class, method and variable name, static variable and so on.

There is only one method area per JVM.

Heap memory :-

It stores information regarding object and instance variables. All the objects are created as a part of HEAP memory so automatically all the instance variables are also the part of HEAP memory.

The is only one HEAP area per JVM.

Stack area :-

For every thread, JVM creates a seperate runtime stack. Each stack is created as a part of stack memory. All the local variables are also the part of stack memory.
Each entry in the stack is called Stack Frame, Each stack frame containd three parts
1) Local variable Array
2) Frame Data
3) Operand Stack

Garbage collector :-

It is an automatic memory management in java. JVM internally contains a component called Garbage collector, It is responsible to delete the unused objects or the objects which are not containing any references in the memory.

PC Register :-

In order to hold the current executing instruction of a thread we use PC register (Program Counter Register). For a single JVM we can have multiple PC Registers.

Native method stack :-

For every thread in java a seperate native stack is created. It stores the native method information.

Interpreter :-

JVM stands for Java Virtual Machine. It is a software in the form of Interpreter written in 'C' language.

The main purpose of JVM to load and execute the .class file.JVM has a component called class loader subsystem responsible to load the required .class file as well as It allocates the necessary memory needed by the java program.

JVM executes our program line by line. It is slow in nature so java software people has provided a special compiler i.e JIT compiler to boost up the execution.

JIT compiler :-

It stands for just in time compiler. It is the part of JVM which increases the speed of execution of a java program.

It holds the frequently used instruction and make it available at the time of executing java program so the execution will become faster.

A static method does not act upon instance variable why ?

We can't use instance variable inside the static method without creating an

object because when we execute a java program the sequence is as follows :

a) The class loader subsystem loads the .class file into JVM memory.

b) JVM executes the static block, if any static block is available in the class

c) JVM starts searching the main() to execute the java program.

d) If user has created any object then object creation and initialization of all instance variables will be started from here.

So, from the above points it is clear that at the time of executing the static method instance variables are not available, that is the reason we can't use instance variables inside a static method without creating an object.

How to invoke a static method :

If a static method is available in the same class then we can directly call the static method but if the static method is available in another class then we can call the static method with the help of class name.

Declaring a static variable :-

If the value of variable is different with each object then we should declare instance variable but if the value is common for all the object then we should declare static variable.

ABSTRACT CLASS AND ABSTRACT METHOD

An abstract method is a common method which is used to provide easiness to the programmer because the programmer faces complexcity to remember the method name.

An abstract method observation is very simple because every abstract method contains abstract keyword, abstract method does not contain any body and at the end there must be a terminator i.e ; (semicolon)

In java whenever action is common but implementations are different then we should use abstract method, Generally we declare abstract method in the super class and its implementation must be provided in the sub class.

if a class contains atleast one method as an abstract method then we should compulsory declare that class as an abstract class.

Once a class is declared as an abstract class we can't create an object for that class.

All the abstract methods declared in the super class must be overridden in the child class otherwise the child class will become as an abstract class hence object can't be created for the child class as well.

Note :- If we have a constructor inside the abstract class then that constructor will be executed.

According to the defination, An abstract class may or may not have an abstract method but an abstract method must have an abstract class.

```
abstract class  Hello
{
      public void show()
      {
      }
}
```

interface :-

An interface is a keyword in java which is similar to a class.

Upto JDK 1.7 an interfcae contains only abstract method that means there is a gurantee that inside an interface we don't have concrete methods.

In order to implement the member of an interface, java software people has provided implements keyword.

All the methods declared inside an interface is bydefault public and abstract so at the time of overriding we should apply public access modifier to sub class method.

All the variables declared inside an interface is bydefault public, static and final.

We should override all the methods of interface to the sub class otherwise the sub class will become as an abstract class hence object can't be created.

We can't create an object for interface, but reference can be created.

By using interfcae we can acheive multiple inheritance in java.

```
interface Moveable
{
      int SPEED = 60;    //public + static + final

      void move();       //What to do ?
}
class Car implements Moveable    //How to do?
{
      @Override
      public void move()
      {
            //SPEED = 80 ;
            System.out.println("Speed of the car is :"+SPEED);
      }
}
class  MoveableDemo
{
      public static void main(String[] args)
      {
            Moveable m = new Car();
            m.move();
            System.out.println("Car speed is :"+Moveable.SPEED);
      }
}
```


```
interface Bank
{
       void deposit(int d);
       void withdraw(int w);
}
class Customer implements Bank
{
     int balance= 1000;
```

```
      @Override
      public void deposit(int d)
      {
            if(d<=0)
            {
                  System.out.println("Amount can't be deposited....");
            }
            else
            {
                  balance = balance + d;
                  System.out.println("Amount after deposit :"+balance);
            }
      }
      @Override
      public void withdraw(int w)
      {
            balance = balance -w;
            System.out.println("Amount after withdraw :"+balance);
      }
}
public class BankDemo
{
      public static void main(String[] args)
      {
            Bank b1 = new Customer();
            b1.deposit(10000);
            b1.withdraw(9000);
      }
}
--------------------------------------------------------------------------------
------

interface A
{
      void sum();
}
interface B
{
      void sum();
}
class C implements A,B
{
      int a = 100;
      int b = 200;
      @Override
      public void sum()
      {
            int sum = a + b;
            System.out.println("The sum of the number is :"+sum);
      }
}
public class MultipleInheritance
{
      public static void main(String[] args)
      {
            C c1 = new C();
            c1.sum();
      }
}
--------------------------------------------------------------------------------
------
interface A1
{
  void m1();
```

```
}

interface B1
{
  void m2();
}

interface C1 extends A1,B1
{
   void m3();
}
class ImplementInterface implements C1
{
      @Override
      public void m1()
      {
            System.out.println("It is M1 method");
      }
      @Override
      public void m2()
      {
            System.out.println("It is M2 method");
      }
      @Override
      public void m3()
      {
            System.out.println("It is M3 method");
      }
}
public class ExtendingInterfcae
{
      public static void main(String[] args)
    {
            ImplementInterface i = new ImplementInterface();
            i.m1();
            i.m2();
            i.m3();
      }
}
--------------------------------------------------------------------------------
------

package com.ravi.veh;

public interface Vehicle
{
   void run();
   void horn();

  default void digitalMeter()  //specific purpose (We can override this method)
  {
        System.out.println("Digital meter");
  }

  static void safe()  //Common purpose (for providing common method)
  {
        System.out.println("Running safely");
  }
}


package com.ravi.veh;

public class Bike implements Vehicle
```

```
{
      @Override
      public void run()
      {
            System.out.println("Running Bike....");
      }

      @Override
      public void horn()
      {
            System.out.println("Peep-peep");

      }

}



package com.ravi.veh;

public class Car implements Vehicle
{
      @Override
      public void run()
      {
            System.out.println("Running car.....");
      }

      @Override
      public void horn()
      {
            System.out.println("Pop-Pop");
      }

        public void digitalMeter()
        {
              System.out.println("Car has digital meter facility...");
        }
}



package com.ravi.veh;

public class Main
{
      public static void main(String[] args)
      {
            Vehicle v = new Car();
            v.digitalMeter();
            v.run();
            v.horn();

            Vehicle v1 = new Bike();
            v1.digitalMeter();
            v1.run();
            v1.horn();
            Vehicle.safe();
      }

}
```

---
interface (JDK 1.8 onwards) :

From JDK 1.8 onwards java compiler allowed to write static and default method inside an interface.

default method we can write inside an interface so we can provide specific implementation for the classes which are implmenting from interface why because we can override default method inside the sub classes.

```
//default method for specific class method implementation
interface HotDrink
{
      void prepare();

      default void expressPrepare()     //possible from jdk 1.8
      {
        System.out.println("Preparing with premium");
      }
}
class Tea implements HotDrink
{
      @Override
      public void prepare()
      {
            System.out.println("Preparing Tea");
      }
      @Override
      public void expressPrepare()
      {
        System.out.println("Preparing premium Tea");
      }
}
class Coffee implements HotDrink
{
 @Override
  public void prepare()
      {
            System.out.println("Preparing Coffee");
      }
}
class DefaultMethod
{
      public static void main(String[] args)
      {
            HotDrink hk1 = new Tea();
            HotDrink hk2 = new Coffee();
            hk1.prepare();
            hk1.expressPrepare();
            hk2.prepare();
      }
}
```

From JDK 1.8 onwards we can define a static method inside an interface, static method is used to provide common implementation for all the classes which are implementing the interface. Here we can provide some common message because we can't override static method.

static methods are bydefault not available to implementer classes we can use static methods with the help of interface only.

```
//static method implemntation common for all
interface HotDrink
{
      void prepare();

      static void expressPrepare()
      {
            System.out.println("Prepare with no sugar...");
      }
}
class Tea implements HotDrink
{
      public void prepare()
      {
            System.out.println("Preparing Tea");
      }
}
class Coffee implements HotDrink
{
  public void prepare()
      {
            System.out.println("Preparing Coffee");
      }
}
class StaticMethod
{
      public static void main(String[] args)
      {
            HotDrink hk1 = new Tea();
            HotDrink hk2 = new Coffee();
            HotDrink.expressPrepare();
            hk1.prepare();
        hk2.prepare();
      }
}
```


Anonymous class :-

As we know to implement the member of an interface we need a class but with the introduction of anonymous class concept we can override the abstract method of an interface inside the body of interface itself as shown in the program below.


```
//Anonymous class
interface Vehicle
{
     void run();
}
public class Anonymous
{
      public static void main(String [] args)
      {
           Vehicle v = new Vehicle()
                  {
                        @Override
                        public void run()
                              {
                                    System.out.println("Running Safely");
                          }
                  };
                  v.run();
      }
}
```


@FunctionalInterface :-

If an interface contains only one abstract method then we can represent that interface by using FunctionalInterface annotation.

```
@FunctionalInterface
interface Vehicle
{
      void run();
}
```


```
@FunctionalInterface
interface Vehicle
{
     void run();
}
public class Anonymous1
{
      public static void main(String [] args)
      {
               Vehicle car = new Vehicle()
                  {
                        @Override
                        public void run()
                              {
                                    System.out.println("Running Car");
                          }
                  };
                  car.run();

                  Vehicle bike = new Vehicle()
                  {
                        @Override
                        public void run()
                              {
                                    System.out.println("Running Bike");
                          }
                  };
                  bike.run();
      }
}
```


There is a predefined interface avaialble in java.lang package called Runnable interface, this Runnable interface contains only one abstract method i.e run(). Hence It is a Functional Interface.


```
class Anonymous2
{
  public static void main(String [] args)
   {
        Runnable r = new Runnable()
         {
         @Override
            public void run()
               {
                  System.out.println("Running....");
            }
        };
        r.run();
   }
}
--------------------------------------------------------------------------------
----
```

Lambda Expression :-

It is an anonymous function.

It can be used with functional interface only.

It is used to concise our code.

Rules of Lambda :
----
1) If the body of the lambda expression contains one statement then curly braces are optional.

2) Java compiler automatically recognize the variable type so even data type is not required.

```
--------------------------------------------------------------------------------
---
@FunctionalInterface
interface Moveable
{
      public void move();
}
class Lambda1
{
      public static void main(String[] args)
      {
            Moveable m = ()->
            {
            System.out.println("Hello Welcome to Lambda Expression");
            };
            m.move();
      }
}
--------------------------------------------------------------------------------
---

--------------------------------------------------------------------------------
---
@FunctionalInterface
interface Calculate
{
       void add(int a, int b);
}
class Lambda2
{
      public static void main(String[] args)
      {
            Calculate c = (a,b)->
            {
                  System.out.println("Sum is :"+(a+b));
            };
            c.add(12,12);
      }
}
--------------------------------------------------------------------------------
```


```
@FunctionalInterface
interface Length
{
       int getLength(String str);
}
class Lambda3
{
      public static void main(String[] args)
      {
            Length l = (str)->
            {
                  return str.length();
            };
            System.out.print("Length is :"+l.getLength("India"));
      }
}
```



```
@FunctionalInterface
interface Length
{
       int getLength(String str);
}

class Lambda4
{
      public static void main(String[] args)
      {
            Length l = (str)-> str.length();

            System.out.print("Length is :"+l.getLength("Ravi"));
      }
}
```


EXCEPTION HANDLING

Exception :-

An exception is an abnormal situation or an unexpected situation in a normal execution flow.

Due to an exception our execution of the program will be disturbed first and then terminated permanently.

An exception occurs due to dependency, when one part of the program is dependent to another part in order to complete the task then there might a chance of getting an exception.

EXCEPTION ALSO OCCURS DUE TO THE WRONG INPUT GIVEN BY THE USER.

Exception Hierarchy :-

As a developer we are responsible to handle the exception whereas errors are taken care by System admin.

Object Orientation has provided some mechanism to handle the exception which are as follows :-

1) try
2) catch
3) finally
4) throw
5) throws

Some exception criteria and their respective classes in Java :-
----

```
1) java.lang.ArithmeticException
   It occurs when we divide a number by zero.

   Eg :- int x = 10/0;

2) java.lang.ArrayIndexOutOfBoundsException
    It occurs when array is out of limit.

    Eg :-  int []x = {12,78,90};
             System.out.println(x[3]);

3) java.lang.NumberFormatException
    It occurs when the number is not in a proper format to convert.

    Eg:-     String x = "Ravi";
             int y = Integer.parseInt(x);

 4) java.lang.NullPointerException
     It occurs when we apply any method on null literal.

     Eg:-  String x = null;
             x.length();
```

Note :-
Exception is the super class of all the exceptions we have in java

```
class ExpTest
{
      public static void main(String[] args)
      {
            Exception e = new ArithmeticException();
            System.out.println(e);

            Exception e1 = new ArrayIndexOutOfBoundsException();
            System.out.println(e1);
      }
}
```


If we don't use exception handling mechanism then our program will halt in the middle with abnormal termination.

```
//Program to describe that if we don't use exception handling mechanism then our program will halt in the middle.
class Test
{
      public static void main(String[] args)
      {
           System.out.println("Main method started....");
```

```
         int x = 10;
             int y = 0;
             int z = x/y;   //HALT ( java.lang.ArithmeticException)

             System.out.println("The value of z is :"+z);
             System.out.println("Main method Ended....");
      }
}
--------------------------------------------------------------------------------
----
```

try :-

Whenever our statement is error suspecting statement then put that statement inside the try block.

The try block will trace the program line by line and if any exception occurs then It will create the exception object and throw the exception object to the nearest catch block.

try block must be followed either by catch block or finnaly block or both. In between the try-catch we can't write any kind of statement.

catch block :-

The main purpose of catch block to handle the exception which is thrown by try block.

The catch block will only execute if there is an exception inside the try block.


```
class FirstExp
{
      public static void main(String[] args)
      {
            System.out.println("Main method Started");
            try
          {
                  int a=10;
                  int b=0;
                  int c=a/b;
                  System.out.println("The c value is :"+c);
          }
          catch (Exception e)
          {
            System.out.println(e);
            }
            System.out.println("Main method Completed");
      }
}
--------------------------------------------------------------------------------
----
```

Note :- After getting an exception inside the try block the remaining code inside the try block will not be executed.

As we know an Exception is the super class for all the exceptions we have in java but according to our requirement we can also provide specific Exception.

```
public class SpecificException
{
      public static void main(String[] args)
      {
            try
            {
                  int a [] = {12,23,34,45};
                  System.out.println(a[4]); //new
ArrayIndexOutOfBoundsException();
                System.out.println("Inside try");
            }
            catch(ArrayIndexOutOfBoundsException e)
            {
              System.err.println(e);
              System.out.println("-------------------");
              e.printStackTrace();//For complete details regarding the exception
              System.out.println("-------------------");
              System.out.println(e.getMessage());
            }
            System.out.println("Main is completed!!!");
      }
}
```


Multiple try catch blocks :

According to our requirement we can take multiple try-catch. Each try block will contain catch block to handle the exception.


```
public class MultipleTryCatch
{
      public static void main(String[] args)
      {
            try
            {
               int a=100,b=0,c;
               c = a/b;
               System.out.println("c value is :"+c);
            }
            catch(ArithmeticException e)
            {
              System.err.println("Divide by zero problem....");
            }
            try
            {
                  int x[] = {12,90};
                  System.out.println(x[2]);
            }
            catch(ArrayIndexOutOfBoundsException e)
            {
               System.err.println("Array is out of limit...");
            }
            System.out.println("Main completed..");
      }
}
```


```
//Exception handling describes how to provide user-friendly messages to our
client
public class CustomerDemo
{
      public static void main(String[] args)
      {
            try
            {
            int a = 10;
            int b = 0;
            int c = a/b;
            System.out.println("c value is :"+c);
            }
            catch(Exception e)
            {
                  System.out.println("Please Don't put zero");
            }
            System.out.println("Hello user your program is completed!!!");
      }
}
--------------------------------------------------------------------------------
-----
```

Nested try :-

If a try block is placed inside another try block then it is called Nested try block. In nested try-block the inner try block will only execute if there is no exception inside the outer try block.

Eg:-

```
try              //outer try block
{
     statements;
     try            //inner try block
     {
     }
     catch(Exception e)
     {
     }
}
catch(Exception e)
{
}
--------------------------------------------------------------------------------
-----
public class NestedTry
{
      public static void main(String[] args)
      {
            try//Outer try
            {
                  String x = null;
                  System.out.println("The length of "+ x+ " is :"+x.length());

                  try //inner try
                  {
                      String y = "Ravi";
                      int z = Integer.parseInt(y);
                      System.out.println("z value is :"+z);
                  }
                  catch(NumberFormatException e)
                  {
                    System.err.println("Number is not in a format");
                  }
            }
            catch(NullPointerException e)
            {
                  System.err.println("Null Pointer problem...");
            }
        System.out.println("Main is completed....");
      }
}
--------------------------------------------------------------------------------
----
```

multiple catch block with single try :-

We can write multiple catch block with a single try block, multiple catch block is taken for providing more clearity regarding exception.

While working with multiple catch blcoks the super class catch block must be the last catch block otherwise we will get a compilation error.

```
try
{
}
catch(ArithemticException e)
{
}
catch(ArrrayIndexOutOfBoundsException e)
{
}
catch(Exception)
{
}
```

Note :- super classs must ne the last catch block.


```
public class MultyCatch
{
	public static void main(String[] args)
	{
		System.out.println("Main Started...");
		try
		{
			int a=10,b=0,c;
			c=a/b;
			System.out.println("c value is :"+c);

			int x[] = {12,78,56};
			System.out.println(x[4]);
		}
		catch(ArrayIndexOutOfBoundsException e1)
		{
			System.err.println("Array is out of limit...");
		}
		catch(ArithmeticException e2)
		{
			System.err.println("Divide By zero problem...");
		}
		catch(Exception e)
		{
			System.err.println("General ");
		}
		System.out.println("Main Ended...");
	}
}
```

-----
finally block in java :-

The finally used to handle the resources. According to software engineering the resources are memory creation, buffer creation, Opening of a databade, opening of a file and so on, So we need to handle them carefully.

finally is a block which is guranted for execution whether an exception has been thrown or not.

We should write all the closing statements inside the finally block so finally block will execute and close all the resources.

```
--------------------------------------------------------------------------------
-----
public class FinallyBlock
{
      public static void main(String[] args)
      {
            try
            {
                  System.out.println("main method started");
                  int a = 10;
                  int b = 0;
                  int c = a/b;  // new ArithmeticException();  HALT
                  System.out.println("c value is :"+c);
            }
            finally
            {
                  System.out.println("Finally block will be executed....");
            }
         System.out.println("Program is halt so It will not be executed");
      }
}
--------------------------------------------------------------------------------
----
```

Note :- If we write try with finally only the resources will be handled but not the exception whereas if we write try-catch and finally then exception and resources both will be handled.

```
public class FinallyWithCatch
{
      public static void main(String[] args)
      {
            try
            {
                  System.out.println("Main is started!!!");
                  int a[] = {12,67,56};
                  System.out.println(a[3]);
            }
            catch(ArrayIndexOutOfBoundsException e)
            {
                  System.err.println("Array is out of limit!!!");
            }
            finally
            {
               System.out.println("Resources will be handled here!!");
            }
            System.out.println("Main method ended!!!");
      }
}
--------------------------------------------------------------------------------
-----
```

*What is the difference between Checked Exception and Unchecked Exception?

Checked Exception :-

The exceptions which are very common in Java are called Checked Exception. Compiler takes very much care regarding these exceptions.

Here Compiler wants some clearity that by using this code you may face some problem at runtime and you didn't report me that how would you handle this code at runtime so provide either try-catch or declare the method as throws.

Unchecked Exception :-

The exceptions which are rarely occurred in java and for these kinds of exception compiler does not take very much care are called unchecked exception.

Unchecked exceptions are directly entertain by JVM because they are rarely occurred in java.

When to provide try-catch or declare the method as throws :-

We should provide try-catch if we want to handle the exception by own as well as if we want to provide user-defined messages to the client but on the other hand we should declare the method as throws if we are not interested to handle the exception and try to send it to the JVM for handling purpose.

Exception propogation :-

It is a mechanism where if any exception occur at a particular method and if the method does not have any exception handling mechanism then the method will verify the exception handling mechanism to the caller method like that exception will be propagated till main method.

If any of the caller method contains exception handling mechanism then exception will be handled otherwise exeception will be unhandled.

```
--------------------------------------------------------------------------------
----
class ExceptionPropogation
{
	public static void main(String[] args)
	{
		System.out.println("Main method started");
		m1();
		System.out.println("Main method ended");
	}
	public static void m1()
	{
		try
		{
			m2();
		}
		catch (Exception e)
		{
			System.out.println("Handled in m1");
		}
	}
	public static void m2()
	{
		m3();
	}
	public static void m3()
	{
		System.out.println(10/0);
	}
}
--------------------------------------------------------------------------------
----
```

Types of exception in java :-

In java we have two kinds of exception which are as follows :-

1) Predefined exception Or Built-in exception

2) Userdefined exception Or Custom exception

Predefined Exception :-

The exceptions which are defined by Java software people for their own use and specification are called Predefined Exception.
Eg:-

java.lang.ArithmeticException, java.lang.NumberFormatException and so on

Userdefined Exceptions :-

The exceptions which are defined by user according to our own requirement are called Userdefined exception. In java we can develop userdefined exception by defining the following rules

1) The User-defined Exception class must be the sub-class of Exception

Eg:-

```
class UserDefinedException extends Exception{}
```

2) The user defined class must contain default or parameterized constructor, if we want to pass some message to the super class then we must have parameterized constructor.

```
class UserDefinedException extends Exception
{
     UserDefinedException()
     {
     }
     UserDefinedException(String msg)
     {
     }
}
```

What is the difference between throw and throws :-

throw :-

As we know try block is responsible to create the exception object and throw the exception object to the catch block but in case of user defined exception a user is responsible to create the exception object and throw it to the catch block by using 'throw' keyword.

throws :-

Whenever a user is not interested to handle the exception and try to send it to JVM that means user doesn't want to handle the exception and try to skip from the situation, in that situation we should declare the method as throws.

```
class InvalidAgeException extends Exception
{
      public InvalidAgeException()
      {
      }

      InvalidAgeException(String msg)
      {
            super(msg);
      }
}
```

```
public class CustomException
{
      public static void validate(int age) throws InvalidAgeException
      {
            if(age<18)
                  throw new InvalidAgeException("Age is not valid");
            else
                  System.out.println("Welcome to voting!!!");
      }
      public static void main(String[] args)
      {
          try
          {
            validate(12);
          }
          catch(Exception e)
          {
            System.err.println("Exception Occured :"+e);
          }
      }
}
```

-

```
class GreaterMarksException extends Exception
{
      GreaterMarksException(){}

      GreaterMarksException(String message)
      {
            super(message);
      }
}
public class CustomException1
{
      public static void main(String[] args)
      {
            validateMarks(102);
      }
      static void validateMarks(int marks)
      {
            try
            {
           if(marks > 100)
               throw new GreaterMarksException("Marks is Invalid");
           else
               System.out.println("Your marks is :"+marks);
            }
            catch(Exception e)
            {
                  System.err.println(e);
            }
      }
}
```

-

THREAD

Write a program in java to proof that main a is thread ?

In java whenever we define main() method internally JVM creates a main thread.

Thread is a predefined class available in java.lang package and it contains a predefined static method currentThread() through which we can find out the currently executing thread at that particular place.

Eg:
```
          Thread t = Thread.currentThread();
```

-

```
public class MainDemo
{
   public static void main(String[] args)
   {
       Thread t = Thread.currentThread();
       String name = t.getName();
       System.out.println("Current thread is :"+name);
   }
}
```

How to create user-defined Thread in java ?

In java we can create user defined thread by using following two techniques.

1) By extending Thread class
2) By implementing Runnable interface

The following are the predefined interface and class in java.lang package
----
-
```
@FunctionalInterface
interface Runnable
{
    void run();
}

class Thread implments Runnable
{
    @Override
    public void run()
    {
    }
    start(){}
    isAlive(){}
    sleep(){}
    join(){}
    setPriority(int priority) {}
    get Priority(){}
    setName(String name)(){}
    getName(){}
    join()
}
```

Creating thread by extending Thread class :-

Ex :-

```
class MyThread extends Thread{}
```

As we know we have a predefined class called Thread available in java.lang package. We can create our own user-defined Thread by extending Thread class, we can also override run() to provide thread implementation.

In order to start a new Thread we can invoke start() method of Thread class. This method will perform the following two task.

1) It will request the Operating System to allocate a new Thread because user wants multithreading.

2) It will internally invoke the run()

```
--------------------------------------------------------------------------------
-
class MyThread  extends Thread
{
      @Override
      public void run()
      {
            System.out.println("Child thread is running...");
      }
}
public class UserThread {
      public static void main(String[] args)
      {
            System.out.println("Main thread is running..");
            MyThread mt1 = new MyThread();
            mt1.start();//New thread is created here
            System.out.println("Main thread ended");
      }
}
--------------------------------------------------------------------------------
-
```

public boolean isAlive() :-

It is a predefined method of Thread class through which we can find out whether a thread has started or not ?

As we know a thread starts after calling the start() so if we use isAlive() before start() method, it will return false but if the same isAlive() if we invoke after the start() method, it will return true.

We can't restart a thread in java if we try to restart then It will generate an exception i.e java.lang.IllegalThreadStateException

```
--------------------------------------------------------------------------------
class Foo extends Thread
{
      @Override
      public void run()
      {
            System.out.println("Child thread is running...");
            System.out.println("It is running with seperate stack");
      }
}
public class IsAlive
{
      public static void main(String[] args)
      {
            System.out.println("Main method is started..");
            Foo f = new Foo();
            System.out.println(f.isAlive()); //false

            f.start();//new Thread is created

            System.out.println(f.isAlive());//true
            System.out.println("Main method is ended..");

            f.start();  //Not possible, will generate exception
      }
}
--------------------------------------------------------------------------------
-
Anonymous class concept in Thread :-
------------------------------------------
public class AnonymousThread
{
public static void main(String args[])
{
      Thread t1 = new Thread()  //Anonymous class concept
       {
            @Override
          public void run()
            {
                System.out.println("My Thread One");
            }
      };

      Thread t2 = new Thread()
      {
            @Override
            public void run()
            {
                  System.out.println("My Thread Two");
            }
      };
            t1.start();
            t2.start();
   }
}
--------------------------------------------------------------------------------
-
```

Setting and getting the name of the thread :

The Thread class has provided two predefined methods setName(String name) and getName() to set and get the name of thread respectively.

If we don't set the name explicitly then bydefault the name of thread would be Thread-0, Thread-1, Thread-2 and so on.

```
--------------------------------------------------------------------------------
class Test extends Thread
{
      @Override
      public void run()
      {
            System.out.println(Thread.currentThread().getName()+" thread is
running...");     //Thread-0, Thread-1
      }
}
public class ThreadName
{
      public static void main(String[] args)
      {
            Test t1 = new Test();
            t1.start();
            Test t2 = new Test();
            t2.start();
            System.out.println(Thread.currentThread().getName()+" thread is
running...");
```

```
      }
}
```

From the above program it is clear that if we don't specify the thread name explicitly by using setName(String name) method then bydefault it would be Thread-0, Thread-1 and so on.

```
class Demo extends Thread
{
      @Override
      public void run()
      {
            System.out.println(Thread.currentThread().getName()+" thread is 
running...");
      }
}
public class ThreadName1
{
      public static void main(String[] args)
      {
           Demo d1 = new Demo();
           Demo d2 = new Demo();
           d1.setName("Child1");
           d2.setName("Child2");

           d1.start();
           d2.start();
           System.out.println(Thread.currentThread().getName()+" thread is 
running...");
      }
}
```

Thread.sleep(long ms) :-

The main purpose of sleep() method to put a Thread into temporarly waiting state, the waiting period of the Thread will depend upon the parameter we passed inside the sleep() method, It takes long ms as a parameter.

It is a static method so we can directly call sleep() method with the help of class name.

It throws a checked exception i.e InterruptedException so we should write sleep() inside try-catch or declare the method as throws.

Program on sleep()

```
class Sleep extends Thread
{
      @Override
      public void run()
      {
            for(int i=1;i<=10; i++)
            {
                  try
                  {
                        Thread.sleep(1000);
                  }
                  catch(InterruptedException e)
                  {
                        System.err.println("Thread has interrupted...");
                  }
                  System.out.println("i value is :"+i);
            }
      }
}
public class SleepDemo
{
      public static void main(String[] args)
      {
            System.out.println("Main Thread started..");
            Sleep s1 = new Sleep();
            s1.start();
      }
}
```

-

How to create a Thread by using Runnable interface approach :

While creating the user defined Thread by implementing Runnable interface we should pass the sub class object reference to Thread class constructor.

```
class Demo1 implements Runnable
{
   @Override
   public void run()
   {
     System.out.println("child thread");
   }
}
class RunnableDemo
{
  public static void main(String [] args)
  {
    Demo1 d1 = new Demo1();
    Thread t1 = new Thread(d1);
    t1.start();
   }
 }
```

 In between extends Thread and implements Runnable, which one is
 ----
 better and why ?

In between extends Thread and implements Runnable approach, implements Runnable is more better due to the following reason

1) When we use extends Thread, all the methods and properties of Thread class is available to sub class so it is heavy weight but this is not the case while implementing Runnable interface.

2) As we know Java does not support multiple inheritance using classes so in the extends Thread approach we can't extend another class but if we use implments Runnable interface still we have chance to extend another class and we can also implement one or more interfaces.

Anonymous class concept using Runnable interface :

```
class AnonymousRunnable
{
      public static void main(String [] args)
      {
            Runnable r1 = new Runnable()
            {
```

```
                        @Override
                        public void run()
                        {
                              System.out.println("Runnable Demo1...");

                        }

            };

            Runnable r2 = new Runnable()
            {
                        @Override
                        public void run()
                        {
                              System.out.println("Runnable Demo2...");

                        }

            };

            Thread t1 = new Thread(r1);
            Thread t2 = new Thread(r2);

            t1.start();
            t2.start();
      }
}
```

----
join() :-

The main purpose of join() method to put the parent thread into waiting state till the completion of child Thread.

It also throws checked exception i.e InterruptedException so better to use try catch or declare the method as throws.

It is an instance method so we can call this method with the help of Thread object reference.

```
class Join extends Thread
{
      public void run()
      {
            for(int i=1;i<=5;i++)
            {
                  try
                  {
                        Thread.sleep(1000);
                  }
                  catch(Exception e) {}
                  System.out.println(i);
            }
      }
}
public class JoinThread
{
      public static void main(String args[]) throws InterruptedException
      {
            System.out.println("Main started");
            Join t1=new Join();
            Join t2=new Join();
            Join t3=new Join();
            t1.start();
```

```
                t1.join();    //main Thread will halt here
            t2.start();
            t3.start();
            System.out.println("Main Ended");
      }
}
```

-

Problem with multithreading :-

Multithreading is very good to complete our task as soon as possible but in some situation, It provides some wrong data or wrong result.

In Data Race or Race condition, all the threads try to access the resource at the same time so the result will be corrupted.

-

```
class Reserve implements Runnable
{
      int available = 1;
      int wanted; //1

      Reserve(int i) //1
      {
            wanted = i;
      }
      @Override
      public   void run()
      {
       System.out.println("Available Berths ="+available);

            if(available >= wanted)
            {
                  String name=Thread.currentThread().getName();
                  System.out.println(wanted + " Berths reserve for "+name);
                  try
                  {
                  Thread.sleep(500);//ticket is printing
                  available = available-wanted;
                  }
                  catch (Exception e)
                  {
                  }
       System.out.println("Congratulations!!!"+name+" Your ticket is booked..");
            }
            else
            System.out.println("Sorry, no berths");
      }
}
public class Unsafe
{
      public static void main(String [] args)
      {
            Reserve r1 = new Reserve(1);
            Thread t1 = new Thread(r1);
            Thread t2 = new Thread(r1);
            t1.setName("Rahul");
            t2.setName("Rohit");
            t1.start();
            t2.start();
      }
}
```

-
We can't perform consucutive task using multithreading.

```
class MyThread implements Runnable
{
      String str;
      MyThread(String str)
      {
            this.str=str;
      }
      @Override
      public void run()
      {
            for(int i=1; i<=10; i++)
            {
                  System.out.println(str+ " : "+i);
                  try
                  {
                        Thread.sleep(100);
                  }
                  catch (Exception e)
                  {
                        e.printStackTrace();
                  }
            }
      }
}
public class Theatre
{
      public static void main(String [] args)
      {
      MyThread obj1 = new MyThread("Cut the Ticket");
      MyThread obj2 = new MyThread("Show the Seat");

            Thread t1 = new Thread(obj1);
            Thread t2 = new Thread(obj2);
            t1.start();
            t2.start();
      }
}
```

Synchronization :-

It is a technique through which we can control multiple threads but accepting only one thread at all the time.

In order to acheive synchronization we use synchronized keyword. Synchronization allows only one thread to enter inside the synchronized area.

Synchronization can be divided into two categories :-

1) Method level synchronization

2) Block level synchronization

Method level synchronization :-

In method level synchronization, the entire method gets synchronized so all the thread will wait at method level and only one thread will enter inside the synchronized area as shown in the diagram.

Block level synchronization :-

In block level synchronization the entire method does not get synchronized, only the part of the method gets synchronized so all the thread will enter inside the method but only one thread will enter inside the synchronized block as shown in the diagram below.

Note :- In between method level synchronization and block level synchronization, block level synchronization is more preferable because all the threads can enter inside the method so only one part of the method is synchronized so only one thread will enter inside the synchronized block.

How synchronization machanism controls multiple threads :-

Every Object has a lock(monitor) in java environment and this lock can be given to only one Thread at a time.

The thread who acquires the lock will enter inside the synchronized area, it will complete its task without any disturbance because at a time there will be only one thread inside the synchronized area. This is known as Thread-safety in java.

The thread which is inside the synchronized area, after completion its task while going back will release the lock so the other threads (which are waiting outside for the lock) will get a chance to enter inside the synchronized area by again taking the lock from the Object and submitting it to the synchronization mechanism.
----

program on method level synchronization :-

```
class Table
{
    public synchronized void printTable(int n)
    {
       for(int i=1; i<=10; i++)
       {
         try
         {
               Thread.sleep(100);
         }
         catch(Exception e) {}
         System.out.println(n*i);
       }
    }
}
class Thread1 extends Thread
{
      Table t ;
      Thread1(Table t)   // t = obj
      {
        this.t = t;
      }
      @Override
      public void run()
      {
            t.printTable(5);
      }
}
class Thread2 extends Thread
{
      Table t;
      Thread2(Table t)
      {
        this.t = t;
      }
      @Override
      public void run()
      {
            t.printTable(10);
      }
}
public class MethodSynchronization
{
      public static void main(String[] args)
      {
         Table obj = new Table();
         Thread1 t1 = new Thread1(obj);
         Thread2 t2 = new Thread2(obj);
         t1.start();  t2.start();
      }
}
--------------------------------------------------------------------------------
-

//Block level synchronization
class ThreadName
{
      public void printThreadName()
      {
            //This area is accessible by all the threads
            synchronized(this)
            {
                  String name = Thread.currentThread().getName();
                  for(int i=1; i<=9; i++)
                  {
                        System.out.println("i value is :"+i+" by :"+name);
                  }
            }
      }
}
class Thread3 extends Thread
{
      ThreadName tn;
      Thread3(ThreadName tn)
      {
            this.tn = tn;
      }
      @Override
      public void run()
      {
            tn.printThreadName();
      }
}
class Thread4 extends Thread
{
      ThreadName tn;
      Thread4(ThreadName tn)
      {
            this.tn = tn;
      }
      @Override
      public void run()
      {
            tn.printThreadName();
      }
}
public class BlockSynchronization
{
      public static void main(String[] args)
      {
            ThreadName tn = new ThreadName();
            Thread3 t3 = new Thread3(tn);
            Thread4 t4 = new Thread4(tn);

            t3.setName("Thread1"); t4.setName("Thread2");
            t3.start();  t4.start();
      }
}
```

-

Problem with Object level synchronization :-

From the above diagram it is clear that there is no interference between t1 and t2 thread because they are passing throgh Object1 whereas on the other hand there is no interferenec even in between t3 and t4 because they are also passing through Object2.

But there may be chance that with t1 Thread, t3 or t4 can enter inside the synchronized area at the same time, simillarly it is also possible that with t2 thread, t3 or t4 can enter inside the synchronized area so the conclusion is synchronization mechanism will not work with multiple Objects.

```
class Table1
{
    public synchronized void printTable(int n)
    {
       for(int i=1; i<=10; i++)
       {
         try
         {
               Thread.sleep(100);
         }
         catch(InterruptedException e)
         {
              System.err.println("Thread is Interrupted...");
         }
         System.out.println(n*i);
       }
    }
}
class Thread5 extends Thread
{
      Table1 t;
      Thread5(Table1 t)  //t = obj
      {
        this.t = t;
      }
      @Override
      public void run()
      {
            t.printTable(5);
      }
}
class Thread6 extends Thread
{
      Table1 t;
      Thread6(Table1 t)
      {
        this.t = t;
      }
      @Override
      public void run()
      {
            t.printTable(10);
      }
}
public class ProblemWithObjectSynchronization
{
      public static void main(String[] args)
      {
         Table1 obj1 = new Table1();      //object 1
         Thread5 t1 = new Thread5(obj1);
         Thread6 t2 = new Thread6(obj1);

         Table1 obj2 = new Table1();  //object 2
         Thread5 t3 = new Thread5(obj2);
         Thread6 t4 = new Thread6(obj2);

         t1.start();  t2.start(); t3.start(); t4.start();
      }
}
```

To avoid the above said problem we introduced static synchronization.

static synchronization :-

In static synchronization we will declare the synchronized method as static method so the lock will be on the class but not the object.

```
--------------------------------------------------------------------------------
class MyTable
{
public static synchronized void printTable(int n)  //static synchronization
          {
             for(int i=1; i<=10; i++)
             {
               try
               {
                     Thread.sleep(100);
               }
               catch(InterruptedException e)
               {
                    System.err.println("Thread is Interrupted...");
               }
               System.out.println(n*i);
             }
             System.out.println("------------------------");
          }
}
class MyThread1 extends Thread
{
      @Override
      public void run()
      {
            MyTable.printTable(5);
      }
}
class MyThread2 extends Thread
{
      @Override
      public void run()
      {
            MyTable.printTable(10);
      }
}
```

```
class MyThread3 extends Thread
{
      @Override
      public void run()
      {
            MyTable.printTable(15);
      }
}
public class StaticSynchronization
{
      public static void main(String[] args)
      {
            MyThread1 t1 = new MyThread1();
            MyThread2 t2 = new MyThread2();
            MyThread3 t3 = new MyThread3();

            t1.start();
            t2.start();
            t3.start();
      }
}
```

-

Thread priority :-

It is possible in java to assign priority to a Thread. Thread class has provided two predefined method setPriority(int priority) and getPriority() to set and get the priority of the thread respectively.

In java we can set the priority of the Thread in number from 1- 10 only where 1 is the minimum priority and 10 is the maximum priority.

Whenever we create a thread in java by default its priority would be 5 that is normal priority.

Thread class has also provided 3 final static variables which are as follows :-

Thread.MIN_PRIORITY :- 01

Thread.NORM_PRIORITY : 05

Thread.MAX_PRIORITY :- 10

Note :- We can't set the priority of the Thread beyond the limit so if we set the priority beyond the limit (1 to 10) then it will generate an exception java.lang.IllegalArgumentException

----

```
public class MainPriority
{
      public static void main(String[] args)
      {
            Thread t = Thread.currentThread();

            System.out.println("Main thread priority is :"+t.getPriority());

            Thread t1 = new Thread();
            System.out.println("User thread priority is :"+t1.getPriority());
      }
}
```

Note :- The user defined thread will get the same priority as per main thread priority.

```
public class MainPriority1
```

```
{
      public static void main(String[] args)
      {
            Thread t = Thread.currentThread();
            t.setPriority(Thread.MAX_PRIORITY);  //10

            //t.setPriority(11); -> java.lang.IllegalArgumentException

            System.out.println("Main thread priority is :"+t.getPriority());

            Thread t1 = new Thread();
            System.out.println("User thread priority is :"+t1.getPriority());
      }
}
```

The following program explains that we can assign some priority to thread so the thread will complete its task as soon as possible.

```
class ThreadPrior1 extends Thread
{
      @Override
      public void run()
      {
            int count=0;
            for(int i=1; i<=1000000; i++)
            {
              count++;
            }
            System.out.println("running thread name
is:"+Thread.currentThread().getName());

            System.out.println("running thread priority
is:"+Thread.currentThread().getPriority());
      }
      public static void main(String args[])
      {
            ThreadPrior1 m1 = new ThreadPrior1();
            ThreadPrior1 m2 = new ThreadPrior1();

            m1.setPriority(Thread.MIN_PRIORITY);//1
            m2.setPriority(Thread.MAX_PRIORITY);//10

            m1.setName("Last");
            m2.setName("First");

            m1.start();
            m2.start();
      }
}
```

Inter thread communication :-

It is a mechanism to communicate two synchronized threads within the context to acheive a particular task.

In ITC we put a thread into wait mode by using wait() method and other thread will complete its task, after completion it will call notify() method so the waiting thread will get a notification to complete its remaining task.

ITC can be implemented by the following method of Object class.

1) public final void wait() throws InterruptedException

2) public final void notify()

3) public final void notifyAll()

public final void wait() throws InterruptedException :-

It will put a thread into temporarly waiting state and it will release the lock.
It will wait till the another thread invokes notify() or notifyAll() for this object.

public final void notify() :-

It will wake up the single thread that is waiting on the same object.

public final void notifyAll() :-

It will wake up all the threads which are waiting on the object.

Note :- wait(), notify() and notifyAll() methods are defined in Object class but not in Thread class because methods are related to lock and Object has a lock so, all these methods are defined inside Object class.

Wap in java that describes the problem we will face if we dno't use inter thread communication :-

```
class Test implements Runnable
{
	int var=0;

	@Override
	public void run()
	{
		for(int i=2; i<=10; i++)
		{
			var = var + i;
			try
			{
				Thread.sleep(100);
			}
			catch (Exception e)
			{
			}
		}
	}
}
class ITCProblem
{
	public static void main(String[] args)
	{
		Test t = new Test();
		Thread t1 = new Thread(t) ;
		t1.start();
		try
		{
			Thread.sleep(100);
		}
		catch (Exception e)
		{
		}
		System.out.println(t.var);
	}
```

```
}
--------------------------------------------------------------------------------
-
class InterThreadComm
{
public static void main(String [] args)
{
SecondThread b = new SecondThread();
b.start();

            synchronized(b)
                  {
                        try
                        {
                              System.out.println("Waiting for b to
complete...");
                              b.wait();   // after releasing the lock, waiting
here
                        }
                        catch (InterruptedException e)
                        {

                        }
                        System.out.println("Value is: " + b.value);
                  }
      }
}
class SecondThread extends Thread
{
      int value = 0;
      @Override
            public void run()
            {
                  synchronized(this)
                  {
                        for(int i=1;i<=10;i++)
                        {
                              value =  value +i;
                        }
                        notify();  //will give notification to waiting thread
                  }
       }
}

--------------------------------------------------------------------------------
-
class Customer
{
int balance=10000;

      synchronized void withdraw(int amount)
      {
            System.out.println("going to withdraw...");
            if(balance<amount)
                  {
                        System.out.println("Less balance; waiting for
deposit...");
                              try
                              {
                                    wait();
                              }
                              catch(Exception e){}
                  }
            balance  = balance - amount;
            System.out.println("withdraw completed...");
      }

      synchronized void deposit(int amount)
            {
                  System.out.println("going to deposit...");
                  balance = balance + amount;
                  System.out.println("deposit completed... ");
                  notify();
            }
}
class InterThreadBalance
{
public static void main(String args[])
      {
      Customer c=new Customer();

            new Thread()  //anonymous class concept
            {
                  public void run()
                  {
                        c.withdraw(15000);
                  }
            }.start();

            new Thread()
            {
                  public void run()
                  {
                        c.deposit(10000);
                  }
            }.start();
    }
}
```

Input Output in java :

The Java software people has provided a package called java.io through which can perform input output operation in java.

In order to take the input from the user java.io package has provided the following predefined classes.

1) DataInputStream
2) BufferedReader

How to create the Object for the classes :-

```
1) DataInputStream d = new DataInputStream(System.in);


2) InputStreamReader i = new InputStreamReader(System.in);
    BufferedReader br = new BufferedReader(i);

                               OR

   BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

Note :- By using BufferReader, internally it creates a buffer so the overall performance will become more better.

Working with methods :
----

1) public int read() :-

It is used to read a single character from the keyboard. The return type of this method is int that is UNICODE values of the character.

2) public String readLine() :-

It is used to read multiple characters or a complete line then we should use readLine() method. The return type is String.

Write a program in java to read your name from the keyboard :-

```
import java.io.*;
class ReadName
{
      public static void main(String[] args) throws IOException
      {
            BufferedReader br = new BufferedReader(new 
InputStreamReader(System.in));

            System.out.print("Please enter your Name :");
            String name = br.readLine();

            System.out.println("Your name is :"+name);
      }
}
```

```
//WAP to read your age from the keyboard
import java.io.*;
class ReadInteger
{
      public static void main(String[] args) throws IOException
      {
            DataInputStream d = new DataInputStream(System.in);
            System.out.print("Enter your Age :");

            String ag = d.readLine();
            int age = Integer.parseInt(ag);
            System.out.println("Your Age is :"+age);
      }
}
```

```
//WAP in java to read a float value(salary) from the keyboard
import java.io.*;
class ReadSalary
{
      public static void main(String[] args) throws IOException
      {
            BufferedReader br = new BufferedReader(new 
InputStreamReader(System.in));

            System.out.print("Please enter your Salary :");
            String sal = br.readLine();

            float salary = Float.parseFloat(sal);
            System.out.print("Your salary is :"+salary);
      }
}
```

While reading the character from the keyboard using read() method, then the

read() method return type(int) must be converted to char

```
//WAP in java to read a character from the keyboard
import java.io.*;
class ReadCharacter
{
      public static void main(String[] args) throws IOException
      {

            BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));

            System.out.print("Please enter your Gender :");
             char gen =  (char)br.read();
             System.out.println("Your Gender is :"+gen);
      }
}
```

-
```
//WAP in java to read employee data
import java.io.*;
class EmpDataDemo
{
public static void main(String [] args) throws IOException
{
       BufferedReader br = new BufferedReader
      (new InputStreamReader(System.in));

      System.out.print("Enter id: ");
      int id = Integer.parseInt(br.readLine());

      System.out.print("Enter sex (M/F): ");
      //char gen = (char) br.read();

      char gen = br.readLine().charAt(0);

      System.out.print("Enter name: ");
      String name = br.readLine();

      System.out.println("Id = "+id);
      System.out.println("Sex = "+gen);
      System.out.println("Name = "+name);
      }
}
```

-

File Handling :-

In order to work with File system java software people has provided number of predefined classes like File, FileInputStream, FileOutputStream and so on. All these classes are available in java.io package. We can read and write the data in the form of Stream.

Stream :-

A Stream is nothing but flow of data or flow of characters to both the end. Stream is divided into two categories

1) byte oriented Stream :-

It used to handle characters, images, audio and video file.

2) character oriented Stream :-

It is used to handle the data in the form of characters or text.

Now Specifically Stream can be categorized as "input stream" and "output stream". input streams are used to read or receive the data where as output streams are used to write or send the data.

All Streams are represented by classes in java.io package. InputStream is the super class for all kind of input operation where as OutputStream is the super class for all kind of output Operation.

File :-

It is a predefined class in java.io package through which we can create file and directory. By using this class we can verify whether the file is existing or not.

```
File f = new File("abc.txt");
```

The above statement will not create a file, It actually create the file object and perform one of the following two task.
   a) If abc.txt does not exist, It will not create it
   b) if abc.txt does exist, the new file object will be refer to the referenec variable f

Now if the file does not exist and to create the file we should use createNewFile() method as shown below.

```
File f = new File("Hello.txt");
      f.createNewFile();
```

File class has also provided a method called exists() through which we can verify the corrosponding File is available or not.

```
import java.io.*;
public class File0
      {
            public static void main(String[] args)
            {
                  try
                        {
                              File f = new File("Hyd12.txt");

                              if(f.exists())
                              {
                                    System.out.println("File is existing");
                              }
                              else
                              {
                                    System.out.println("File is not existing");
                              }
                              if (f.createNewFile())
                                 {
                                     System.out.println("File created: " +
f.getName());
                        }
                                 else
                                    {
                             System.out.println("File is already existing....");
                         }
                  }
                         catch (IOException e)
```

```
                {
                            System.err.println(e);
                        }
            }
}
--------------------------------------------------------------------------------
-
FileOutputStream :-
--------------------
```

It is a predefined class available in java.io package. The main purpose of this class to create a new file and write the data to the file. Whenever we want to write the into the file using FileOutoutStream class then data must be available into the form of byte because It is byte-oriented class.

Note :- String class has provided a method called getBytes() through which we can read the data in byte format and the return type of this method is byte[].

Eg:-

```
            String x = "India is Great";
            byte b [] = x.getBytes();
--------------------------------------------------------------------------------
import java.io.*;
class File1
{
     public static void main(String args[]) throws IOException
     {
        FileOutputStream fout = new FileOutputStream("India.txt");
         String s = "India is a Great country";
             byte b[] = s.getBytes();
             fout.write(b);
         fout.close();
         System.out.println("Success....");
     }
}

--------------------------------------------------------------------------------
FileInputStream :-
-------------------
```

It is a predefined class available in java.io package. The main purpose of this class is to read the data or content from the specified file.

Eg:-

```
FileInputStream f = new FileInputStream("Hello.txt");
                        f.read();

--------------------------------------------------------------------------------
-
//Reading tha data from the file
import java.io.*;
class File2
{
     public static void main(String s[]) throws IOException
     {
             FileInputStream fin = null;   //here fin is a local variable
         try
         {
              fin = new FileInputStream("File1.java");
         int i;

         while((i = fin.read() ) != -1)
         {
              System.out.print((char)i);
         }
         }
         catch(FileNotFoundException e)
             {
                   System.err.println("File is not present");
             }
             catch (IOException e )
         {
                   System.err.println("IO exception..");
         }
             finally
             {
                     fin.close();
             }
     }
}
--------------------------------------------------------------------------------
-
//wap in java to read the data from one file and to write the data to another
file.
import java.io.*;
class File3
{
     public static void main(String s[]) throws IOException
     {
   FileInputStream fin = new FileInputStream("File2.java");
   FileOutputStream fout = new FileOutputStream("Hello.txt");
         int i;
         while((i = fin.read())!= -1)
         {
              System.out.print((char)i);
              fout.write((byte)i);
         }
         fin.close();
             fout.close();
     }
}
--------------------------------------------------------------------------------
-
```

Limitation of FileInputStream :-

By using FileInputStream class we can read the data from a single file only but if we want to read the data from two files at the same time then it is not possible by using FileInputStream, In this situation we should take a seperate Stream called SequenceInputStream.

SequenceInputStream :

It is a predefined class available in java.io package. This class is used to read the data from two files at the same time.

```
//Proram to read the data from two files at the same time
import java.io.*;
class File4
{
     public static void main(String args[]) throws IOException
     {
         FileInputStream f1 = new FileInputStream("File1.java");
         FileInputStream f2 = new FileInputStream("File2.java");
         SequenceInputStream s = new SequenceInputStream(f1,f2);
         int i;
         while(true)
             {
                    i = s.read();
                    System.out.print((char)i);
                    if(i==-1)
                          break;
              }
     }
}
--------------------------------------------------------------------------------
//Reading the data from two files and writing the data to a single file
import java.io.*;
class File5
{
      public static void main(String x[]) throws IOException
      {
               FileInputStream f1 = new FileInputStream("File3.java");
           FileInputStream f2 = new FileInputStream("File4.java");

           FileOutputStream fout = new FileOutputStream("Naresh.txt");

           SequenceInputStream s = new SequenceInputStream(f1,f2);
           int i;
           while((i = s.read()) != -1)
           {
               System.out.print((char)i);
               fout.write((byte)i);
           }
     }
}
--------------------------------------------------------------------------------
```

Limitation of FileOutputStream :-

It is used to write the data to a single file only. It is not suitable if we want to write the data more then one file. In order to write the data more than one file we should use a seperate Stream called ByteArrayOutputStream.

ByteArrayOutputStream :-

It is a predefined class available in java.io package. By using this class we can write the data to multiple files. ByteArrayOutputStream class provides a method called writeTo(), through which we can write the data to multiple files.

```
//Program to write the data on multiple files.
import java.io.*;
class File6
{
     public static void main(String args[]) throws IOException
     {
                  FileInputStream fin = new FileInputStream("File1.java");

                  FileOutputStream f1 = new FileOutputStream("one.txt");
                  FileOutputStream f2 = new FileOutputStream("two.txt");
                  FileOutputStream f3 = new FileOutputStream("three.txt");

                  ByteArrayOutputStream bout = new
                  ByteArrayOutputStream();
                  int i;
                  while((i = fin.read()) != -1)
            {
               bout.write((byte)i); //writing tha data to ByteArrayOutputStream
             }
           bout.writeTo(f1);
           bout.writeTo(f2);
               bout.writeTo(f3);
           bout.flush();  //clear the buffer for reuse ByteArrayOutputStream
           bout.close();
     }
}
--------------------------------------------------------------------------------
-
import java.io.*;
class File7
{
      public static void main(String[] args) throws IOException
      {
             FileInputStream fin = new FileInputStream("Sunset.jpg");
         FileOutputStream f1 = new FileOutputStream("E:\\a.jpg");
         FileOutputStream f2 = new FileOutputStream("E:\\b.jpg");

         ByteArrayOutputStream bout = new ByteArrayOutputStream();
         int i;
         while((i = fin.read()) != -1)
         {
               bout.write((byte)i);
         }
         bout.writeTo(f1);
         bout.writeTo(f2);
         System.out.println("success...");
             bout.flush();
         bout.close();
      }
}
--------------------------------------------------------------------------------
-
```

BufferedOutputStream :-

It is a predefined class available in java.io package.Whenever we use FileOutStream the data is on the Stream but not the buffer so there may be chance of miss memory management, It is always preferable that the data should be in the buffer.

By using this BufferedOutputStrean now the data is on the buffer so the execution will become faster.

```
//Program to put the data in the buffer for fast execution
import java.io.*;
class File8
{
      public static void main(String args[]) throws IOException
      {
           FileOutputStream fout = new FileOutputStream("abc.txt");
           BufferedOutputStream bout = new BufferedOutputStream(fout);

           String s = "Hyderabad is a nice city. It is in India";
           byte b[] = s.getBytes();
           bout.write(b);
               bout.close();
           System.out.print("success...");
      }
}
--------------------------------------------------------------------------------
-
```

BufferedInputStream :-

It is a predefined class available in java.io package. Whenever we use FileInputStrean to read the data from the file the data will be on the Stream but not the buffer so there may be a chance of miss memory management so we should take the data into the buffer by using BufferedInputStream class so overall the execution will become faster.

```
import java.io.*;
class File9
{
      public static void main(String args[]) throws IOException
      {
           FileInputStream fin = new FileInputStream("abc.txt");
           BufferedInputStream bin = new BufferedInputStream(fin);
           int i;
           while((i = bin.read()) != -1)
           {
                System.out.print((char)i);
           }
           bin.close();
      }
}
```
----

writing the primitive data to the files :-

It is possible to write the primitive data(byte,short,int, long, float, double, char and boolean) to the file. In order to write primitive data to the file we should use a predefined class available in java.io package called DataOutputStream. It provides various method like writeByte(), writeShort(), writeInt() and so on.

If we want to read the primitive data from the file we can use a predefined class available in java.io package called DataInputStream, this class provides various method like readByte(), readShort(),
readInt() and so on.

```
import java.io.*;
class File10
{
      public static void main(String args[]) throws IOException
      {
          FileOutputStream fout = new FileOutputStream("primitive.txt");
          DataOutputStream dout = new DataOutputStream(fout);

          dout.writeBoolean(true);
          dout.writeChar('A');
          dout.writeByte(Byte.MAX_VALUE);
          dout.writeShort(Short.MAX_VALUE);
          dout.writeInt(Integer.MAX_VALUE);
          dout.writeLong(Long.MIN_VALUE);
          dout.writeFloat(Float.MAX_VALUE);
          dout.writeDouble(Math.PI);
              dout.writeBytes("Hello India...");
          dout.flush();
          dout.close();
          FileInputStream fin = new FileInputStream("primitive.txt");
          DataInputStream din = new DataInputStream(fin);
          boolean f = din.readBoolean();
          char c = din.readChar();
          byte b = din.readByte();
          short s = din.readShort();
          int i = din.readInt();
          long l = din.readLong();
          float ft = din.readFloat();
          double d = din.readDouble();
               String x=  din.readLine();
          System.out.println(f +"\n"+c+"\n"+b+"\n"+s+"\n"+i+"\n"+l+"\n"+ft+"\
n"+d+"\n"+x);
          din.close();
      }
}
```

-

FileWriter class :

It is a predefined class available in java.io package, Using this class we can directly write String to the file.

Actually It is a character oriented Stream whereas if we work with FileOutputStream class, It is byte oriented Stream.

```
import java.io.*;
class File11
{
    public static void main(String args[]) throws IOException
    {
         FileWriter fw = new FileWriter("HelloIndia.txt");
         fw.write("Hello India...");
         fw.close();
         System.out.print("Success....");
    }
}
```

-

```
import java.io.*;
class File12
{
    public static void main(String args[]) throws IOException
    {
         FileWriter fw = new FileWriter("Data.txt");
         char c[ ] =  {'H','E','L','L','O', ' ',' ','W','O','R','L','D'};
         fw.write(c);
         fw.close();
         System.out.print("Success....");
    }
}
```

-

FileReader :-

It is a predefined class available in java.io package, It is a character oriented Stream. The main purpose of this class to read the data from the specified file.

```
import java.io.*;
class File13
{
    public static void main(String args[]) throws IOException
    {
         FileReader fr = new FileReader(args[0]);
         while(true)
         {
              int i = fr.read();
              if(i == -1)
                   break;
              System.out.print((char)i);
         }
         fr.close();
    }
}
```

-

```
Serialization :-
-----------------
It is a technique through which we can store the object in a file. Storing the
object into a file is called Serialization on the other hand Reading the object
from a file is called De-serialization.

In order to perform serialization, a class must implement serializable
interfcae, predefined interface in java.io package.

Java.io package has provided a predfined class called ObjectOutputStream to
perform serialization i.e writing Object to a file using writeObject().

whereas ObjectInputStream is also a predefined class available in java.io
package through which we can read the Object from a file using readObject(). The
return type of readObject() is Object.

--------------------------------------------------------------------------------
-
Employee.java
-----------------
import java.io.*;
import java.util.Date;
class Employee implements Serializable
{
	private int id;
	private String name;
	private float sal;
	private Date doj;

	Employee(int i, String n, float s, Date d)
	{
		id=i;
		name=n;
		sal=s;
		doj=d;
	}

	void display()
	{
		System.out.println(id+"\t"+name+"\t"+sal+"\t"+doj);
	}

	static Employee getData() throws IOException
	{
		BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));

		System.out.print("Enter emp id :");
		int id = Integer.parseInt(br.readLine());

		System.out.print("Enter Name :");
		String name=br.readLine();

		System.out.print("Enter Salary :");
		float sal=Float.parseFloat(br.readLine());

		Date d = new Date();
		Employee e = new Employee(id,name,sal,d);
		return e;
	}
}

StoreObj.java
---------------
```

```
//ObjectOutputStream is used to store objects to a file
import java.io.*;
import java.util.*;
class  StoreObj
{
      public static void main(String[] args) throws IOException
      {
            BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));

            FileOutputStream fos = new FileOutputStream("objfile.txt");
            ObjectOutputStream oos = new ObjectOutputStream(fos);

            System.out.print("How many objects?");
            int n =Integer.parseInt(br.readLine());     //3   i=0;

            for(int i=0; i<n; i++)                            //for(int i = 0;
i<3; i++)
            {
                  Employee e1=Employee.getData();
                  oos.writeObject(e1);
            }
            oos.close();
      }
}
```

```
GetObj.java
-------------
//ObjectInputStream is used to read objects from a file (de-serialization)
import java.io.*;
class GetObj
{
      public static void main(String[] args)  throws IOException
      {
            FileInputStream fis = new FileInputStream("objfile.txt");
            ObjectInputStream ois = new ObjectInputStream(fis);
            try
            {
                  Employee e;
                  while((e=(Employee)ois.readObject())!=null)
                  {
                        e.display();
                  }
            }
            catch (Exception ee)
            {
                  System.out.println("End of file reached :"+ee);
            }
            finally
            {
                  ois.close();
            }
      }
}
```

-

Note :- transient is a keyword in java and if we declare any variable as transient variable during serialization then transient variable will not be serialized and it will provide its dafault value like if the variable is int variable it will provide 0(zero).

Networking In Java :

```
-----------------------

IP Address :
-------------
An IP address is a unique identification number allocated to each and every
device connected through the network.

By using this IP address we can recognise a client in the network. IP address
contains some bytes which identify the network and the actual computer inside
the network.

Eg:- 192.168.100.09

Finding the IP Address of a server :
--------------------------------------
It is possible to find out the IP address of any website on internet. In order
to do the same java.net package has provided a predefined class called
InetAddress which contains a static method getByName(String host) through which
we can find out the IP address of any website.

Eg :- InetAddress.getByName("www.google.com");

Here getByName() will return the IP address of Google.com website.

The following are the imporatnt methods of InetAddress class :
-----------------------------------------------------------------------

1) public static InetAddress getByName(String host) throws
      UnknownHostException :-

    It will return the IP address of the specified host.

2) public static InetAddress getLocalHost() throws
    UnknownHostException :-

   It will return both the IP address and name of the local system.

3) public String getHostName() :- It will return the name of the System


4) public String getHostAddress() :-It will the return the IP Address of
    the System
--------------------------------------------------------------------------------
//Program to find out the local host IP address and Name.
import java.net.*;
class Inet
{
      public static void main(String args[])
      {
           try
           {
           InetAddress ia = InetAddress.getLocalHost();
           System.out.println("Local Name and IP Address  : "+ia);
           }
               catch(UnknownHostException e)
               {
                System.err.println(e);
               }
      }
}
--------------------------------------------------------------------------------
-
import java.net.*;
class Inet3
```

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

            InetAddress ia = InetAddress.getLocalHost();

             String addr = ia.getHostAddress();
             String name = ia.getHostName();
             System.out.println("My host name is :"+name+" My address is :
"+addr);
      }
}
--------------------------------------------------------------------------------
-
//program to find out the IP address of the given host
import java.net.*;
import java.io.*;
class IpAddress
{
public static void main(String[] args ) throws IOException
    {
            BufferedReader br = new BufferedReader
            (new InputStreamReader(System.in));

            System.out.print("Enter the host name: ");
                String s = br.readLine();
                try
                {
                   InetAddress ia = InetAddress.getByName(s);
                  System.out.println("IP address of "+s+" is : " + ia);
                }
            catch(UnknownHostException e )
            {
            System.err.println(e);
            }
    }
}
--------------------------------------------------------------------------------
```

URL class in Java :-

It is a predefined class available in java.net package. It stands for Uniform Resource Locator. The URL class represents the address that we specify at browser URL to access some resource.

Example of URL:-

https ://www.gmail.com:25/index.jsp

From this above URL

1) Protocol Name :- https

2) server name or host name :- www.gmail.com

3) port name :- 25 (-1 will be the value, if not supplied by the website)

4) File name :- index.jsp

```
-------------------------------------------------------------------------------
import java.net.*;
public class URLInfo
{
public static void main(String[] args)
{
      try
      {
      URL url=new URL("https://www.gmail.com:25/index.jsp");
      System.out.println("Protocol: "+url.getProtocol());
      System.out.println("Host Name: "+url.getHost());
      System.out.println("Port Number: "+url.getPort());
      System.out.println("File Name: "+url.getFile());
      }
      catch(Exception e)
      {
      System.out.println(e);
      }
 }
}
```

URLConnection class :

It is a predefined class availavle in java.net package. This class is useful to connect to a website or a resource in the network, it will fetch all the details of the specified web page as a part of URL class.

The URL class provides a method called openConnection(), this method will establish a connection with the specified web page and returns the URLConnection object, that is nothing but the complete details of the web page.

```
import java.io.*;
import java.net.*;
public class Details
{
public static void main(String[] args)
      {
            try
            {
            URL url=new URL("https://www.onlinesbi.com/");

            URLConnection urlcon=url.openConnection();

            InputStream stream=urlcon.getInputStream();
            int i;
            while((i=stream.read())!=-1)
                  {
                  System.out.print((char)i);

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

Socket :-

It is possible to eastblish a logical connection point between the server and the client so that communication can be done through this point is called Socket.

Now, each socket is given an identification number which is called port number. The range of port number is 0-65535.

Eastablishing a communication between the client and server using socket is called Socket Programming.

Socket programming can be connection oriented or connection less. In java Socket and ServerSocket are the predefined classes available in java.net package for connection oriented Socket Programming.

The client socket programming must have two information.
1) Ip address of the server
2) port number

The following are the importatnt methods of Socket class :

```
1) public InputStream getInputStream();

2) public OutputStream getOutputStream();

3) public synchronized void close();
```

The following are the importatnt methods of ServerSocket class :

```
1) public Socket accept() :- It is used to put the server in wait mode till
                                          a client accept or eastblish the
connection

1) public InputStream getInputStream();

2) public OutputStream getOutputStream();

3) public synchronized void close();
```

Creating a server that can send some data :

We can create a Socket that can be used to connect to a server and a client. Once the Socket is created, the Server can send the data to the client and client can receive the data.

All we have to do to just to send the data from server to socket. The socket will take care of whom to send the data on the network.

We should write the following java code to create a server that can send some data to the client.

1) At server side, create a server socket with some port number, This is done by using ServerSocket class

```
            ServerSocket ss = new ServerSocket(777);
```

2) We should make the server wait till a client accept connections. This can be done by using accept().

```
            Socket s = ss.accept();
```

3) Now Attach some output stream to ServerSocket using getOutputStream() method. This method returns OutputStream object. This method is used to send the data from Socket to the client

```
            OutputStream obj =  s.getOutputStream();
```

4) Take another Stream like PrintStream or DataOutputStream to send the data till the Socket.

```
                PrintStream ps = new PrintStream(obj);
```

5) Now to print the data which we are sending from Server to the client we can use print() or println() method available in PrintStream class.

```
            ps.println(String data);
```

6) Now close all the connections.

```
            ss.close();
            s.close();
            ps.close();
```

Creating a client that can receive some data :-

We can write client program that receives all the String sent from server to client machine.We should write the followinng java code

1) We should create a Socket at client side by using Socket class as

```
      Socket s = new Socket("Ip address", port number);
```

Note :- If the client and server both are running in a single machine then it is calles "localhost"

2) We should add InputStream to the Socket so that the Socket will be able to receive the data on the InputStream.

```
            InputStream obj = s.getInputStream();
```

3) Now to read the data from Socket to the client machine we can take the help of BufferedReader as

```
            BufferedReader br  = new BufferedReader(new InputStreamReader(obj));
```

4) Now We can read the data from the BufferedReader as

```
            String str = br.readLine();
```

5) close the connection

```
                  br.close();
                  s.close();
```

WAP in java where server can send some data and client can receive it.

```
//Program to send String to the client
import java.io.*;
import java.net.*;
class Server1
{
      public static void main(String args[]) throws IOException
      {
            ServerSocket ss = new ServerSocket(777);
            Socket s=ss.accept();

            System.out.println("Connection establish...");

            OutputStream obj=s.getOutputStream();
            PrintStream ps = new PrintStream(obj);

            String str="Hello Client";
            ps.println(str);
            ps.println("Bye-Bye");
            ps.close();
            ss.close();
            s.close();
      }
}


//Program to receive the String sent from Server
import java.io.*;
import java.net.*;
class Client1
{
      public static void main(String args[]) throws Exception
      {
      Socket s = new Socket("localhost",777);

        InputStream obj = s.getInputStream();

        BufferedReader br = new BufferedReader(new InputStreamReader(obj));

        String str;

        while((str=br.readLine()) !=null)
            {
              System.out.println("From Server :"+str);
            }
        br.close();
        s.close();
      }
}
--------------------------------------------------------------------------------
-----
```

WAP in java where we can perform chating between client and Server.

```
//A server that receives and send the data
import java.io.*;
import java.net.*;
class Server2
{
      public static void main(String [] args) throws Exception
      {
            ServerSocket ss = new ServerSocket(888);
            Socket s = ss.accept();
            System.out.println("Connection Established..");

            PrintStream ps = new PrintStream(s.getOutputStream());

            BufferedReader br = new BufferedReader
                  (new InputStreamReader( s.getInputStream()));

            BufferedReader kb = new BufferedReader(new
InputStreamReader(System.in));

            while(true)
            {
                  String str,str1;

                  while((str=br.readLine()) !=null)
                  {
                        System.out.println(str);
                        str1=kb.readLine();
                        ps.println(str1);
                  }
                  ps.close();
                  br.close();
                  kb.close();
                  ss.close();
                  s.close();
                  System.exit(0);//Shutdown the JVM
            }
      }
}

//A client that receives and send the data
import java.io.*;
import java.net.*;
class Client2
{
      public static void main(String [] args) throws Exception
      {
            Socket s = new Socket("localhost",888);

            DataOutputStream dos = new DataOutputStream(s.getOutputStream());

            BufferedReader br = new BufferedReader(new
InputStreamReader(s.getInputStream()));

            BufferedReader kb = new BufferedReader(new
InputStreamReader(System.in));

            String str,str1;
            while(! (str=kb.readLine()).equals("exit"))  //exit.equals("exit")
            {
                  dos.writeBytes(str+"\n");
                  str1=br.readLine();
                  System.out.println(str1);
            }
            dos.close();
            br.close();
            kb.close();
            s.close();
      }
}
```


WAP in java to downlaod the contant of the file, if the file is available at server.

```
//A server that sends a file content to the client
import java.io.*;
import java.net.*;
class FileServer
{
      public static void main(String [] args ) throws Exception
      {
            ServerSocket ss = new ServerSocket(8888);
            Socket s = ss.accept();

            System.out.println("Connection established...");

            BufferedReader in = new BufferedReader(new
InputStreamReader(s.getInputStream()));
```

```
            DataOutputStream out = new DataOutputStream(s.getOutputStream());

            String fname = in.readLine();     //fname = abc.txt
                  boolean flag;

            File f = new File(fname);
            if(f.exists())
                  flag=true;
            else
                 flag=false;

            if(flag==true)
                  out.writeBytes("yes"+"\n");
            else
                  out.writeBytes("No"+"\n");

            if(flag==true)
            {
                  FileReader fr=new FileReader(fname);
                  BufferedReader file = new BufferedReader(fr);

                  String str;
                  while((str=file.readLine()) !=null)
                  {
                        out.writeBytes(str+"\n");
                  }
                  file.close();
                  out.close();
                  in.close();
                  fr.close();
                  s.close();
                  ss.close();
            }
      }
}

//A Client receiving a file content
import java.io.*;
import java.net.*;
class FileClient
{
      public static void main(String [] args ) throws Exception
      {
            Socket s = new Socket("localhost",8888);

            BufferedReader kb = new BufferedReader(new
InputStreamReader(System.in));

            System.out.print("Enter a file name :");

            String fname = kb.readLine();  //fname = abc.txt

            DataOutputStream out = new DataOutputStream(s.getOutputStream());

            out.writeBytes(fname+"\n"); //first of all client is sending file
name

            BufferedReader in = new BufferedReader(new
InputStreamReader(s.getInputStream()));

            String str = in.readLine();

            if(str.equals("yes"))
            {
                  while((str=in.readLine()) !=null)
                        System.out.println(str);
                  kb.close();
                  out.close();
                  in.close();
                  s.close();
            }
            else
                  System.out.println("File not found..");
      }
}
```


Array in java :

An ordinary variable can hold only one value at a time.

Eg:- int x = 10;

But if we want to hold multiple values then we should go with array concept.

An array is a collection of homogeneous type of element. It can hold multiple values of same type.

In java array is a dynamic array where as in C and C++, array is a static memory.

*In java array is object so the array object always created and stored in the heap memory.

In order to find out the length of an array variable we can use length property or variable of an array.

Array stores the element in index wise where the index will always starts from 0.

In java we can create array by using two ways :-

1) By using array Literal

```
    int [] x = {12,23,34,45,56,67,78};
```

2) By using new Keyword

```
    int [] y = new int[5];
```

The major problem with array is that it can hold only homogeneous (same kind of value) but not hetrogeneous as well as array is fixed in size that means we can't increase and decrease the size of an array.

There is a predefined class available in java.util package called Arrays, which contains a static method sort() through which we can sort the array elements in ascending order.


```
import java.util.Arrays;
class Arr1
{
      public static void main(String[] args)
      {
            int a[]={12,45,78,90,87,9};

            Arrays.sort(a);

        System.out.println("The length of array is :"+a.length);

            System.out.println("Elements of an array using for loop...");

            for(int i=0;i<a.length ;i++)
            {
                  System.out.println(a[i]);
            }

         System.out.println("Elements of an array using for  each loop");

            for(int x : a)  //here a is an array variable where as x is an
ordinary var
            {
                  System.out.println(x);
            }
      }
}
--------------------------------------------------------------------------------
-----
//creation of an array object using new keyword
class Arr2
{
public static void main(String args[])
      {
            int x[]=new int[3];
            x[0]=11;
            x[1]=22;
            x[2]=33;

            for(int i=0;i<x.length;i++)
            System.out.println(x[i]);


            System.out.println(".............");

            for (int y : x)
                  System.out.println(y);
      }
}

--------------------------------------------------------------------------------
-----
//Program to calculate the student marks and find out the average
import java.io.*;
class Arr3
{
      public static void main(String [] args) throws IOException
      {
            BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));

            System.out.print("How Many Subjects: ");

            int n = Integer.parseInt(br.readLine());   //n = 3

            int [] marks=new  int[n];     //create array with size n

            for(int i=0; i<n; i++)
            {
                  System.out.print("Enter Marks :");
                  marks[i]=Integer.parseInt(br.readLine());
            }
            //find total marks
            int tot=0;

            for(int i=0; i<n;i++)
                  tot = tot+marks[i];

            System.out.println("The total marks is :"+tot);
            double avg = tot/n;
            System.out.println("The Avg is :"+avg);
      }
}
--------------------------------------------------------------------------------
-----
//Program to find the minimum value of an Array
import java.io.*;
class Test
{
      static void minValue(int arr[])   //18  9  23      min = 9
      {
            int min = arr[0];

            for(int i=1;i<arr.length;i++)
            {
                  if(min>arr[i])             //   9 > 23
                        min=arr[i];
        }
            System.out.println("The minimum value is :"+min);
      }
}
 class Arr4
 {
      public static void main(String args[]) throws IOException
            {
                  BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));

            System.out.print("How Many Values: ");

                  int n = Integer.parseInt(br.readLine());   //n = 3

                  int [] val =new  int[n];

                  for(int i=0; i<n; i++)
                  {
                  System.out.print("Enter Value of an array :");
                  val[i]=Integer.parseInt(br.readLine());
                  }

                  Test.minValue(val);
            }
}

--------------------------------------------------------------------------------
-----
//Modifying the value of an array
class Demo
{
      static int [] changeValue(int arr[])      //arr = {12,15,5,7,6}
      {
            arr[0] = 12;
            arr[1] = 15;
            return arr;
      }
}
 class Arr5
 {
      public static void main(String args[])
            {
                  int a[]={9,2,5,7,6};

            System.out.println("Before Change...");
                  for (int p : a)
                  {
                        System.out.println(p);
                  }

                  int b[] = Demo.changeValue(a);  //b = {12,15,5,7,6}

            System.out.println("After Change...");
                  for (int q : b)
                  {
                        System.out.println(q);
                  }
            }
      }
```

Wrapper class :

In java we have 8 primitive data types i.e byte, short, int, long, float, double, char and boolean.

Except these primitives, everything is object in java.

Wrapper class is a technique through which we can convert the primitives to corrosponding object.

Auto boxing

When we convert the primitive data types into object then it is called Auto boxing as shown below.

Primitive		Object
byte	-	Byte
short	-	Short
int	-	Integer
long	-	Long
float	-	Float
double	-	Double
char	-	Chracter
boolean	-	boolean

```
------class AutoBoxing1
{
      public static void main(String[] args)
      {
            Integer x = Integer.valueOf(12); //Upto 1.4 version
            System.out.println(x);

        int y = 15;
            Integer i = new Integer(y); //From JDK 1.5 onwards
            System.out.println(i);

      }
}
--------------------------------------------------------------------------------
----
class AutoBoxing2
      {
            public static void main(String args[])
                  {
                        Integer x = 15;//AutoBoxing
                        System.out.println(x);

                        var i = "Ravi";  //i is dynamically typed
                        System.out.println(i);


                  }
}
--------------------------------------------------------------------------------
-----
```

Auto unboxing :

Converting wrapper object to corrosponding primitive type is called auto unboxing.

Byte	-	byte
Short	-	short
Integer	-	int
Long	-	long
Float	-	float
Double	-	double
Chracter	-	char
Boolean	-	boolean

```
Integer i = new Integer(15);
int x = i.intValue();
--------------------------------------------------------------------------------
------
class AutoUnboxing1
{
   public static void main(String args[])
         {
            Integer obj = new Integer(25); //Upto 1.4
                  int x = obj.intValue();
                  System.out.println(x);
            }
}
--------------------------------------------------------------------------------
-----  class AutoUnboxing2
{
      public static void main(String[] args)
      {
                  Integer x= 15;
                  int y = x;
                  System.out.println(y);
      }
}
--------------------------------------------------------------------------------
------    Integer Buffer Test :
------------------------
class BufferTest
{
      public static void main(String[] args)
      {
            Integer x = 100;
            Integer y = 100;
            System.out.println(x==y); //true
      }
}
--------------------------------------------------------------------------------
---
class BufferTest1
{
      public static void main(String[] args)
      {
            Integer x = 128;
            Integer y = 128;
            System.out.println(x==y);  //false (upto 127 , true)

            Integer a = new Integer(10);
            Integer b = new Integer(10);
            System.out.println(a==b);  //false
      }
}
--------------------------------------------------------------------------------
---
```

Collection Framework

Collection framework is nothing but handling group of Objects. We know only object can move from one network to another network.

A collection framework is a class library to handle group of Objects.

It is implemented by using java.util package.

It provides an architecture to store an manipulate group of objects.

All the operation that we can perform on data such as searching, sorting, insertion and deletion can be done by using collection framework because It is the data structure of Java.

The simple meaning of collection is single unit of Objects.

It provides the following interfaces :

1) List (Accept duplicate elements)
2) Set (Not accepting duplicate elements)
3) Queue (Storing and Fetching the elements based on some order)

The following are the methods of Collection(I) interface

a) public boolean add(Object element) :- It is used to add an item/element in the collection.

b) public boolean addAll(Collection c) :- It is used to insert the specified collection elements in the existing collection

c) public boolean remove(Object element) :- It is used to delete an element from the collection.

d) public boolean removeAll(Collection c) :- It is used to delete all the elemnts from the existing collection.

List :

It is predefined interface, which is the sub interface of collection. It contains List of elements having duplicate values.

Behavior of List interface Collection classes :
----
1) Unlike array It accepts hetrogeneous type of data (different kinds of data)

2) Just like array List interface collection classes store the element on the basis of index.

3) It is dynamically growable in nature but we have some performance issue to copy paste the old data to the new memory.
      So if we know the size in advance it is better to go with Array but if we don't know the size then go with collection.

```
   //demo program
     import java.util.*;
class CollectionDemo
{
      public static void main(String[] args)
      {
            Vector v = new Vector();
            v.add(0,12);
            v.add("Naresh");
            v.add(2,23.78);
            v.add(true);
            v.add(new String("Ravi"));
            v.add(12);
            v.add("Naresh");
            v.add(23.78);
            v.add(true);
            v.add(new String("Ravi"));
            v.add(15);

            System.out.println(v);

            System.out.println("Initail capacity of Vector :"+v.capacity());
      }
}
```


 //As we know collection only takes hetrogeneous so even we are taking String will well get an error.

```
  import java.util.*;
class VectorDemo
{
      public static void main(String[] args)
      {
            Vector v  = new Vector();
            v.add("Rahul");
            v.add("Ranjan");
            v.add("Amit");
            v.add("Prakash");
            v.add("Rakshan");

            for(String x : v)  /error
            {
                  System.out.println(x);
            }
      }
}


--------------------------------------------------------------------------------
----import java.util.*;
class VectorDemo
{
      public static void main(String[] args)
      {
            Vector v  = new Vector();
            v.add("Rahul");
            v.add("Ranjan");
            v.add("Amit");
            v.add("Prakash");
            v.add("Rakshan");

            for(Object x : v) //Object is compulsory
            {
                  System.out.println(x);
            }
      }
}
--------------------------------------------------------------------------------
-----
```

How to iterate(Fetching) the elements from the Collection :

In order to fetch or iterate the elements from the collecetion we can use the following two interfaces :-

1) Iterator :

   It will fetch the elements or items in the forward direction only.It supports following methods

   a) public boolean hasNext() :- It will verify whether the data is available or not, if available it will return true otherwise it will return false.

   b) public Object next() :- It will return the Object from the collection.

2) ListIterator :

   It will fetch the elements in the forward direction as well as in the backward direction. It supports the following methods

   a) public boolean hasNext()
   b) public Object next()
   c) public boolean hasPrevious()
   d) public Object previous()

ArrayList :-

It is a predefined class available in java.util package under List interface.

It accepts duplicate elements and null values.

It is dynamically growable array.

It stores the elements on index basis so it is simillar to an array.

Initial capacity of ArrayList is 10.The capacity of Arraylist can be calculated by using the formula
new capacity = (current capacity * 3/2) + 1

All the methods declared inside an ArrayList is not synchronized so multiple thread can access the method of ArrayList.

It implements List,Serializable, Clonable, RandomAccess interfcaes

If we want to fetch the elements frequently then ArrayList is a good choice.

Note :

In java.util, Collection is a predefined interface but in the same package we have a predefined class called Collections which provides some static method like sort(), min(), max() and so on.


```
import java.util.*;
class ArrayListDemo
{
      public static void main(String... a)
      {
            ArrayList<String> arl = new ArrayList<String>();   //raw type

            arl.add("Apple");
            arl.add("Orange");
            arl.add("Grapes");
            arl.add("Mango");
            arl.add("Guava");
            arl.add("Mango");

            System.out.println("Contents :"+arl); //[Apple,Orange......]

            arl.remove(2);
            arl.remove("Guava");

            System.out.println("Contents After Removing :"+arl);
            System.out.println("Size of the ArrayList:"+arl.size());

            Collections.sort(arl);

            for(String x : arl)
            {
          System.out.println(x);
            }
      }
}
```


```
import java.util.*;
class Employee
  {
        int eno;
        String name;
        int age;

        Employee(int eno,String name,int age)
              {
                    this.eno=eno;
                    this.name=name;
                    this.age=age;
               }
  }

class ArrayListDemo1
{
  public static void main(String args[])
      {
        Employee e1=new Employee(101,"Pallavi",23);
        Employee e2=new Employee(102,"Ravi",21);
        Employee e3=new Employee(103,"Rahul",25);

        ArrayList<Employee> al=new ArrayList<Employee>();
        al.add(e1);
        al.add(e2);
        al.add(e3);

        Iterator itr=al.iterator();

        while(itr.hasNext())
        {
            Employee e = (Employee) itr.next();
         System.out.println(e.eno+":"+e.name+":"+e.age);
        }
   }
}
--------------------------------------------------------------------------------
-----
//Program to merge two collection
import java.util.*;
class ArrayListDemo2
      {
            public static void main(String args[])
            {
              ArrayList<String> al1=new ArrayList<String>();
              al1.add("Ravi");
              al1.add("Rahul");
              al1.add("Rohit");


              ArrayList<String> al2=new ArrayList<String>();
              al2.add("Pallavi");
              al2.add("Sweta");

              al1.addAll(al2);

              Iterator itr=al1.iterator();

              while(itr.hasNext())
              {
               System.out.println(itr.next());
              }
     }
}
--------------------------------------------------------------------------------
-----
//Program to fetch the elements in forward and backward direction using
ListIterator interface

import java.util.*;
class ArrayListDemo3
{
  public static void main(String args[])
      {
            ArrayList<String> al=new ArrayList<String>();
            al.add("Pallavi");
            al.add("Ravi");
            al.add("Rahul");
            al.add("Sachin");
            al.add("Aswin");
            al.add("Ananya");
            al.add("Bina");

            System.out.println("element at 2nd position: "+al.get(2));
            Collections.sort(al);
            Collections.reverse(al);

            ListIterator itr=al.listIterator();

            System.out.println("traversing elements in forward direction...");
            while(itr.hasNext())
             {
                  System.out.println(itr.next());
             }

            System.out.println("traversing elements in backward direction...");
            while(itr.hasPrevious())
             {
            System.out.println(itr.previous());
             }
      }
 }

--------------------------------------------------------------------------------
-----

////Serialization
import java.io.*;
import java.util.*;
 class ArrayListDemo4
       {
        public static void main(String [] args)
         {
           ArrayList<String> al=new ArrayList<String>();
           al.add("Nagpur");
           al.add("Vijaywada");
           al.add("Hyderabad");
               al.add("Jamshedpur");
          try
          {
                        //Serialization
                        FileOutputStream fos=new FileOutputStream("City.txt");
                        ObjectOutputStream oos=new ObjectOutputStream(fos);
                         oos.writeObject(al);
                         fos.close();
                         oos.close();
                        //Deserialization
                        FileInputStream fis=new FileInputStream("City.txt");
                        ObjectInputStream ois=new ObjectInputStream(fis);

                        ArrayList  list=(ArrayList)ois.readObject();
                        System.out.println(list);
          }
              catch(Exception e)
          {
              System.err.println(e);
            }
        }
    }
```


As we know whenever we create an object of ArrayList with empty constructor then by default its capacity would be 10.

But we we want to resize the capacity of ArrayList then we can use a method called ensureCapacity(int initialCapacity) as shown in the program below.

```
import java.util.ArrayList;

class ArrayListDemo5
{
      public static void main(String[] args)
      {
            ArrayList<String> city= new ArrayList<String>();//default capacity
is 10
            city.ensureCapacity(3);//resized the arraylist to store 3 elements.

            city.add("Hyderabad");
            city.add("Mumbai");
            city.add("Delhi");

            city.add("Kolkata");
            System.out.println("ArrayList: " + city);
       }
}
```


```
//Program on ArrayList that contains null values as well as we can pass the
element position basis

import java.util.*;
class ArrayListDemo6
{
      public static void main(String[] args)
      {
            ArrayList al = new ArrayList(); //raw type(unsafe operation)
            al.add(12);
            al.add("Ravi");
            al.add(12);
            al.add(0,"Hyderabad"); //add(int index, Object o)method of List
interface
            al.add(1,"Naresh");
            al.add(null);
            al.add(11);
            System.out.println(al);
      }
}
```


LinkedList :

It is a predefined class available in java.util package under List interface.

It is ordered by index position like ArrayList except the elements (nodes) are doubly linked to one another. This linkage gives us new method for adding and removing the elements from the middle of LinkedList.

*The important thing is, LikedList may iterate more slowely than ArrayList but

LinkedList is a good choice when we want to insert or delete the elements frequently in the list.

From jdk 1.5 onwards LinkedList has enhanced to support basic queue operation.

Constructor:

It has 2 constructors

1) LinkedList list1 = new LinkedList();
    It will create a LinkedList object with 0 capacity.

2) LinkedList list2 = new LinkedList(Collection c);
    Interconversion between the collection

Methods of LinkedList class:
----
1) void addFirst(Object o)
2) void addLast(Object o)

3) Object getFirst()
4) Object getLast()

5) Object removeFirst()
6) Object removeLast()

```
import java.util.*;
class LinkedListDemo
{
 public static void main(String args[])
 {
      List list=new LinkedList();
        list.add("Ravi");
        list.add("Vijay");
        list.add("Ravi");
        list.add(null);
        list.add(42);
        Iterator itr=list.iterator();
        while(itr.hasNext())
        {
         System.out.println(itr.next());
        }
  }
}
```

```
import java.util.*;
class LinkedListDemo1
{
      public static void main(String args[])
      {
           LinkedList<String> list= new LinkedList<String>();
           list.add("Item 2");//2
           list.add("Item 3");//3
           list.add("Item 4");//4
           list.add("Item 5");//5
           list.add("Item 6");//6
           list.add("Item 7");//7
           list.add("Item 9"); //10

           list.add(0,"Item 0");//0
           list.add(1,"Item 1"); //1
           list.add(8,"Item 8");//8
```

```
               list.add(9,"Item 10");//9
            System.out.println(list);

            list.remove("Item 5");
            System.out.println(list);

            list.removeLast();
            System.out.println(list);

                  list.removeFirst();
            System.out.println(list);

           list.set(0,"Ajay"); //set() will override the existing value
           list.set(1,"vijay");
           list.set(2,"Anand");
           list.set(3,"Aman");
           list.set(4,"Suresh");
           list.set(5,"Ganesh");
           list.set(6,"Harsh");
            System.out.println(list);

            for (String str : list)
            {
                        System.out.println(str);
            }
      }
}
--------------------------------------------------------------------------------
----
import java.util.LinkedList;
public class LinkedListDemo2
{
      public static void main(String[] argv)
      {
            LinkedList list = new LinkedList();
            list.addFirst("Ravi");
            list.add("Rahul");
            list.addLast("Anand");
            System.out.println(list.getFirst());
            System.out.println(list.getLast());
            list.removeFirst();
            list.removeLast();
            System.out.println(list); //[Rahul]
      }
}
--------------------------------------------------------------------------------
-----
```

ListIterator interface :

As we know ListIteraror is used to read the elements in forward direction as well as in backward direction
Some extra methods are added which are as follows :

```
            a) public boolean hasNext()
            b) public Object next()
            c) public boolean hasPrevious()
            d) public Object previous()
            e) public void remove()
            f) public void add(Object newElement)
            g) public void set(Object newElement)  //override
--------------------------------------------------------------------------------
----
import java.util.*;
```

```
class LinkedListDemo3
{
      public static void main(String[] args)
      {
            LinkedList city = new LinkedList();
         city.add("Kolkata");
             city.add("Bangalore");
             city.add("Hyderabad");
             city.add("Pune");
             System.out.println(city);

            ListIterator lt = city.listIterator();

         while(lt.hasNext())
            {
                  String x = (String) lt.next();

                  if(x.equals("Kolkata"))
                  {
                   lt.remove();
                  }

                  else if(x.equals("Hyderabad"))
                  {
                   lt.add("Ameerpet");
                  }

                  else if(x.equals("Pune"))
                  {
                   lt.set("Mumbai");
                  }
            }

            for(Object o : city)
            {
                  System.out.println(o);
            }
      }
}
```


Vector :

Vector is a predefined class available in java.util package under List interface. Vector is always from java means it is available from jdk 1.0 version.

Vector and Hashtable, these two classes are available from jdk 1.0, remaining classes were added from 1.2 version. That is the reason Vector and Hashtable are called legacy classes.

The main difference between Vector and ArrayList is ArrayList methods are not synchronized so multiple threads can access the method of ArrayList where as on the other hand most the methods are synchronized in Vector so performance wise Vector is slow.

*We should go with ArrayList when Threadsafety is not required on the other hand we should go with Vector when we need ThreadSafety

It also stores the elements on index basis.It is dynamically growable with initial capacity 10.

Just like ArrayList it also implments List, Serializable, Clonable, RandomAccess interfaces.

Ex:-
public class Vector extends AbstractList implements List, Serializable, Clonable, RandomAccess

Constructor in Vetor :

We have 4 constructors in Vector

1) Vector v = new Vector();   capacity is ----10

2) Vector v1 = new Vector(int initialCapacity); -------500

      new capacity = Current Capacity * 2;

3) Vector v2 = new Vector(int initialCapacity, int incrementalCapacity);

4) Vector v3 = new Vector(Collection c);

```
import java.util.*;
class VectorExampleDemo1
{
      public static void main(String[] args)
      {
            Vector v = new Vector();  //initial capacity is 10
            System.out.println("Initial capacity is :"+v.capacity());

        for(int i =1; i<=100; i++)
            {
             v.add(i);
            }
            System.out.println("After adding 10 elements  capacity 
is :"+v.capacity());
            v.add(11);
            System.out.println("After adding 11 elements  capacity 
is :"+v.capacity());

            System.out.println(v);
      }
}
```

```
//Vector also stores the element on the basis of index
 import java.util.Vector;
 class VectorExampleDemo2
      {
        public static void main(String[] args)
            {
                  Vector v = new Vector();
                  v.add(1);
                  v.add("2");
                  v.add(34.90f);
                  System.out.println(v);  //[1, 2, 34.90]
                  System.out.println("Getting elements of Vector");
                  System.out.println(v.get(0));
                  System.out.println(v.get(1));
                  System.out.println(v.get(2));
           }
}
```

```
import java.util.*;
class VectorExampleDemo3
{
      public static void main(String args[])
      {
            Vector<Integer> v = new Vector<Integer>();

            int x[]={22,20,10,40,15,58};

            for(int i=0; i<x.length; i++)
            {
                  v.add(x[i]);
            }

            Collections.sort(v);
            System.out.println("Maximum element is :"+Collections.max(v));
            System.out.println("Minimum element is :"+Collections.min(v));
            System.out.println("Vector Elements :");

            for(int i=0; i<v.size();i++)
            {
            System.out.println(v.get(i));
            }
      }
}
```

Stack :

It is a predefined class available in java.util package. It is the sub class of Vector class.

It is a linear data structure that is used to store the Objects in LIFO (Last In first out) basis.

Inserting an element into a Stack is known as push operation and It can be done by push() method whereas extracting an element from the stack is known as pop operation and it can be done by pop(). pop() method will extract and delete the element from the top of the stack.

If we want to extract the element from the top of the Stack without removing it then we should use peek() of Stack class.

It throws an exception called EmptyStackException.

It has only one constructor as shown below

Stack s = new Stack();

Method :

----

push(Object o) :- To insert an element

pop() :- To remove and return the element from the top of the Stack

peek() :- Will fetch the element from top of the Stack without removing

empty() :- Tests whether stack is empty or not (boolean return type)

search(Object o) :- It will search a particular element in the Stack and it returns OffSet. If the element is not present in the Stack it will return -1

```
--------------------------------------------------------------------------------
----
//Program to insert and fetch the elements from stack
import java.util.*;
class Stack1
{
      public static void main(String args[])
      {
            Stack<Integer> s = new Stack<Integer>();
            try
            {
                  s.push(0);
                  s.push(1);
                          s.push(2);
                          s.push(3);
                          s.push(4);
                          s.push(5);
                          System.out.println("After insertion elements 
are :"+s);
                  System.out.println(s.pop());
                  System.out.println(s.pop());
                  System.out.println(s.pop());
                  System.out.println(s.pop());
                  System.out.println(s.pop());
                          System.out.println(s.pop());
                          System.out.println("After deletion elements are :"+s);
                          System.out.println("Is the Stack is 
empty ? :"+s.empty());
               }
                 catch(EmptyStackException e)
                 {}
      }
}
--------------------------------------------------------------------------------
------
//add() is the method of Vector class
import java.util.*;
class Stack2
{
      public static void main(String args[])
      {
            Stack<Integer> st1 = new Stack<Integer>();
            st1.add(10);
            st1.add(20);
            for(int k : st1)
            {
                  System.out.println(k+"   ");
            }

            Stack<String> st2 = new Stack<String>();
            st2.add("Java");
            st2.add("is");
            st2.add("programming");
            st2.add("language");
            for(String k : st2)
            {
                  System.out.println(k+"   ");
            }

            Stack<Character> st3 = new Stack<Character>();
            st3.add('A');
            st3.add('B');
            for(Character k : st3)
            {
                  System.out.println(k+"   ");
            }

            Stack<Double> st4 = new Stack<Double>();
            st4.add(10.5);
            st4.add(20.5);
            for(Double k : st4)
            {
                  System.out.println(k+"   ");
            }
      }
}
--------------------------------------------------------------------------------
-----
import java.util.Stack;
public class Stack3
{
      public static void main(String[] args)
            {
                  Stack<String> stk= new Stack<String>();
                  stk.push("Apple");
                  stk.push("Grapes");
                  stk.push("Mango");
                  stk.push("Orange");
                  System.out.println("Stack: " + stk);

                  String fruit = stk.peek();
                  System.out.println("Element at top: " + fruit);
                  System.out.println("Stack elements are : " + stk);
            }
}

--------------------------------------------------------------------------------
-----
import java.util.Stack;
public class Stack4
{
      public static void main(String[] args)
            {
                  Stack<String> stk= new Stack<String>();
                  stk.push("Apple");
                  stk.push("Grapes");
                  stk.push("Mango");

                  System.out.println("Position is : " + stk.search("Mango"));

                  System.out.println("Position is : " + stk.search("Banana"));

                System.out.println("Is stack empty ?"+stk.empty());

            }
}
--------------------------------------------------------------------------------
-----
```
Set interface :

Set interface does not accept duplicate elements here our friend is equals(Object) method of Object class which compares two objects and if both objects are identical it will accept only one.

Set interface does not provide any own method, it inherits all the methods from Collection interface.

```
----
class Test
{
      public static void main(String[] args)
      {
            Test t1 = new Test();
            System.out.println(t1.hashCode());  //toString()

      }
}
```

Note :- JVM provides a unique number for each and every object we create in java. It is represented by hashCode() method of Object class.

HashSet :   [UNSORTED, UNORDERED, NO DUPLICATES]

It is a predefined class available in java.util package under Set interface.

It is an unsorted and unordered set.

It accepts hetrogeneous kind of data.

It uses the hashcode of the object being inserted into the Collection.

It doesn't contain any duplicate elements as well as It does not maintain any order while iterating the elements from the collection.

It can accept null value.

HashSet is used for searching operation.

```
import java.util.*;
class HashSetDemo
{
 public static void main(String args[])
 {
        HashSet<String> hs=new HashSet<String>();
        hs.add("Ravi");
        hs.add("Vijay");
        hs.add("Ravi");
        hs.add("Ajay");
        hs.add("Palavi");
        hs.add("Sweta");
        hs.add(null);
        hs.add(null);

     //Collections.sort(hs);   //Invalid not possible

        Iterator itr=hs.iterator();
        while(itr.hasNext())
        {
         System.out.println(itr.next());
        }
   }
}
```

From the above program It is clear that HashSet does not maintain any order.

```
import java.util.*;
public class HashSetDemo1
```

```
{
      public static void main(String[] argv)
      {
            boolean[] ba = new boolean[6];

            Set s = new HashSet();

            ba[0] = s.add("a");
            ba[1] = s.add(42);
            ba[2] = s.add("b");
            ba[3] = s.add("a");
            ba[4] = s.add("new Object()");
                  ba[5] = s.add(new Object());

            for(int x = 0; x<ba.length; x++)
                 System.out.print(ba[x]+"    ");

            System.out.println("\n");

                  for(Object o : s)
                 System.out.print(o+"   ");
      }
}
```

--
LinkedHashSet :

It is a predefined class in java.util package under Set interface.

It is the sub class of HashSet class

It is an orderd version of HashSet that maintains a doubly linked list across all the elements.

We should use LinkedHashSet class when we want to maintain an order.

When we iterate the elements through HashSet the order will be unpredictable, while when we iterate the elments through LinkedHashSet then the order will be same as they were inserted in the collection.

It accepts hetrogeneous and null value is allowed.

```
import java.util.*;
class LinkedHashSetDemo
{
 public static void main(String args[])
      {
              LinkedHashSet<String> al=new LinkedHashSet<String>();
              al.add("Ravi");
              al.add("Vijay");
              al.add("Ravi");
              al.add("Ajay");
              al.add("Pawan");
              al.add("Shiva");
              al.add("Ganesh");
          al.add(null);

             //Collections.sort(al);  //not possible

              Iterator itr=al.iterator();
              while(itr.hasNext())
              {
               System.out.println(itr.next());
              }
      }
}
--------------------------------------------------------------------------------
--
SortedSet :
------------
-> We can't perform sorting using HashSet and LinkedHashSet so java software
people has provided a separate interface i.e SortedSet interface

-> It is the sub interface of Set interface

-> We should go with SortedSet interface when we don't want duplicate
   elements and wants to store the element based on some default natural
   sorting order.

-> default natural sorting order
    --------------------------------
   a) For numbers -> For number the default natural sorting order is
                     ascending order.

   b) For String -> For String the default natural sorting order is
                    lexicographical comparison or alphabetical or
                    dictionary order comparison

--------------------------------------------------------------------------------
-
*What is the difference between Comparable and Comparator interface :

All wrapper classes (Byte, Short, Intgeger, Long, Float and so on) and String
class implements Comparable interface so we can compare two objects to sort
them.

Comparable interface contains a predefined method called compareTo(), which
compare the contant of two object on the following criteria

current Object data == another object data  -> 0

current object data > another object data  -> +ve OR 1

current object data < another object data  -> -ve OR -1

//Sorting the Customer data based on customer age using Comparable interface

//BLC
public class Customer implements Comparable<Customer>
{
   int cid;
   String cname;
   int cage;

   public Customer(int cid, String cname, int cage)
   {
      super();
      this.cid = cid;
      this.cname = cname;
      this.cage = cage;
   }

      @Override
      public int compareTo(Customer c1)
      {
            if(this.cage == c1.cage)
                  return 0;
            else if (this.cage > c1.cage)
                  return 1;
            else
                  return -1;
      }

}

//ELC
import java.util.ArrayList;
import java.util.Collections;

public class CustomerComparable
{

      public static void main(String[] args)
      {
            ArrayList<Customer> al = new ArrayList<Customer>();

            al.add(new Customer(101,"Zaheer",27));
            al.add(new Customer(103,"Ravi",23));
            al.add(new Customer(102,"Aryan",24));

            Collections.sort(al);

            for(Customer ct :al)
            {
      System.out.println("Cid = "+ct.cid+" Cname ="+ct.cname +" Cage
="+ct.cage);
            }


      }
}
--------------------------------------------------------------------------------
---
```

Sorting the employee data by Comparator interface :

```
-----------------------------------------------------------
//BLC
public class Employee
{
  int eid;
  String ename;
  String eaddr;
      public Employee(int eid, String ename, String eaddr)
      {
            super();
            this.eid = eid;
            this.ename = ename;
            this.eaddr = eaddr;
      }
}

//ELC
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

public class EmployeeComparator
{
      public static void main(String[] args)
      {
            ArrayList<Employee> al = new ArrayList<Employee>();
            al.add(new Employee(201, "Zaheer", "Mumbai"));
            al.add(new Employee(203, "Aryan", "Ahmedabad"));
            al.add(new Employee(202, "Dimple", "Hyderabad"));

            //Sorting the Employee Object based on employee id
            Comparator<Employee> cmpId = new Comparator<Employee>()
            {
                  @Override
                  public int compare(Employee e1, Employee e2)
                  {
                        return e1.eid - e2.eid;
                  }
            };

            Collections.sort(al, cmpId);
            System.out.println("Sorting the Employee based on Employee Id:");
            for (Employee e1 : al)
            {
                  System.out.println(e1.eid+": "+e1.ename+" : "+e1.eaddr);
            }

            //Sorting the Employee Object based on employee name
                        Comparator<Employee> cmpName = new
Comparator<Employee>()
                        {
                              @Override
                              public int compare(Employee e1, Employee e2)
                              {
                                    return e1.ename.compareTo(e2.ename);
                              }
                        };

                        Collections.sort(al, cmpName);
                        System.out.println("Sorting the Employee based on
Employee Name:");
                        for (Employee e1 : al)
                        {
                              System.out.println(e1.eid+": "+e1.ename+" :
"+e1.eaddr);
                        }

      }

}
```


Program to sort the Student Data using Comparator

```
//BLC
public class Student
{
  int roll;
  String name;

  public Student(int roll, String name)
  {
      super();
      this.roll = roll;
      this.name = name;
  }

}
```

```
//ELC

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

public class StudentComparator
{
      public static void main(String[] args)
      {
            ArrayList<Student> al = new ArrayList<Student>();
            al.add(new Student(333,"Aryan"));
            al.add(new Student(222,"Raj"));
            al.add(new Student(111,"Samir"));

            System.out.println("Sorting the student data based on roll number");
            Comparator<Student> cmpRoll = new Comparator<Student>()
            {
                  @Override
                  public int compare(Student s1, Student s2)
                  {
                        return s1.roll - s2.roll;
                  }
            };
            Collections.sort(al, cmpRoll);

            for(Student s : al)
            {
                  System.out.println(s.roll+" : "+s.name);
            }

            System.out.println("Sorting the student data based on the name");

            Comparator<Student> cmpName = new Comparator<Student>()
            {
                  @Override
                  public int compare(Student s1, Student s2)
                  {
                        return s1.name.compareTo(s2.name);
                  }
            };

            Collections.sort(al, cmpName);

            for(Student s : al)
            {
                  System.out.println(s.roll+" : "+s.name);
            }



      }
}
```


TreeSet :

It is a predefined class available in java.util package under Set interface.

TreeSet and TreeMap are the two sorted collection in the entire Collection Framework so both the classes never accepting hetrogeneous kind of the data.

It will sort the elements in natural sorting order i.e ascending order in case of number and alphabetical order in the case of String. In order to sort the elements It uses Comparable interface.

It does not accept duplicate and null value.

It does not accept hetrogeneous i.e non-comparable object type of data if we try to insert it will throw a runtime exception i.e java.lang.ClassCastException

TreeSet implements NavigableSet.

NavigableSet extends SortedSet

```
import java.util.*;
class TreeSetDemo
{
      public static void main(String[] args)
      {
            Set<Integer> t = new TreeSet<Integer>();
            t.add(4);
            t.add(7);
            t.add(2);
            t.add(1);
            t.add(9);
            //t.add(null);  not possible
            //t.add("Ravi"); not possible
            System.out.println(t);
      }
}
```

By dafault the elements are sorted by natural sorting order starting with smallest to largest.

But it is possible to itearte the elements in descending order by using the predefined method called descendingIterator() of TreeSet class as shown below.

Iteartor descendingIterator()

```
import java.util.*;
class TreeSetDemo1
{
      public static void main(String[] args)
      {
            TreeSet<String> t1 = new TreeSet<String>();
            t1.add("Orange");
            t1.add("Mango");
            t1.add("Pear");
            t1.add("Banana");
            t1.add("Apple");

            System.out.println("In Ascending order");
            Iterator itr1 = t1.iterator();
            while(itr1.hasNext())
                  {
                        String element = (String) itr1.next();
                        System.out.println(element);
                  }


            TreeSet<String> t2 = new TreeSet<String>();

            t2.add("Orange");
            t2.add("Mango");
            t2.add("Pear");
            t2.add("Banana");
            t2.add("Apple");

        System.out.println("In Descending order");
            Iterator itr2 = t2.descendingIterator();
            while(itr2.hasNext())
                  {
                        String element = (String) itr2.next();
                        System.out.println(element);
                  }
      }
}
--------------------------------------------------------------------------------
--
import java.util.*;
class TreeSetDemo2
{
      public static void main(String[] args)
      {
            Set<String> t = new TreeSet<String>();
            t.add("6");
            t.add("5");
            t.add("4");
            t.add("2");
            t.add("9");
            System.out.println(t);   //[A--65   B--66]
      }
}
--------------------------------------------------------------------------------
--
//customized sorting order
import java.io.*;
import java.util.Comparator;
import java.util.TreeSet;

class TreeSetDemo3
      {
      public static void main(String[] args)
      {
            System.out.println("Sorting name -> Ascending Order");

            TreeSet<Employee> ts1 = new TreeSet<Employee>(new
FirstComparator());

            ts1.add(new Employee(101, "Zaheer", 24));
            ts1.add(new Employee(201, "Aryan", 27));
            ts1.add(new Employee(301, "Pooja", 26));



            for (Employee e1 : ts1)
            {
                  System.out.println(e1);
            }
        System.out.println("----------------------------------------");

            System.out.println("Sorting name -> Descending Order");

            TreeSet<Employee> ts2
                  = new TreeSet<>(new SecondComparator());
            ts2.add(new Employee(101, "Zaheer", 24));
            ts2.add(new Employee(201, "Aryan", 27));
```

```java
            ts2.add(new Employee(301, "Pooja", 26));


            for (Employee e2 : ts2)
            {
                  System.out.println(e2);
            }
        System.out.println("----------------------------------------");
            System.out.println("Sorting Age -> Ascending Order");
            TreeSet<Employee> ts3
                  = new TreeSet<>(new ThirdComparator());

            ts3.add(new Employee(101, "Zaheer", 24));
            ts3.add(new Employee(201, "Aryan", 27));
            ts3.add(new Employee(301, "Pooja", 26));


            for (Employee e3 : ts3)
            {
                  System.out.println(e3);
            }

            System.out.println("----------------------------------------");
            System.out.println("Sorting Age ->  Descending Order");
            TreeSet<Employee> ts4 = new TreeSet<>(new FourthComparator());

            ts4.add(new Employee(101, "Zaheer", 24));
            ts4.add(new Employee(201, "Aryan", 27));
            ts4.add(new Employee(301, "Pooja", 26));


            for (Employee e4 : ts4)
            {
                  System.out.println(e4);
            }
      }
}

// for sorting name in ascending order
class FirstComparator implements Comparator<Employee>
{
      @Override
      public int compare(Employee e1, Employee e2)
      {
            return e1.name.compareTo(e2.name);
      }
}

// for sorting name in descending order
class SecondComparator implements Comparator<Employee>
{
      @Override
      public int compare(Employee e1, Employee e2)
      {
            return - (e1.name).compareTo(e2.name);
      }
}

// for sorting age in ascending order
class ThirdComparator implements Comparator<Employee>
{
      @Override public int compare(Employee e1, Employee e2)
      {
            return e1.age - e2.age;
      }
}


// for sorting age in descending order
class FourthComparator implements Comparator<Employee>
{
      @Override public int compare(Employee e1, Employee e2)
      {
            return - (e1.age - e2.age);
      }
}

// Employee class
class Employee
      {
      public int id;
      public String name;
      public Integer age;

      Employee(int id, String name, int age)
      {
            this.id = id;
            this.name = name;
            this.age = age;
      }

      @Override
      public String toString()
      {
            return " " + this.id + " " + this.name + " "+ this.age;
      }
}
```

--

Methods of SortedSet
----

1) first() :- will return first value

2) last() :- will return last value

3) headSet(int range) :- will return less than the specified value

4) tailSet(int range) :- will return equal or greater than the specified value

5) subSet(int startRange, int endRange) :- will return range of value where
     startRange is inclusive where as endRange is exclusive.


```
 import java.util.*;
public class SortedSetMethodDemo
{
       public static void main(String[] args)
       {
            TreeSet<Integer> times = new TreeSet<Integer>();
            times.add(1205);
            times.add(1505);
            times.add(1545);
                  times.add(1600);
            times.add(1830);
            times.add(2010);
            times.add(2100);
            SortedSet<Integer> sub = new TreeSet<Integer>();

                  sub =   times.subSet(1545,2100);
            System.out.println("Using subSet() :-"+sub);
            System.out.println(sub.first());
            System.out.println(sub.last());

                sub = times.headSet(1545);
                  System.out.println("Using headSet() :-"+sub);

                sub =   times.tailSet(1545);
                  System.out.println("Using tailSet() :-"+sub);
       }
}
```


Map interface :
----
*Why Map interface is not the part of Collection interface even it comes under Collection Framework ?

Collection interface works with individual Object where as Map interface works with group of Objects as a key-value pair that is the reason Map is a separate interface it is not the part of Collection.

A Map interface contains unique identifiers. In Map we have a unique key to a specific value, where key and value both are Object.

Just like Set interface, Map interface also depends upon equals() method of Object class to determine whether two keys are same or different.

Note : Collection interface is not the root interface of collection framework.

All the sub classes of collection interface work with individual object. But Map interface works with key-value pair where key and value both are object.

Before Map interface we have Dictionary class which is an abstract class and Hashtable is the sub class of dictionary class to hold the key-value pairs.

Methods of Map interface :
----
1) Object put(Object key, Object value) :- Used to insert key and value pair.

2) Object get(Object key) :- We will get the value of the specified key, if the key is not available it will return null.

3) void putAll(Map m) :- for merging two collection

4) int size() :- To find out the size of the Map

5) void clear() :- To clear the map that means the Map will become empty

6) remove(Object key) :- It will remove a particular Entry(key and value)

7) Object getOrDefault(Object key, Object default value) :- If the key is present in the map collection then it will return the corrosponding value otherwise it will set and return the specified default value.

8) boolean isEmpty() :- Verify whether map is empty or not

9) boolean containsKey(Object key) :- verify whether the key is available or not.

10) boolean containsValue(Object value) :- verify whether the value is available or not

11) Set keySet() :- It will return all the keys [set format]

12) Collection values() :- It will return all the values

13) Set<Map.Entry<K,V>> entrySet() :- will return key and value pair

HashMap :
----
public class HashMap<K,V> extends AbstractMap<K,V> implements Map<K,V>, clonable, Serializable

It is a predefined class available in java.util package under Map interface.

It gives us unsorted and Unordered map. when we need a map and we don't care about the order while iterating the elements through it then we should use HashMap.

It is unsynchronized and provides a gurantee that the order will remain constant.

It inserts the element based on the hashCode of the Object key using hashing technique.

It does not accept duplicate keys but value may be duplicate.

It accepts only one null key(because duplicate keys are not allowed) but multiple null values.

HashMap is not synchronized.

Time complexcity of search, insert and delete will be O(1)

We should use HashMap to perform searching opeartion.

It has 4 constructor

1) HashMap hm1 = new HashMap();
    will create HashMap Object with default capacity is 16 and load factor 0.75

2) HashMap hm2 = new HashMap(int initialCapacity);

3) HashMap hm3 = new HashMap(int initialCapacity, float loadFactor);

4) HashMap hm4 = new HashMap(HashMap hm);

```
import java.util.*;
public class HashMapDemo
{
      public static void main(String[] a)
      {
           Map<String,String> map = new HashMap<String,String>();
               map.put("Ravi", "12345");
               map.put("Rahul", "12345");
               map.put("Aswin", "5678");
               map.put(null, "6390");
               map.put("Ravi","1529");
               System.out.println(map.get(null));
               System.out.println(map.get("virat"));
               System.out.println(map);
      }
}
--------------------------------------------------------------------------------
-----
import java.util.*;

public class HashMapDemo1
{
      public static void main(String args[])
      {
            HashMap hm = new HashMap();
            hm.put(1, "JSE");
            hm.put(2, "JEE");
            hm.put(3, "JME");
            hm.put(4,"JavaFX");
            hm.put(5,null);

            System.out.println("Initial map elements: " + hm);
            System.out.println("key 2 is present or not :"+hm.containsKey(2));


            System.out.println("JME is present or 
not :"+hm.containsValue("JME"));
            System.out.println("Size of Map : " + hm.size());
            hm.clear();
            System.out.println("Map elements after clear: " + hm);
        }
}
--------------------------------------------------------------------------------
-----
```
Whenever we insert the key-value pair using put(Object key, Object value) then the put method will return the old Object value.

Note :-

If we want to fetch the key and value pair as an Entry then we should go with Map.Entry interface

Map is an interface which contains Entry interface.

```
//Collection view method by entrySet()
import java.util.*;
public class HashMapDemo2
{
public static void main(String args[])
      {
                  Map map = new HashMap();
                  map.put(1, "C");
                  map.put(2, "C++");
                  map.put(3, "Java");
                  map.put(4, ".net");
                  System.out.println(map);

                  System.out.println("Return old value :"+map.put(4,"Python"));

                  Set set=map.entrySet();
                  System.out.println("Set values: " + set); //[]

                  Iterator<Map.Entry> itr = set.iterator();
                  while(itr.hasNext())
               {
                        Map.Entry m =  itr.next(); //We receive each entry
                        System.out.println(m.getKey()+":"+m.getValue());

                        if(m.getKey().equals(4))
                     {
                              m.setValue("Advanced Java");
                     }
               }
                  System.out.println(map);
      }
}
--------------------------------------------------------------------------------
----
import java.util.*;

public class HashMapDemo3
{
public static void main(String args[])
  {
            HashMap<Integer,String> map = new HashMap<Integer,String>(10);
            map.put(1, "Java");
            map.put(2, "is");
            map.put(3, "best");
            map.remove(3);
            String val=(String)map.get(3);
            System.out.println("Value for key 3 is: " + val);
   }
}
--------------------------------------------------------------------------------
-----
import java.util.*;

public class HashMapDemo4
{
public static void main(String args[])
      {
            HashMap newmap1 = new HashMap();
            HashMap newmap2 = new HashMap();


            newmap1.put(1, "SCJP");
            newmap1.put(2, "is");
            newmap1.put(3, "best");

            System.out.println("Values in newmap1: "+ newmap1);

            newmap2.put(4, "Exam");
            newmap2.putAll(newmap1);

            System.out.println("Values in newmap2: "+ newmap2);
   }
}
--------------------------------------------------------------------------------
-----
import java.util.*;
public class HashMapDemo5
{
     public static void main(String[] argv)
     {
          Map map = new HashMap(9, 0.85f);
          map.put("key", "value");
          map.put("key2", "value2");
          map.put("key3", "value3");
```

```
               Set k_set = map.keySet();//keySet return type is Set
               System.out.println(k_set );

          Collection v_set = map.values(); //values return type is collection
          System.out.println(v_set);

              map.clear();
          System.out.println(map);
      }
}
--------------------------------------------------------------------------------
-----
import java.util.*;
class  HashMapDemo6
{
      public static void main(String[] args)
      {
            Map<String, String> map = new HashMap<String, String>();

            map.put("A", "1");
            map.put("B", "2");
            map.put("C", "3");

            String value = map.getOrDefault("D", "Key is not Present");
            System.out.println(value);
            System.out.println(map);
      }
}
--------------------------------------------------------------------------------
----
//interconversion of two HashMap
import java.io.*;
import java.util.*;

class HashMapDemo7
      {
      public static void main(String args[])
      {

            Map<Integer, String> hm1 = new HashMap<Integer, String>();

            hm1.put(1, "Ravi");
            hm1.put(2, "Rahul");
            hm1.put(3, "Rajen");

      HashMap<Integer, String> hm2  = new HashMap<Integer,String>(hm1);

            System.out.println("Mappings of HashMap hm1 are : "   + hm1);

            System.out.println("Mapping of HashMap hm2 are : " + hm2);
      }
}
--------------------------------------------------------------------------------
-----
```

LinkedHashMap :
----

public class LinkedHashMap<K,V> extends HashMap<K,V> implements Map

It is a predefined class available in java.util package under Map interface.

It is the sub class of HashMap class.

It maintains insertion order actually It is an ordered version of HashMap. It contains a doubly linked with the elements or nodes so It will iterate more slowly in comparison to HashMap.

It uses Hashtable and LinkedList data structure.

If We want to fetch the elements in the same order as they were inserted then we should go with LinkedHashMap.

It accepts one null key and multiple null values.

It is not synchronized.

It has also 4 constructors same as HashMap

```
1) LinkedHashMap hm1 = new LinkedHashMap();
    will create an empty LinkedHashMap

2) LinkedHashMap hm1 = new LinkedHashMap(iny initialCapacity);

3) LinkedHashMap hm1 = new LinkedHashMap(iny initialCapacity, float loadFactor);

4) LinkedHashMap hm1 = new LinkedHashMap(Map m);
```

```
import java.util.*;
class LinkedHashMapDemo
{
      public static void main(String[] args)
      {
            LinkedHashMap<Integer,String> l = new LinkedHashMap();
            l.put(1,"abc");
            l.put(3,"xyz");
            l.put(2,"pqr");
            l.put(4,"def");
            l.put(null,"ghi");
            System.out.println(l);
      }
}
```

```
import java.util.LinkedHashMap;
import java.util.Map;

public class LinkedHashMapDemo1
{
      public static void main(String[] a)
      {
           Map<String,String> map = new LinkedHashMap<String,String>();
           map.put("Ravi", "1234");
               map.put("Rahul", "1234");
               map.put("Aswin", "1456");
               map.put("Samir", "1239");
           System.out.println(map);
      }
}
```

Hashtable :

It is predefined class available in java.util package under Map interface.

Like Vector, Hashtable is also form the birth of java so called legacy class.

It is the sub class of Dictionary class which is an abstract class.

The major difference between HashMap and Hashtable is, HashMap can have null values as well as null keys where as Hashtable does not contain anything as a null. if we try to add null value JVM will throw an exception i.e NullPointerException.

The initial default capacity of Hashtable class is 11 whereas loadFactor is 0.75.

Like Vector Hashtable methods are also synchronized So it provides ThreadSafety

It has also same constructor as we have in HashMap.(4 constructors)

Constructors :
----

1) Hashtable ht1 = new Hashtable();
    create Hashtable Object with initial capacity as 11 and load factor 0.75

2) Hashtable ht2 = new Hashtable(int initialCapacity);

3) Hashtable ht3 = new Hashtable(int initialCapacity, float loadFactor);

4) Hashtable ht4 = new Hashtable(Map m);

```
--------------------------------------------------------------------------------
-----
import java.util.*;
class HashtableDemo
      {
       public static void main(String args[])
            {
              Hashtable<Integer,String> map=new Hashtable<Integer,String>();
              map.put(1, "Java");
              map.put(2, "is");
              map.put(3, "best");
              map.put(4,"language");
              //map.put(null,"program");

              System.out.println(map);

              for(Map.Entry m : map.entrySet())
                   {
                         System.out.println(m.getKey()+" = "+m.getValue());
                  }
        }
}
--------------------------------------------------------------------------------
-----
```

putIfAbsent() :-

This method will insert the Entry if and only if the specified key is not available in the Map collection.

```
--------------------------------------------------------------------------------
-----import java.util.*;
class HashtableDemo1
{
   public static void main(String args[])
      {
    Hashtable<Integer,String> map=new Hashtable<Integer,String>();
     map.put(1,"Priyanka");
     map.put(2,"Ruby");
     map.put(3,"Vibha");
     map.put(4,"Kanchan");
```

```
       System.out.println("Initial Map: "+map);

         map.putIfAbsent(5,"Bina");
       map.putIfAbsent(24,"Pooja");
       map.putIfAbsent(26,"Ankita");
     System.out.println("Updated Map: "+map);

     map.putIfAbsent(1,"Priya");
     System.out.println("Updated Map: "+map);
 }
}
```


IdintityHashMap :
----

public class IdentityHashMap<K,V> extends AbstractMap<K,V> implements Map<K,V>, Clonable, Serializable.

It was introduced from JDK 1.4 onwards.

The IdentityHashMap uses == operator to compare keys and values.

As we know HashMap uses equals() and hashCode() method for comparing objects.

So We should use IdentityHashMap where we need to check the reference instead of logical equality.

HashMap uses hashCode of the "Object key" to find out the bucket loaction, on the other hand IdentityHashMap does not use hashCode() method actually It uses System.identityHashCode(Object o)

It has 3 constructors :
----

1) IdentityHashMap ihm1 = new IdentityHashMap();

2) IdentityHashMap ihm2 = new IdentityHashMap(int initialCapacity);

3) IdentityHashMap ihm1 = new IdentityHashMap(Map m);


```
import java.util.*;
public class IdentityHashMapDemo
{
      public static void main(String[] args)
      {
            HashMap<String,Integer> hm = new HashMap<String,Integer>();

            IdentityHashMap<String,Integer> ihm = new
IdentityHashMap<String,Integer>();

            hm.put("Ravi",23);
            hm.put(new String("Ravi"), 24);

            ihm.put("Ravi",23);
            ihm.put(new String("Ravi"), 24); //compares based on == operator

            System.out.println("HashMap size :"+hm.size());
            System.out.println("IdentityHashMap size :"+ihm.size());

      }

}
```

```
----
WeakHashMap :
----------------
public WeakHashMap<K,V> extends AbstractMap<K,V> implements Map<K,V>

It is a predefined class in java.util package under Map interface.

While working with HashMap, keys of HashMap are of strong reference type. This
means the entry of  map is not removed by the garbage collector even though the
key is set to be null and  still it is not eligible for Garbage Collector.

On the other hand while working with WeakHashMap, keys of WeakHashMap are of
weak reference type. This means the entry of a map is  removed by the garbage
collector because it is of weak type.

So, HashMap dominates over Garbage Collector where as Garbage Collector
dominates over WeakHashMap.

It contains 4 types of Constructor :
---------------------------------------
1) WeakHashMap wm = new WeakHashMap();

    Creates an empty WeakHashMap object with default capacity is 16 and load
fator 0.75

2) WeakHashMap wm1 = new WeakHashMap(int initialCapacity);


3) WeakHashMap wm2 = new WeakHashMap(int initialCapacity, float loadFactor);

    Eg:- WeakHashMap wm2 = new WeakHashMap(10,0.9);

    capacity - The capacity of this map is 10. Meaning, it can store 10 entries.

    loadFactor - The load factor of this map is 0.9. This means whenever our
hash table is filled by 90%, the entries are moved to a new hash table of double
the size of the original hash table.

4) WeakHashMap wm = new WeakHashMap(Map m);
--------------------------------------------------------------------------------
-----
import java.util.*;
class WeakHashMapDemo
{
      public static void main(String args[])  throws Exception
      {
            WeakHashMap m = new WeakHashMap();

            Test  t  = new Test();
            m.put(t," Rahul ");  //here we are passing reference 't' as a key

            System.out.println(m);

            t = null;

            System.gc();

            Thread.sleep(3000);

            System.out.println(m);
      }
}

class Test
```

```
{
      public String toString()
      {
            return "demo";
      }

      public void finalize()
      {
            System.out.println("finalize method is called");
      }
}
```


SortedMap :
----

1) firstKey()

2) lastKey()

3) headMap(int keyRange)

4) tailMap(int keyRange)

5) subMap(int startRange, int endRange)

```
import java.util.*;
class SortedMapMethodDemo
      {
 public static void main(String args[])
       {
            SortedMap<Integer,String> map=new TreeMap<Integer,String>();
              map.put(100,"Amit");
              map.put(101,"Ravi");
              map.put(102,"Vijay");
              map.put(103,"Rahul");
              System.out.println("First Key: "+map.firstKey());
               System.out.println("Last Key "+map.lastKey());
               System.out.println("headMap: "+map.headMap(102));
               System.out.println("tailMap: "+map.tailMap(102));
               System.out.println("subMap: "+map.subMap(100, 102));


       }
 }
```


Queue :-
----
1) It is sub interface of Collection(I)

2) It works in FIFO(First in first out) order.

3) It is an ordered collection.

4) In a queue, insertion is possible from last is called REAR where as deletion is possible from the starting is called FRONT of the queue.

5) From jdk 1.5 onwards LinkedList class implments Queue interface to handle the basic queue operations.

PriorityQueue :-

It is a predefined class in java.util package from Jdk 1.5 onwards.

The LinkedList class has been enhanced to implement Queue interface to handle basic operation of Queue.

A PriorityQueue is used when we want to store the Objects based on some priority.

The main purpose of PriorityQueue class is to create a "Priority in, Priority out" queue which is opposite to classical FIFO order.

A priority Queue stores the element in a natural sorting order where the elements which are sorted first, will be accessed first.

A priority queue does not permit null elements as well as It uses Comparator interface to sort the elements.

It provides natural sorting order so we can't take non-comparable objects.

The initial capacity of PriorityQueue is 11.

Methods :-
----
offer() :- Used to add an element, It can be also done by add()

poll() :- It is used to fetch the elements from top of the queue, after fetching it will delete the elements.

peek() :- It is also used to fetch the elements from top of the queue, it will not delete the elements like poll()

boolean remove(Object element) :- It is used to remove an element. The return type is boolean.

```
import java.util.PriorityQueue;

public class PriorityQueueDemo
{
      public static void main(String[] argv)
      {
            PriorityQueue<String> pq = new PriorityQueue<String>();
            pq.add("Orange");
                  pq.add("Apple");
                  //pq.add(null); not possible
                  System.out.println(pq.peek() );
                  System.out.println(pq.poll() );
                  System.out.println(pq.peek() );
      }
}
```

```
import java.util.PriorityQueue;
public class PriorityQueueDemo1
{
      public static void main(String[] argv)
      {
            PriorityQueue<String> pq = new PriorityQueue<String>();
            pq.add("9");
            pq.add("8");
                  pq.add("7");
            System.out.print(pq.peek() + " "); //7356
            pq.offer("6");
```

```
                 pq.offer("5");
            pq.add("3");
            pq.remove("1");
            System.out.print(pq.poll() + " ");
            if (pq.remove("2"))
                System.out.print(pq.poll() + " ");
            System.out.println(pq.poll() + " " + pq.peek());
      }
}
```


```
import java.util.PriorityQueue;

public class PriorityQueueDemo2
{
      public static void main(String[] argv)
      {
            PriorityQueue<String> pq = new PriorityQueue<String>();
            pq.add("2");
            pq.add("4");
                 pq.add("6");
            System.out.print(pq.peek() + " ");
            pq.offer("1");
                 pq.offer("9");
            pq.add("3");
            pq.remove("1");
            System.out.print(pq.poll() + " ");
            if (pq.remove("2"))
                System.out.print(pq.poll() + " ");
            System.out.println(pq.poll() + " " + pq.peek()+"  "+pq.poll());
        }
}
```

-----

classes in java.util package :-

Date :- It is a predefined class available in java.util package to fetch the current system date and time.

-

```
import java.util.*;
class DateTime
{
      public static void main(String[] args)
      {
            Date d = new Date();
            System.out.println(d);  //d.toString();
      }
}
```


Calendar class :-

It is predefined abstract class in java.util package. It is an abstract class so we can't craete an object but we have a static method inside the Calendar class to get the instance(Object) of Calendar class.

```
Calendar c1 = Calendar.getInstance();
```

It contains get() method where we can pass the final static variables.

Now we have number of predefined final static variables of Calendar class which are as follows :-

1) Calendar.YEAR :- will display the current year

2) Calendar.MONTH :- will display the current month , month starts from
                                  0 (Zero)

3) Calendar.DATE :- will display the current date.

4) Calendar.HOUR :- will display current hour according to the system

5) Calendar.MINUTE :- will display current minute according to the system
----
```
import java.util.*;
public class CalendarTest
{
  public static void main(String[] args)
        {

         Calendar cal = Calendar.getInstance();
         System.out.println("Current Year is :" + cal.get(Calendar.YEAR));
         System.out.println("Current Month is : " + cal.get(Calendar.MONTH));
         System.out.println("Current Day is : " + cal.get(Calendar.DATE));
           System.out.println("Current Hour is : " + cal.get(Calendar.HOUR));
         System.out.println("Current Minute is : " + cal.get(Calendar.MINUTE));

      }
}
```

```
import java.util.*;
public class CalendarTest1
      {
   public static void main(String[] args)
         {
               Calendar cal= Calendar.getInstance();
               System.out.print("The current system date and Time is : " +
cal.getTime());
      }
}
```

StringTokenizer :-

It is a predefined class available in java.util package. It is used to break the String into number of pieces , called tokens.

These tokens are stored in StringTokenizer object from where we can fetch the String using these tokens.

Methods :-
----
1) int countTokens() :-

To count the number of tokens.

2) boolean hasMoreTokens() :-

It will verify the tokens are available or not, if available it will return true if not it will return false.

3) String nextToken() :-

It will fetch the String as a token.

```
--------------------------------------------------------------------------------
-----
import java.util.*;
class STDemo
{
public static void main(String [] args)
      {

            String str ="Hyderabad is a lovely place";
            StringTokenizer st = new StringTokenizer(str,"a");

            System.out.println("Number of tokens :"+st.countTokens());
            System.out.println("The tokens are :");
            while(st.hasMoreTokens())
            {
                  String token = st.nextToken();
                  System.out.println(token);
            }
      }
}

--------------------------------------------------------------------------------
-----
Generics :
-----------
Why generic came into picture :
-------------------------------
As we know our compiler is known for Strict type checking because java is a
strongle typed checked language.

The basic problem with collection is It can hold any kind of Object.

ArrayList al = new ArrayList();
al.add("Ravi");
al.add("Aswin");
al.add("Rahul");
al.add("Raj");
al.add("Samir");

for(int i =0; i<al.size(); i++)
{
   String s = (String) al.get(i);
   System.out.println(s);
}

By looking the above code it is clear that Collection stores everything in the
form of Object so here even after adding String type only we need type casting
as shown below

import java.util.*;
class Test1
{
      public static void main(String[] args)
      {
            ArrayList al = new ArrayList();
            al.add("Ravi");
            al.add("Ajay");

            for(int i=0; i<al.size(); i++)
            {
            String name =(String) al.get(i);
            System.out.println(name);
            }
      }
}
```

Even after type casting there is no gurantee that the things which are coming from ArrayList Object is String only because we can add anything in the Collection as a result java.lang.ClassCastException as shown in the program below

```
 import java.util.*;
class Test1
{
      public static void main(String[] args)
      {
            ArrayList al = new ArrayList();
            al.add("Ravi");
            al.add("Ajay");
            al.add(12);

            for(int i=0; i<al.size(); i++)
            {
            String name =(String) al.get(i); //can't perform type casting
because Integer Object  has also added in the ArrayList.
            System.out.println(name);
            }
      }
}
```

To avoid all the above said problem Generics came into picture from JDK 1.5 onwards

-> It deals with type safe Object so there is a gurantee of both the end i.e putting inside and getting out

Advantages :-
----
a) Type safe Object (No compiler warning)

b) Strict compile time checking

c) No need of type casting

```
 import java.util.*;
class Test2
{
      public static void main(String[] args)
      {
            ArrayList<String> al = new ArrayList<String>();
            al.add("Ravi");
            al.add("Ajay");
            al.add("Vijay");
            //al.add(12);   // It is a compilation error

        for(int i=0; i<al.size(); i++)
            {
            String name = al.get(i); //no type casting is required
            System.out.println(name);
            }
      }
}
```

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

```
public class Test3
{
public static void main(String[] args)
{
            ArrayList t = new ArrayList();
            t.add("alpha");
            t.add("beta");
            for (int i = 0; i < t.size(); i++)
            {
              String str =(String) t.get(i);
              System.out.println(str);
            }
             t.add(1234);
             t.add(1256);
              for (int i = 0; i < t.size(); ++i)
             {
                   Object obj=t.get(i); //we can't perform type casting here
                   System.out.println(obj);
              }
   }
}
```


Whenever we print any Object reference then it will automatically call the toString() of Object class.

```
 import java.util.*;
class Test
{
      public static void main(String[] args)
      {
         Test t1 = new Test();
             System.out.println(t1);  //toString() method of Obejct class

      }
}
```


```
//Program that describes the return type of any method can be type safe
import java.util.*;
public class Test4
{
      public static void main(String [] args)
      {
            Dog d1 = new Dog();
            Dog d2 =  d1.getDogList().get(0);
            System.out.println(d2);
      }
}
class Dog
{
      public List<Dog> getDogList()
      {
            List<Dog> d = new ArrayList<Dog>();
            d.add(new Dog());
            d.add(new Dog());
            return d;
      }
}
```

Note :- In the above program the compiler will stop us from returning anything which is not compaitable List<Dog> and there is a gurantee that only "type safe list of Dog object" will be returned so we need not to provide type casting as shown below

```
Dog d2 = (Dog) d1.getDogList().get(0);  //before generic we need type cast
--------------------------------------------------------------------------------
-----
```

It is apossible to assign generic to raw type as well as raw to generic.

```
import java.util.*;
class Car
{
}
class Test5
{
      public static void main(String [] args)
      {
      ArrayList<Car> a = new ArrayList<Car>();
      a.add(new Car());

      ArrayList b = a;  //Generic to raw type


      for (Object obj : b)
            System.out.println(obj);
      }
}
--------------------------------------------------------------------------------
-----
//Mixing generic to non-generic
import java.util.*;
public class Test6
{
      public static void main(String[] args)
      {
            List<Integer> myList = new ArrayList<Integer>();
            myList.add(4);
            myList.add(6);
            myList.add(5);

            UnknownClass u = new UnknownClass();
            int total = u.addValues(myList);
            System.out.println(total);
      }
}
class UnknownClass
{
      int addValues(List list) //type safe Integer Object  to raw type
      {
      Iterator it = list.iterator();
      int total = 0;
      while (it.hasNext())
      {
      int i = ((Integer)it.next());
      total += i;                           //total = 15
      }
      return total;
      }
}
```

In the above program the compiler will not generate any warning message because even though we are assigning type safe Integer Object to unsafe or raw type List Object but this List Object is not inserting anything in the collection so there is no risk to the caller.

```
--------------------------------------------------------------------------------
-----
//Mixing generic to non-generic
import java.util.*;
public class Test7
```

```
{
      public static void main(String[] args)
      {
            List< Integer > myList = new ArrayList< Integer >();
            myList.add(4);
            myList.add(6);
            UnknownClass u = new UnknownClass();
            int total = u.addValues(myList);
            System.out.println(total);
      }
}
class UnknownClass
{
int addValues(List list)
      {
            list.add(5);      //adding object to raw type
            Iterator it = list.iterator();
            int total = 0;
            while (it.hasNext())
            {
            int i = ((Integer)it.next());
            total += i;
            }
            return total;
      }
}
```

In the above program the compiler will generate warning message because the unsafe List Object is inserting the Integer object 5 so the type safe Integer objcet is getting value 5 from unsafe type so there is a problem to the caller method.

By writing ArrayList<Integer> actually JVM does not have any idea that our ArrayList was suppose to hold only Integers.

All the type safe information does not exist at runtime. All our generic code is Strictly for compiler. There is a process done by java compiler called "Type erasure" in which the java compiler converts generic version to non-generic type.

```
List<Integer> myList = new ArrayList<Integer>();
```

At the compilation time it is fine but at runtime for JVM the code becomes

```
List myList = new ArrayList();
```

Note :- GENERIC IS STRICTLY A COMPILE TIME PROTECTION

Polymorphism with Array :

```
//Polymorphism with array
import java.util.*;
abstract class Animal
{
      public abstract void checkup();
}

class Dog extends Animal
{
      public void checkup()
      {
            System.out.println("Dog checkup");
      }
```

```
}

class Cat extends Animal
{
      public void checkup()
      {
            System.out.println("Cat checkup");
      }
}

class Bird extends Animal
{
      public void checkup()
      {
            System.out.println("Bird checkup");
      }
}
public class  Test8
{
      public void checkAnimals(Animal ...x)
      {
            for(Animal a : x)
            {
                  a.checkup();
            }
      }
      public static void main(String[] args)
      {
            Dog[] dogs={new Dog(), new Dog()};

            Cat[] cats={new Cat(), new Cat(), new Cat()};

            Bird[] birds = {new Bird()};

            Test8 t = new Test8();

            t.checkAnimals(dogs);
            t.checkAnimals(cats);
            t.checkAnimals(birds);
      }
}
--------------------------------------------------------------------------------
-----
Polymorphism with generic
------------------------------
import java.util.*;
abstract class Animal
{
      public abstract void checkup();
}

class Dog extends Animal
{
      public void checkup()
      {
            System.out.println("Dog checkup");
      }
}

class Cat extends Animal
{
      public void checkup()
      {
            System.out.println("Cat checkup");
      }
}
class Bird extends Animal
{
      public void checkup()
      {
            System.out.println("Bird checkup");
      }
}
public class Test9
{
      public void checkAnimals(List<Animal> animals)
      {

      }
      public static void main(String[] args)
      {
            List<Dog> dogs = new ArrayList<Dog>();
            dogs.add(new Dog());
            dogs.add(new Dog());

            List<Cat> cats = new ArrayList<Cat>();
            cats.add(new Cat());
            cats.add(new Cat());

            List<Bird> birds = new ArrayList<Bird>();
            birds.add(new Bird());

            Test9 t = new Test9();
            t.checkAnimals(dogs);
            t.checkAnimals(cats);
            t.checkAnimals(birds);

      }
}
```

So from the above program it is clear that polymorphism does not work in the same way for generics as it does with arrays.

Eg:-

```
Parent [] arr = new Child[5]; //valid
Object [] arr = new String[5]; //valid
```

But in generics the same type is not valid

```
List<Object> list = new ArrayList<Integer>(); //Invalid
List<Number> mylist = new ArrayList<Integer>(); //Invalid
```


```
class Test10
{
public static void main(String [] args)
      {
            Object []obj = new String[3]; //valid
            obj[0] = "Ravi";
            obj[1] = "hyd";
            obj[2] = 12;  //java.lang.ArrayStoreException
      }
}
```


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

```
class Parent
{

}
class Child extends Parent
{
}
class Test11
{
public static void main(String [] args)
      {
            //ArrayList<Parent> lp = new ArrayList<Child>();//error

            ArrayList<Parent> lp1 = new ArrayList<Parent>();

            ArrayList<Child> lp2 = new ArrayList<Child>();
      }
}
```

<?> wild card character :

As we know by default Polymorphism does not work with Generics so java software people has provided wild card character <?> which will accept any type as shown in the program below.

```
//program on wild-card chracter
import java.util.*;
class Parent
{

}
class Child extends Parent
{
}
class Test12
{
public static void main(String [] args)
      {
            List<?> lp = new ArrayList<Child>();
            System.out.println("Wild card....");
      }
}
```

```
<?>               -: Many possibilities

<Dog>       -: Only <Dog> can assign

<? super Dog>     -: Dog, Animal, Object can assign (Compiler has surity)

<? extends Dog) -: Below of Dog means, sub class of Dog (But the
                              compiler does not have surity because you can have
many sub class of Dog in the future, so chances of wrong collections)
```

```
import java.util.*;
class Test13
{
      public static void main(String[] args)
      {

            List<? extends Number> list1 = new ArrayList<Integer>();
            List<? super String> list2 = new ArrayList<Object>();

            List<Integer> list3 = new ArrayList<Integer>();

            List list4 = new ArrayList();

            System.out.println("yes");
      }
}
--------------------------------------------------------------------------------
----
import java.util.*;
class  Test14
{
      public static void main(String[] args)
      {
            try
            {
                  List<Object> x = new ArrayList<Object>();
            x.add(10);
                  System.out.println(x);
            }
            catch (Exception e)
            {
                  System.out.println(e);
            }
      }
}
--------------------------------------------------------------------------------
-----
```

<T> type parameter or type argument :

Java has introduced <T> type parameter or type argument which can accept any type of parameter because it is independent of data type.

```
class MyClass<T>
{
      T obj;
      MyClass(T obj)
      {
            this.obj=obj;
      }

      T getobj()
      {
            return obj;
      }
}
class Test15
{
      public static void main(String[] args)
      {
            Integer i=12;
            MyClass<Integer> mi = new MyClass<Integer>(i);
            System.out.println("Integer object stored :"+mi.getobj());

            Float f=12.34f;
            MyClass<Float> mf = new MyClass<Float>(f);
            System.out.println("Float object stored :"+mf.getobj());

            MyClass<String> ms = new MyClass<String>("Rahul");
            System.out.println("String object stored :"+ms.getobj());
      }
}
```


E parameter and Argument :

'E' stands for Element, we can pass this 'E' parameter to any class or method which will accept any type of Element.

```
class Fruit
{
}
class Apple extends Fruit   //Fruit is the super, Apple is sub class
{
}
class Basket<E>
{
      private E element;
      public void setElement(E x)
      {
            element = x;
      }
      public E getElement()
      {
            return element;
      }
}

class Test16
{
      public static void main(String[] args)
      {
            Basket<Fruit> b = new Basket<Fruit>();
            b.setElement(new Apple());

            Apple x =(Apple) b.getElement();
            System.out.println(x); //toString() of Object class

      }
}
```


Inner class in java :

In java it is possible to define a class (inner class) inside another class (outer class)

Java supports four kinds of inner classes :
----

1) Nested inner class OR Member class OR Regular class

2) Method local inner class

3) Static nested inner class

4) Anonymous inner class

Member Inner class OR Nested Inner class OR Regular class :

A non-static class that is created inside a class but outside of a method is called Member Inner class OR Nested Inner class OR Regular class.

It can be declared with access modifiers like public, default, private, protedcted, final and abstract.

```
class Outer
{
	int a = 7;
	class Inner
	{
		public  void displayValue()
		{
			System.out.println("Value of a is " + a);
		}
	}
}
public class Test1
{
	public static void main(String... args)
	{

		//Outer.Inner inner = new Outer().new Inner();

		Outer mo = new Outer();
		Outer.Inner inner = mo.new Inner();

		inner.displayValue();
	}
}
```

```
class MyOuter
{
	private int x = 7;
	public void makeInner()
	{
		MyInner in = new MyInner();
		in.seeOuter();
	}

	class MyInner
	{
		public void seeOuter()
		{
			System.out.println("Outer x is "+x);
		}
	}
}
public class Test2
{
	public static void main(String args[])
	{
		MyOuter m = new MyOuter();
		m.makeInner();
	}
}
```

```
class MyOuter
{
	private int x = 15;
	class MyInner
	{
		public void seeOuter()
		{
			System.out.println("Outer x is "+x);
```

```
            }
      }
}
public class Test3
{
      public static void main(String args[])
      {
           MyOuter.MyInner m = new MyOuter().new MyInner();
                  m.seeOuter();
      }
}
--------------------------------------------------------------------------------
-----
class MyOuter
{
      static int x = 7;
      class MyInner
      {
            public static void seeOuter()
            {
                  System.out.println("Outer x is "+x);
            }
      }
}
public class Test4
{
      public static void main(String args[])
      {
          MyOuter.MyInner.seeOuter();
      }
}
--------------------------------------------------------------------------------
-----
class OuterClass
{
      int x;
      public class InnerClass
      {
            int x;
      }
}
public class Test5
{
}
--------------------------------------------------------------------------------
-----
class OuterClass
{
      int x;
      protected class InnerClass
      {
            int x;
      }
}
public class Test6
{
}
--------------------------------------------------------------------------------
-----class OuterClass
{
      int x;
      static class InnerClass
      {
            int x;
      }
}
public class Test7
{
}
--------------------------------------------------------------------------------
----
class OuterClass
{
      int x;
      abstract class InnerClass
      {
            int x;
      }
}
public class Test8
{
}
--------------------------------------------------------------------------------
-----
class OuterClass
{
      int x;
      final class InnerClass
      {
            int x;
      }
}
public class Test9
{
}
```

Note:- From Test5.java to Test9.java programs show that we can declare an inner class as public, protected, abstract, final and static

```
--------------------------------------------------------------------------------
----
class OuterClass   //OuterClass.class
{
      private int x=200;
      class InnerClass     //OuterClass$InnerClass.class
      {
            public void display()
            {
            System.out.println("Inner class display");
            }

            public void getValue()
            {
                  display();
                  System.out.println("Can access outer private var :"+x);
            }
      }
            public void display()
            {
                  System.out.println("Outer class display");
            }
}
public class Test10  //Test10.class
{
      public static void main(String [] args)
      {
            OuterClass outobj = new OuterClass();
            OuterClass.InnerClass inobj=outobj.new InnerClass();
            inobj.getValue();

      }
}
```

Note :- In the above program we have a display method inside the inner class as well as inside the outer class but inner class Object will give more priority to inner class display method.

If we want to call outer class display method, we can call with the help of outer class Object reference.

----

2) Method local inner class :

If a class is declared inside the method then it is called local inner class.

We cann't apply any access modifier on method local inner class but they can be marked as abstract and final.

A local inner class we can't access outside of the method.

```
//program on method local inner class
class MyOuter3
{
      private String x = "Outer3";
      void doSttuff()
      {
           String z = "local variable";  //must be final till JDK 1.7
            class MyInner                   //only final and abstract
           {
                 public void seeOuter()
                 {
                      System.out.println("Outer x is "+x);
                      System.out.println("Local variable z is : "+z);
                 }
           }
           MyInner mi = new MyInner();
           mi.seeOuter();
      }
}
public class Test11
{
      public static void main(String args[])
      {
            MyOuter3 m = new MyOuter3();
            m.doSttuff();
      }
}
```

```
//local inner class we can't access outside of the method
class MyOuter3
{
      private String x = "Outer3";
      void doSttuff()
      {
           String z = "local variable";
            class MyInner
           {
                 public void seeOuter()
                 {
                      System.out.println("Outer x is "+x);
                      System.out.println("Local variable z is : "+z);
                 }
           }
      }
                  MyInner mi = new MyInner();//invalid
                  mi.seeOuter();
}
public class Test12
{
      public static void main(String args[])
      {
            MyOuter3 m = new MyOuter3();
            m.doSttuff();
      }
}
--------------------------------------------------------------------------------
-----
```

3) Static Nested Inner class :

A static inner class which is declared inside an outer class is called static Nested inner class.

It cann't access non-static variables and methods of outer class.

```
//static nested inner class
class BigOuter
{
     static class Nest
     {
          void go()
          {
               System.out.println("Hello welcome to static nested class");
          }
     }
}
class Test13
{
      public static void main(String args[])
      {
            BigOuter.Nest n = new BigOuter.Nest();
            n.go();

      }
}
--------------------------------------------------------------------------------
----
class Outer
{
        static int x=15;
        static class Inner
        {
                  void msg()
                        {
                              System.out.println("x value is  "+x);
                        }
        }
}
class Test14
{
      public static void main(String args[])
      {
            Outer.Inner obj=new Outer.Inner();
            obj.msg();
      }
}
--------------------------------------------------------------------------------
-----
class Outer
{
        static int x=15;
        static class Inner
        {
                  static void msg()
                        {
                              System.out.println("x value is  "+x);

                        }
        }
}
class Test15
{
      public static void main(String args[])
      {
            Outer.Inner.msg();
      }
}
--------------------------------------------------------------------------------
-----
class Outer
{
        int x=15;  //error (not possible because try to access instance
variable)
        static class Inner
        {
                  void msg()
                        {
                              System.out.println("x value is  "+x);
                        }
        }
}
class Test16
{
      public static void main(String args[])
      {
            Outer.Inner obj=new Outer.Inner();
            obj.msg();
      }
}

--------------------------------------------------------------------------------
-----
```

4) Anonymous inner class :

It is an inner class without a name and for this kind of inner class only single Object is created. (Singleton class)

A normal class can implement any number of inetrfaces but an anonymous inner class can implement only one interface at a time.

A general class can extend a class and implement any number of interfaces at the same time but an anonymous inner can extend one class or can implament one interface at a time.

```
//Anonymous inner class
class Tech
{
       public void tech()
       {
            System.out.println("Tech");
       }
}
public class Test17
{
       Tech a = new Tech()
       {
            public void tech()
              {
                 System.out.println("anonymous tech");
                }
       };
       public void dothis()
       {
             a.tech();
       }
       public static void main(String... args)
       {
             Test17 b = new Test17();
             b.dothis();
       }
}
```

```
--------------------------------------------------------------------------------
-----
@FunctionalInterface
interface Vehicle
{
      void move();
}

class Test18
{
      public static void main(String[] args)
      {
            Vehicle car = new Vehicle()
            {
                  @Override
                  public void move()
                  {
                        System.out.println("Moving with Car...");
                  }
            };
            car.move();

       Vehicle bike = new Vehicle()
            {
                  @Override
                  public void move()
                  {
                        System.out.println("Moving with Bike...");
                  }
            };
            bike.move();
      }
}
--------------------------------------------------------------------------------
-----
```

Enum in java :

Enum concept is introduced from JDK 1.5 onwards. It is a class in java which is used to represent a group of constants.

In order to create an enum we can use "enum" keyword and all the constant members of the enum should be separated by comma. Semicolon is optional.

eg:-

```
enum Color
{
    RED, BLACK, BLUE ;    //(; is optional)
}
```

All the above enum constants i.e RED, BLACK and BLUE are by default public static and final.

we can define an enum outside of the class, inside of the class and even inside the method as well.

If we declare an enum inside the class but outside of the method then it can be declared as public, default, protected, private and static.

Every enum in java by default extends java.lang.Enum class so it may implement interfaces but cann't extend any class.

Every enum is implicitly final so we can't inherit enum.

In order to get all the constant values of enum we can use values() method but this values() method is the part of enum keyword.

In order the get the order position of enum constants we can use ordinal() method. The order will start from 0(zero)

Just like a class we can also define main() method inside the enum.

We can also define constructor inside an enum.

All the enum constants are by default Object of type enum.

```
class Test1
{
      public static void main(String[] args)
      {
            enum Month
            {
                  JANUARY, FEBRUARY,MARCH;
            }
            System.out.println(Month.MARCH);
      }
}
```

```
enum Month {JANUARY,FEBRUARY,MARCH}

class Test2
{
      enum Color {RED,BLUE,BLACK}

     public static void main(String[] args)
      {
            enum Week {SUNDAY, MONDAY, TUESDAY}

            System.out.println(Month.MARCH);
            System.out.println(Color.BLACK);
            System.out.println(Week.SUNDAY);
      }
}
--------------------------------------------------------------------------------
-----
class Test3
{
      enum Color {RED,BLUE}

     public static void main(String args[])
     {
          Color c1 = Color.RED;
          Color c2 = Color.RED;

          if(c1 == c2)
          {
                System.out.println("==");
          }
          if(c1.equals(c2))
          {
                 System.out.println("equals");
          }
     }

}
--------------------------------------------------------------------------------
----
class Test4
{
      static enum Season   //private, public, protected, static
      {
      SPRING, SUMMER, WINTER, FALL;
      }
      public static void main(String[] args)
      {
            System.out.println(Season.WINTER);
      }
}
--------------------------------------------------------------------------------
---
class Hello
{
      int x = 100;
}

enum Direction extends Hello //error
{
      EAST, WEST, NORTH, SOUTH;
}

class Test5
{
      public static void main(String[] args)
      {
            System.out.println(Direction.SOUTH);
      }
}
--------------------------------------------------------------------------------
-----
//All enum are by default final so can't inherit
enum Color
{
      RED, BLUE, PINK;
}
class Test6 extends Color
```

```
{
      public static void main(String[] args)
      {
            System.out.println("Hello World!");
      }
}
--------------------------------------------------------------------------------
-----
//values() to get all the values()
class Test7
{
      static enum Season
      {
      SPRING, SUMMER, WINTER, FALL;
      }
      public static void main(String[] args)
      {
            Season s1[] = Season.values(); //enum keyword method

            for(Season x : s1)
                  System.out.println(x);
      }
}
--------------------------------------------------------------------------------
-----
//ordinal() to find out the order
class Test8
{
      static enum Season
      {
      SPRING, SUMMER, WINTER, FALL;
      }
      public static void main(String[] args)
      {
            Season s1[] = Season.values();

            for(Season x : s1)
                  System.out.println(x+" order is :"+x.ordinal());
      }
}
--------------------------------------------------------------------------------
-----
//We can take main () inside an enum
enum Test9
{
      TEST1, TEST2, TEST3 ;       //; is compulsory

      public static void main(String[] args)
      {
            System.out.println("Enum  main method");
      }
}


/* Note Inside an enum we can main method but ; is compulsory and all the enum
constants must be the first line of enum */
--------------------------------------------------------------------------------
-----
//constant must be in first line of an enum
enum Test10
{
      public static void main(String[] args)
      {
            System.out.println("Enum  main method");
      }

      TEST1, TEST2, TEST3;
}
```

```
//Writing constructor in enum
enum Season
{
      WINTER, SUMMER,SPRING;

      Season()
      {
            System.out.println("Constructor is executed....");
      }
}
class Test11
{
      public static void main(String[] args)
      {
            System.out.println(Season.WINTER);
      }
}
```

```
   //Writing constructor with message
   enum Season
      {
      SPRING("Pleasant"), SUMMER("UnPleasent"), WINTER;

      String msg;
        Season(String msg)
            {
              this.msg = msg;
            }
            Season()
            {
                  this.msg = "Cold";
            }

            public String getMessage()
            {
                  return msg;
            }
      }

class Test12
{

      public static void main(String[] args)
      {
            Season s1[] = Season.values();

            for(Season x : s1)
                  System.out.println(x+"  is :"+x.getMessage());
      }
}
```

GUI Application :

GUI stands for Graphical User Interface where as CUI stands for Character User Interface.

GUI Programs are nothing but creating the Buttons, RadioButtons, TextField, TextArea and so on

So far we have complated all CUI related programs in java but by using AWT (Abstract window toolkit) package, we can develop GUI related programs.

Why AWT Programs are called platform dependent :

The AWT programs depend upon local Operating System for visibility purpose. Let say if we create a Button using AWT then the appearence of the button would be different in different-different operating System so AWT programs are platform dependent program because It depends upon local operating system for visibility purpose.
To overcome this problem java software people introduced swing concept from JDK 1.2 onwards.

What is a container :-

It is one type of component which is responsible to contain group of components like frame, window, applet, pannel, dialog e.t.c

frame, window and applet these three are big containers where as pannel and dialog these are the small containers.

In general a small container can fit inside the big container.

```
--------------------------------------------------------------------------------
-----
import java.awt.*;
class FrameDemo1 extends Frame
{
      FrameDemo1()
      {
            setVisible(true);
            setSize(500,500);
            setTitle("My Frame");
      }
      public static void main(String [] args)
      {
            new FrameDemo1(); //nameless object OR Anonymous Object
      }
}
```

In the above program, Frame is the predefined class available in java.awt package. It works like a container.

We can't close the Frame directly by clicking on the close button. First of all we need to minimize the Frame and then press control + c from the keyboard.

```
--------------------------------------------------------------------------------
-----
import java.awt.*;
class FrameDemo2
{
    FrameDemo2()
      {
            Frame f1 = new Frame();
            f1.setVisible(true);
            f1.setSize(1000,1000);
            f1.setTitle("My Frame Demo");
      }
      public static void main(String[] args)
      {
            new FrameDemo2();
      }
```

}

There is a predefined class called Button available in java.awt package through which we can create a Button.

Eg:-
Button b1 = new Button("ClickMe");

Now to add the b1 Button into the Frame container, Frame class has provided a predefined method called add(b1), so the button would be added to the Frame.

```
import java.awt.*;
class ButtonDemo1 extends Frame
{
    ButtonDemo1()
      {
            Button b1 = new Button("Hello");
            add(b1);     //To add Button in the frame container

            Button b2 = new Button("ClickMe");
            add(b2);

            setVisible(true);
            setSize(500,500);
            setTitle("My Button Demo");
      }
      public static void main(String[] args)
      {
            new ButtonDemo1();
      }
}
```

Note :- In the above program two Buttons are created i.e Hello and ClickMe. Hello button is overlapped by ClickMe bitton.

Layout :-
----
layout :-

A layout is nothing but predefined arrangements of component. AWT supports different kinds of layout.

1) Border layout :-

In this layout the components are arranged in different direction, when we don't specify any direction then by default direction is center, that is the reason components are overlapped.

The direction can be given East, West, North, South and center

```
import java.awt.*;
class ButtonDemo2 extends Frame
{
      ButtonDemo2()
      {
            Button b1 = new Button("Hyd");
            add("North",b1);

            Button b2 = new Button("Ngp");
            add("East",b2);

            Button b3 = new Button("Jsr");
            add("West",b3);

            Button b4 = new Button("Mum");
            add("South",b4);

            setSize(800,800);
            setVisible(true);
            setTitle("Button With Direction");
      }
      public static void main(String[] args)
      {
            new ButtonDemo2();
      }
}
```

-----
2) GridLayout :-

In GridLayout the components are arranged in a row and columnn wise. we should define the number of row and coloumn as a part of GridLayout.

```
//Program on GridLayout()
import java.awt.*;
class ButtonDemo3 extends Frame
{
      ButtonDemo3()
      {
            setLayout(new GridLayout(3,3));  //3 rows and 3 columns
            Button b1 = new Button("C");
            Button b2 = new Button("PHP");
            Button b3 = new Button("ASP");
            Button b4 = new Button("Java");
            Button b5 = new Button("C++");
            Button b6 = new Button(".Net");
            Button b7 = new Button("Python");
            add(b1);
            add(b2);
            add(b3);
            add(b4);
            add(b5);
            add(b6);
            add(b7);
            setSize(500,500);
            setVisible(true);
            setTitle("My program");
      }
      public static void main(String[] args)
      {
            new ButtonDemo3();
      }
}
```

-----
3) FlowLayout :-

In FlowLayout, the components are arranged from left to right (in a row wise) and once the right part is filled, automatically it will come to the next line. This layout is most frequently used layouts.

There is a predefined class in java.awt package called Choice which creates dropdown box and another class called List which will create list box.

```
import java.awt.*;
class ListAndDropDown extends Frame
```

```
{
      ListAndDropDown()
      {
            setLayout(new FlowLayout()); //arranged the items in a row manner

            Choice c = new Choice();  // It will create the dropdown
            c.add("Core java");
            c.add("Adv java");
            c.add(".Net");
            c.add("Python");
            add(c);

            List l = new List();  //It will create the ListBox
            l.add("Sachin");
            l.add("Dhoni");
            l.add("Zaheer");
            add(l);

            setVisible(true);
            setSize(500,500);
            setTitle("My Frame");
      }
      public static void main(String[] args)
      {
         new ListAndDropDown();
      }
}
```

----
setBounds(int x-axis, int y-axis, int width, int height) :-

It is used to set the component inside the container like frame, window and so on.

It takes 4 parameter x-axis (to set the component from x-axis), y-axis(to set the position for y axis), width of the component and height of the component.

```
//setBounds(int x-axis, int y-axis, int width, int height)

import java.awt.*;
public class ButtonDemo4 extends Frame
{
   ButtonDemo4()
       {
      Button b1 = new Button("Login");
      // setting button position on Frame
      b1.setBounds(130,150,80,30);
      add(b1);

         Button b2 = new Button("SignUp");
      // setting button position on Frame
      b2.setBounds(230,150,80,30);
      add(b2);

      setSize(500,500);
      setTitle("My Frame");
      setLayout(null);
      setVisible(true);
      }
      public static void main(String args[])
            {
                ButtonDemo4  bt = new ButtonDemo4();
            }
}
```


TextField :-

It is a predefined class available in java.awt package. It is used to create the textbox and by using add method we can add the textbox in the container.

```
Eg:-  TextField f = new TextField();
```

Label :-

Label is also a Predefined class available is java.awt package. It is used to print something on the container.

```
Eg:-   Label l = new Label("Student Id");
```


```
import java.awt.*;

class TextBoxDemo1
{
   TextBoxDemo1()
      {
      Frame f = new Frame();
      Label l = new Label("Student id:");
      Button b = new Button("Submit");

      TextField t = new TextField();

      l.setBounds(20, 80, 80, 30);
      t.setBounds(20, 105, 80, 30);
      b.setBounds(100, 100, 80, 30);

      f.add(b);
      f.add(l);
      f.add(t);
      f.setSize(500,500);
      f.setTitle("Student Details");

      f.setLayout(null);
      f.setVisible(true);
      }

      public static void main(String args[])
            {
                  new TextBoxDemo1();
            }
}
```


Event Handling in java :

An event is an action performed whenever we click on Button or whenever the focus is shifted from one component to another component.

The java software prople has provided a separate package for performing the event i.e java.awt.event.

There is a predefined interface called ActionListner is available in java.awt.event package as shown below.

```
interface ActionListner
```

```
{
    void actionPerformed(ActionEvent ae);
}
--------------------------------------------------------------------------------
-----
```

Now, to perform event handling we need to implment ActionListner interface in our class and we need to override the actionPerformed() method.

We also need to register the ActionListner with the component as show below.

```
Button b1 = new Button("ClickMe");
b1.addActionListner(this);
--------------------------------------------------------------------------------
-----
import java.awt.*;
import java.awt.event.*;
class  EventByButton extends Frame implements ActionListener
{
      TextField tf;
      EventByButton()
            {
            tf=new TextField();
            tf.setBounds(60,50,170,25);
            Button b=new Button("ClickMe");
            b.setBounds(100,120,80,30);

            //register listener
            b.addActionListener(this);//passing current instance

            add(b);add(tf);
            setSize(500,500);
            setLayout(null);
            setVisible(true);
            }
            @Override
            public void actionPerformed(ActionEvent e)
            {
                  tf.setText("Hello Programmer!");
            }
            public static void main(String args[])
            {
                  new  EventByButton();
            }
}
--------------------------------------------------------------------------------
-----
javax.swing :-
--------------
```

As we know AWT is platform dependent technology so to provide platform independency java software people has provided a seperate concept called swing from jdk 1.2 onwards.

Swing is the sub package of AWT, actually it is derived from AWT that is the reason it is called extended package and we write it javax.swing. Here 'x' represents it is an extended package.

The following are the main differences between AWT and Swings

1) AWT is platform dependent on the other hand swing is platform independent.

2) AWT is heavy weight because It utilises maximum resource of the CPU where as Swing is light weight because It utilises minimum resource of the CPU.

3) AWT doesn't support pluggable where as swing is supporting pluggable.

Note :- As we know swing is the derived package of AWT hence it supports all the features of AWT, Hence all AWT classes we can also use in Swing by prefixing 'J' .

Eg:- JFrame, JButton, JRadioButton and so on.


```
//Program to create a Button the the Frame window

import javax.swing.*;
public class Simple
{
      Simple()
      {
      JFrame f=new JFrame();
      JButton b=new JButton("Click");
      b.setBounds(130,100,100, 40);
      f.add(b);
      //It used to close the Frame
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      f.setSize(500,500);
      f.setLayout(null);
      f.setVisible(true);
      }
      public static void main(String[] args)
      {
      new Simple();
      }
}
```


```
//Program to create a Button the the Frame window
import javax.swing.*;
public class Simple2 extends JFrame
{
            Simple2()
            {
            JButton b=new JButton("Click");
            b.setBounds(130,100,100, 40);

            add(b);

            setSize(400,500);
            setLayout(null);
            setVisible(true);
            setTitle("Swings Frame");
            setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            }

            public static void main(String[] args)
            {
            new Simple2();
            }
}
```


How to create an Image button in swing :

There is a predefined class called ImageIcon which takes picture as a parameter

as shown below.

```
ImageIcon i = new ImageIcon("a.jpg");
JButton button = new JButton(i);

                  OR

JButton button = new JButton(new ImageIcon("b.jpg"));


//Displaying the image on the Button
import javax.swing.*;

public class ImageButton
{
    ImageButton()
   {
            JFrame f=new JFrame();
            JButton b=new JButton(new ImageIcon("login.jpg"));
            b.setBounds(130,100,100, 40);
            f.add(b);
            f.setSize(500,500);
            f.setLayout(null);
            f.setVisible(true);
            f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      }
public static void main(String[] args)
{
      new ImageButton();
}
}
```


JRadioButton :

It is a predefined class available in javax.swing package.

It is used to create RadioButtons, through which we can select onle one item at a time.

Eg :-

```
JRadioButton r1 = new JRadioButton();
JRadioButton r2 = new JRadioButton();(Now grouping of the button is reqd)

ButtonGroup bg = new ButtonGroup();
                    bg.add(r1);
                bg.add(r2);
--------------------------------------------------------------------------------
-----

import javax.swing.*;
public class Radio
{
            JFrame f;

            Radio()
            {
            f=new JFrame();
            JRadioButton r1=new JRadioButton(" Male");
            JRadioButton r2=new JRadioButton(" FeMale");
            r1.setBounds(50,100,70,30);
            r2.setBounds(50,150,70,30);
            ButtonGroup bg=new ButtonGroup();// For selecting only one radio
            bg.add(r1);
            bg.add(r2);

            f.add(r1);
            f.add(r2);
            f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            f.setSize(300,300);
            f.setLayout(null);
            f.setVisible(true);
            }
            public static void main(String[] args)
            {
                  new Radio();
            }
}
```

-----JComboBox :

In AWT if we want to create ComboBox (Dropdown box) then we can use
predefined class called Choice, but in swing if we want to create dropdown box
or ComboBox then we should use a predefined class called JComboBox as shown
below.

```
String fruits [] = {"Apple", "Banana", "Mango","Orange"};

JComboBox cb = new JComboBox(fruits);
```

//Progran to create ComboBox or DropDown Box

```
import javax.swing.*;
public class Combo
      {
      JFrame f;
      Combo()
      {
      f=new JFrame();
      //displaying list of country to the combobox
      String country[]={"India","Aus","U.S.A","England","Newzeland"};

      JComboBox cb=new JComboBox(country);
      cb.setBounds(50, 50,90,20);
      f.add(cb);
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      f.setLayout(null);
      f.setSize(400,500);
      f.setVisible(true);
}
public static void main(String[] args)
      {
      new Combo();
     }
}
```


JTextField :

It is a predefined class available in javax.swing package.

It is used to create a textbox through which we can accept input from the
user.As we know to display some text we can Use a predefined class called JLable
just to display some message.

Eg:-

```
JLabel label = new JLabel("Roll Number");

JTextField tf = new JTextField();

import javax.swing.*;

//Program on TextField
class TextFieldExample extends JFrame
{
    TextFieldExample()
      {
            JTextField t1 = new JTextField();
            JTextField t2 = new JTextField();
            JButton b1 = new JButton("Login");
            t1.setBounds(50,100,200,30);
            t2.setBounds(50,150,200,30);
            b1.setBounds(50,200,100,30);

            add(t1);
            add(t2);
            add(b1);
            setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            setSize(300,300);
            setLayout(null);
            setVisible(true);
      }

      public static void main(String[] args)
      {
            new TextFieldExample();
      }
}
```


JTextArea :

It is a predefined class available in javax.swing package.

It is the extension of JTextField because JTextField is suitable to insert name, roll number e.t.c

On the other hand TextArea is suitable for inserting Address, Feedback form or Comments

Eg:-

```
JTextArea  jarea = new JTextArea(500,500);
```

Color is a predefined class available in java.awt package which contains predefined static variables like red, blue, black and soon.

```
//Program on TextArea

import java.awt.Color;
import javax.swing.*;

public class TArea
      {
    JTextArea area;
    JFrame f;
      TArea()
      {
      f=new JFrame();
      area=new JTextArea(500,500);
      area.setBounds(10,30,300,300);

      area.setBackground(Color.yellow); //static variable of Color class
      area.setForeground(Color.black); //static variable of Color class

      f.add(area);

      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      }
      public static void main(String[] args)
      {
            new TArea();
      }
}
```

JCheckBox

It is a predefined class available in javax.swing package.

It is used to select one or multiple items.

Among the number of check boxes available in the form we can select single or multiple check boxes.

Eg:

```
JCheckBox c1 = new JCheckeBox("Cricket");

JCheckBox c2 = new JCheckeBox("Football", true);
```

```
import javax.swing.*;
public class JCheckBoxDemo
{
            JCheckBoxDemo()
            {
             JFrame f= new JFrame("My CheckBox");

        JCheckBox c1 = new JCheckBox("Cricket");
        c1.setBounds(100,50, 150,150);

        JCheckBox c2 = new JCheckBox("Football", true);
        c2.setBounds(100,150, 150,150);

        f.add(c1);
        f.add(c2);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
            f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            }
            public static void main(String[] args)
            {
            new JCheckBoxDemo();
            }
}
```

```
--------------------------------------------------------------------------------
-----
JPasswordField :
------------------
It is a predefined class available in javax.swing package.

By using this class we can develop a textfield where the contants are not
visible so it is known as PasswordField.

Eg:
---
JPasswordField j = new JPasswordField();

Program on JPasswordField with event handling
-----------------------------------------------------
import javax.swing.*;
import java.awt.event.*;
public class PasswordFieldExample
      {
    public static void main(String[] args)
      {
    JFrame f=new JFrame("Password Field Example");
     JLabel label = new JLabel();
     label.setBounds(20,150, 200,150);

     JPasswordField value = new JPasswordField();
     value.setBounds(100,75,100,30);

     JLabel l1=new JLabel("Username:");
     l1.setBounds(20,20, 80,30);

     JLabel l2=new JLabel("Password:");
     l2.setBounds(20,75, 80,30);

     JButton b = new JButton("Login");
     b.setBounds(100,120, 80,30);

     JTextField text = new JTextField();
     text.setBounds(100,20, 100,30);

     f.add(value); f.add(l1); f.add(label); f.add(l2); f.add(b); f.add(text);
     f.setSize(500,500);
     f.setLayout(null);
     f.setVisible(true);
       f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

     b.addActionListener(new ActionListener()
            {
        public void actionPerformed(ActionEvent e)
                  {
                   String data = "Username " + text.getText();

      data += ", Password: " + new String(value.getPassword());

                   label.setText(data);
             }
             });
         }
}
--------------------------------------------------------------------------------
-----
import javax.swing.*;
import java.awt.event.*;
public class SimpleEvent implements ActionListener
```

```
{
      JButton button;
      public static void main(String [] args)
      {
            SimpleEvent se = new SimpleEvent();
            se.go();
      }
      public void go()
      {
            JFrame f = new JFrame();
            button = new JButton("Click Me");
            button.setBounds(30,50,80,30);
            button.addActionListener(this);
            f.add(button);
            f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            f.setSize(500,500);
            f.setVisible(true);
      }
      @Override
      public void actionPerformed(ActionEvent event)
      {
            button.setText("Button is Clicked");
      }
}
```


JOptionPane :-

It is the predefined class available in javax.swing package.

It provides standard dialog box such as message dialog box, confirm dialog box, input dialog box and so on.

These dialog boxes are used provide some meaningful information to the user as well as we can also take input from the user.

Eg:-

```
JOptionPane.showMessageDialog(frame, "Hello learner??");
```

Here JOptionPane is the predefined class which contains a static method showMessageDialog(), which will display message dialog box in the specified Frame.


```
import javax.swing.*;
public class OptionPaneExample
      {
      JFrame f;
      OptionPaneExample()
            {
    f=new JFrame();
    JOptionPane.showMessageDialog(f,"Hello, Welcome to dialog box of Java 
Swings.");
        }
      public static void main(String[] args)
      {
    new OptionPaneExample();
      }
}
```


```
import javax.swing.*;
public class OptionPaneExample1
```

```
      {
JFrame f;
OptionPaneExample1()
      {
    f=new JFrame();

    JOptionPane.showMessageDialog(f,"Transaction declined
","Alert",JOptionPane.WARNING_MESSAGE);

      }
public static void main(String[] args)
      {
    new OptionPaneExample1();
      }
}
--------------------------------------------------------------------------------
-----
import javax.swing.*;
public class OptionPaneExample2
      {
JFrame f;
      OptionPaneExample2()
      {
    f=new JFrame();
    String name=JOptionPane.showInputDialog(f,"Enter Name");
      }
public static void main(String[] args)
      {
    new OptionPaneExample2();
      }
}
--------------------------------------------------------------------------------
-----
import javax.swing.*;
import java.awt.event.*;
public class OptionPaneExample3 extends WindowAdapter
      {
JFrame f;
OptionPaneExample3()
      {
    f=new JFrame();
    f.addWindowListener(this);
    f.setSize(300, 300);
    f.setLayout(null);
    f.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
    f.setVisible(true);
      }
public void windowClosing(WindowEvent e)
      {
    int a=JOptionPane.showConfirmDialog(f,"Are you sure?");
      if(a==JOptionPane.YES_OPTION)
            {
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            }
      }
public static void main(String[] args)
      {
    new  OptionPaneExample3();
      }
}
--------------------------------------------------------------------------------
-----
```

Anonymous class Programs :

```
//Anonymous class
@FunctionalInterface
interface Vehicle
{
     void run();
}
public class Anonymous
{
      public static void main(String [] args)
      {
           Vehicle v = new Vehicle()
                  {
                        @Override
                        public void run()
                              {
                                    System.out.println("Running Safely");
                          }
                  };
                  v.run();
      }
}
```

```
@FunctionalInterface
interface Vehicle
{
     void run();
}
public class Anonymous1
{
      public static void main(String [] args)
      {
               Vehicle car = new Vehicle()
                  {
                        @Override
                        public void run()
                              {
                                    System.out.println("Running Car");
                          }
                  };
                  car.run();



                  Vehicle bike = new Vehicle()
                  {
                        @Override
                        public void run()
                              {
                                    System.out.println("Running Bike");
                          }
                  };
                  bike.run();
      }
}
```

It is possible to work with predefined interfaces that means the interfaces which are already defined by java software people.

```
class Anonymous2
{
  public static void main(String [] args)
   {
        Runnable r = new Runnable()
         {
                 @Override
            public void run()
               {
                       System.out.println("Running....");
               }
         };
         r.run();
   }
}
--------------------------------------------------------------------------------
-----

import java.util.*;
class Anonymous3
{
      public static void main(String[] args)
      {
            ArrayList<Integer> al = new ArrayList<Integer>();
            al.add(12);
            al.add(15);
            al.add(9);
            al.add(2);


            Comparator<Integer> cmp = new Comparator<Integer>()
            {
                  @Override
                  public int compare(Integer i1, Integer i2)
                  {
                        return - (i1-i2);  //descending
                  }
            };

            Collections.sort(al,cmp);

            for (Integer i : al)
            {
                  System.out.println(i);
            }
      }
}
--------------------------------------------------------------------------------
-----
```

Lambda expression
----

1) It is an anonymous function(Function without any name or nameless function)

2) It is used to enable functional programming in java.

3) It is used concise our code as well as we can remove boilerplate code.

4) It can be used with functional interface only.

5) If the body of the Lambda expression contains only one statement then curly braces are optional.

6) we can also remove the data type of the variable because in functional interface we have only one abtstract method so compiler can easily recognise the type of the variable.

```
//Program on Lambda expression
@FunctionalInterface
interface Moveable
{
       void move();
}
class Lambda1
{
      public static void main(String[] args)
      {
            Moveable m = () ->
            {
            System.out.println("moving here......");
            };
            m.move();
      }
}
--------------------------------------------------------------------------------
----
@FunctionalInterface
interface Calculate
{
       void add(int a, int b);
}
class Lambda2
{
      public static void main(String[] args)
      {
            Calculate c = (a,b)->
            {
                  System.out.println("Sum is :"+(a+b));
            };
            c.add(12,12);
      }
}
--------------------------------------------------------------------------------
-----
@FunctionalInterface
interface Length
{
       int getLength(String str);
}
class Lambda3
{
      public static void main(String[] args)
      {
            Length l = (str)-> { return str.length(); };

            System.out.print("Length is :"+l.getLength("India"));
      }
}
```

Note :- We cannot remove curly braces if the body of Lambda expression contains return statement.

```
--------------------------------------------------------------------------------
----
@FunctionalInterface
interface Length
{
       int getLength(String str);
}

class Lambda4
{
      public static void main(String[] args)
      {
            Length l = (str)-> str.length();

            System.out.print("Length is :"+l.getLength("Ravi"));
      }
}
```


Working with predefined functional interfaces :

As a Java 8 version, java software people has provided some predefined functional interfaces.

In order to help the programmer or developer to write concise code in day to day programming so we can use Lambda expression with the following predefined functional interfaces

Predicate
Consumer

All these predefined functional interfaces has been defined inside the package java.util.function

Predicate :

It is a predefined functional interface inside java.util.function package.

It represents a predicate i.e boolean based value function with one argumnet as shown below

```
@FunctionalInterface
public interface Predicate<T>
{
  boolean test(T t);
}
```


```
/*@FunctionalInterface
public interface Predicate<T>
{
  boolean test(T t);
}
*/

import java.util.function.*;

class Lambda5
{
      public static void main(String[] args)
      {
       Predicate<Integer> p = i -> i%2 ==0;
            System.out.println(p.test(10));
            System.out.println(p.test(15));
      }
}
```


Consumer :

It is a predefined functional interface inside java.util.function package.

It contains a predefined abstract method accept(T t) which takes T (type parameter) as a parameter and returns nothing.

```
public interface Consumer<T>
{
     void accept(T t);
}

/*
@FunctionalInterface
public interface Consumer<T>
{
  void accept(T x);
}
*/

import java.util.function.*;
class  Lambda6
{
      public static void main(String[] args)
      {
            Consumer<String> printString = x -> System.out.println(x);
        printString.accept("Ravi");

            Consumer<Integer> printInteger = x -> System.out.println(x);
        printInteger.accept(15);

      }
}
```

----
forEach() :

As we know we have a for-each loop which is also known as enhanced loop to fetch the values from the collection.

As a part of java 8, java software people has provided forEach() to iterate the elements.

This method is defined inside iterable interface in the collection framework as well as in the Stream interface available in java.util.function.Stream.

This forEach() method take parameter as a Consumer interface so we can implement Lambda expression inside forEach() method.


```
//After collection, forEach()
import java.util.function.*;
import java.util.*;

class ConsumerImpl implements Consumer<Integer>
{
    public void accept(Integer i)
      {
          System.out.println(i);
      }

}
class Lambda7
{
      public static void main(String[] args)
      {
            List<Integer> list = Arrays.asList(1,2,3,4,5);
            Consumer<Integer> con = new ConsumerImpl();
            list.forEach(con);
      }
}



--------------------------------------------------------------------------------
----
import java.util.*;
public class ForEachDemo1
{
      public static void main(String[] args)
      {
            List<String> player = Arrays.asList("Virat","Rohit","Rahul");

            player.forEach(s -> System.out.println(s));  //accept(T t);

      }

}
--------------------------------------------------------------------------------
-----
import java.util.function.*;
import java.util.*;

class Lambda8
{
      public static void main(String[] args)
      {
            List<Integer> list = Arrays.asList(1,2,3,4,5);
            Consumer<Integer> con = i -> System.out.println(i);
            list.forEach(con);
      }
}
--------------------------------------------------------------------------------
-----
import java.util.List;
import java.util.function.Consumer;
import java.util.ArrayList;

public class ForEachDemo2
{
      public static void main(String[] args)
      {
            List<String> nameList = new ArrayList<String>();
            nameList.add("Virat");
            nameList.add("Rahul");
            nameList.add("Rohit");
            nameList.add("Bumrah");

            Consumer<String> makeUpperCase = new Consumer<String>()
            {
                  @Override
                  public void accept(String t)
                  {
                        System.out.println(t.toUpperCase());

                  }
            };
            nameList.forEach(makeUpperCase);
      }
}
```

```
--------------------------------------------------------------------------------
----
import java.util.HashMap;
import java.util.Map;

public class ForEachDemo3
{
      public static void main(String[] args)
      {
            Map<String,String> map = new HashMap<String,String>();
            map.put("A","Ravi");
            map.put("B", "Rahul");
            map.put("C", "Mona");
            map.put("D", "Elina");

            map.forEach((k,v)-> System.out.println("key = "+ k +" value = "+v));
      }
}
--------------------------------------------------------------------------------
-----
```

Method Reference :

It is introduced in java 8.

It is an improvement over Lambda expression where we can call a method as a reference as shown below.

```
import java.util.Arrays;
import java.util.List;

public class MethodReference {

      public static void main(String[] args)
      {
            List<String> player = Arrays.asList("Virat","Rohit","Rahul");

            player.forEach(System.out::println); //Method reference

            player.forEach(s -> System.out.println(s)); //Lambda expression
      }
}

--------------------------------------------------------------------------------
-----
```

Working with Stream interface :

Stream interface is introduced from java 8.

This interface is available in java.util.stream.

The main purpose of Stream interface to process the Collection Objects that is the reason it is also known as Collection Stream.

Stream does not store data, we can use Stream to filter, collect and print the data which are coming from a particular source.

```
//Converting into Stream
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Stream;

public class StreamDemo1
{
      public static void main(String[] args)
      {
            List<String> fruit = new ArrayList<String>();
            fruit.add("Orange");
            fruit.add("Mango");
            fruit.add("Banana");
            fruit.add("Apple");

            Stream<String> stream = fruit.stream();
            stream.forEach(p->System.out.println(p));

      }

}
--------------------------------------------------------------------------------
----
import java.util.stream.Stream;

public class StreamDemo2
{
      public static void main(String[] args)
      {
            Stream<Integer> stream = Stream.of(1,2,3,4,5,6,7,8,9);
            stream.forEach(p-> System.out.println(p));
            System.out.println("...............");
            Stream<Integer> stream1 = Stream.of(new Integer[] {12,67,34,90});
            stream1.forEach(p-> System.out.println(p));
      }

}
--------------------------------------------------------------------------------
----
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class StreamEvenCollector
{
      public static void main(String[] args)
      {
            ArrayList<Integer> al = new ArrayList<Integer>();

            for(int i=1; i<=10; i++)
            {
                  al.add(i);
            }
        System.out.println(al);

        //Without Stream concept to fetch all the even number from the al
        ArrayList<Integer> evenList = new ArrayList<Integer>();

        for(Integer i : al)
        {
            if(i%2==0)
                  evenList.add(i);
        }
        System.out.println(evenList);

        //With Stream concept to fetch all even number

        Stream<Integer> stream = al.stream();
        List<Integer> evenStream =
        stream.filter(i->i%2==0).collect(Collectors.toList());
        System.out.println(evenStream);
      }

}
```

