PROGRAMMING IN JAVA

UNIT-2

Object in Java

An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.

Characteristics:

- **State:** represents the data (value) of an object.
- **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

different ways to create an object in Java

There are many ways to create an object in java. They are:

1. By new keyword
2. By newInstance() method
3. By clone() method
4. By deserialization
5. By factory method etc.

initialize object

There are 3 ways to initialize object in Java.

1. By reference variable
2. By method
3. By constructor

Anonymous object:

Anonymous simply means nameless. An object which has no reference is known as an anonymous object. It can be used at the time of object creation only.

If you have to use an object only once, an anonymous object is a good approach.

For example:

```
new Calculation();//anonymous object
```

Class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

- Fields
- Methods
- Constructors
- Blocks
- Nested class and interface

Syntax to declare a class:

```
class <class_name>{
   field;
   method;
}
```


Instance variable:

A variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

Java Reference Variable

Reference variables are used to refer to an object. They are declared with a specific type which cannot be changed.

Types of reference variables:

1. Static Variable
2. Instance Variable
3. Method Parameter
4. Local Variable

Method:

In Java, a method is like a function which is used to expose the behavior of an object.

Advantage of Method

- Code Reusability
- Code Optimization

Method overloading:

If two or more method in a class has same name but different parameters, it is known as method overloading. Overloading always occur in the same class (unlike method overriding). Method overloading is one of the ways through which java supports polymorphism.

Advantage of method overloading

- Method overloading increases the readability of the program.

Different ways of Method overloading

There are three different ways of method overloading

1. Method overloading by changing data type of Arguments
2. Method overloading by changing no. of argument.
3. Method overloading by changing sequence of data type of arguments.

Example:

```
class Addition
{
Void sum(int a, int b)
{
System.out.println(a+b);
}
void sum(int a, int b, int c)
{
System.out.println(a+b+c);
}
public static void main(String args[])
{
Addition obj=new Addition();
```

```
obj.sum(10, 20);
obj.sum(10, 20, 30);
}
}
```

Output: 30
60

Method Overriding:

Whenever same method name is existing in both base class and derived class with same types of parameters or same order of parameters is known as method Overriding.

Note: Without Inheritance method overriding is not possible.

Advantage of Java Method Overriding

- Method Overriding is used to provide specific implementation of a method that is already provided by its super class.
- Method Overriding is used for Runtime Polymorphism

Example:

```
class Walking
{
void walk()
{
System.out.println("Man walking fastly");
}
}
class Man extends walking
{
void walk()
{
System.out.println("Man walking slowly");
}
```

```
}
class OverridingDemo
{
public static void main(String args[])
{
Man obj = new Man();
obj.walk();
}
        }
```

Output: Man walking slowly

Constructor:

Constructor in java is a special type of method that is used to initialize the object. Java constructor is invoked at the time of object creation. It constructs the values i.e., provides data for the object that is why it is known as constructor.

Rules for creating java constructor

There are basically two rules defined for the constructor.

1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

Types of java constructors:

There are two types of constructors:

1. Default Constructor:

A constructor that have no parameter is known as default constructor.

Syntax:

```
    <class_name>(){
}
```

Example:

```
class Bike
{
Bike()
```

```
{
System.out.println("Bike is created");
}
public static void main(String args[])
{
Bike b=new Bike();
}
}
```

Output: Bike is created

2. Parameterized constructor:

A constructor that has parameters is known as parameterized constructor.

Syntax:

```
<class_name>(parameters){}
```

Example:

```
class Student
{
   int id;
   String name;
   Student(int i,String n)
  {
   id = i;
   name = n;
   }
   void display()
{
System.out.println(id+" "+name);
}
   public static void main(String args[]){
   Student s1 = new Student(111,"Karan");
```

```
    Student s2 = new Student(222,"Aryan");
    s1.display();
    s2.display();
   }
}
```

Output: 111 Karan
222 Aryan

Constructor Overloading:

Constructor overloading is a concept of having more than one constructor with different parameters list, in such a way so that each constructor performs a different task.

Example:

```
class Student{
   int id;
   String name;
   int age;
   Student(int i,String n)
   {
   id = i;
   name = n;
   }
   Student(int i,String n,int a)
  {
   id = i;
   name = n;
   age=a;
   }
   void display()
  {
   System.out.println(id+" "+name+" "+age);
  }
```

```
    public static void main(String args[])
   {
    Student s1 = new Student(111,"Karan");
    Student s2 = new Student(222,"Aryan",25);
    s1.display();
    s2.display();
   }
}
```

Output: 111 Karan 0

222 Aryan 25

Inheritance:

Inheritance in java is a mechanism in which one class acquires the properties (data members) and functionalities(methods) of another class.

Syntax :

```
class derived-class extends base-class
{
 //methods and fields
}
```


Types of inheritance in java

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

1. Single Inheritance:

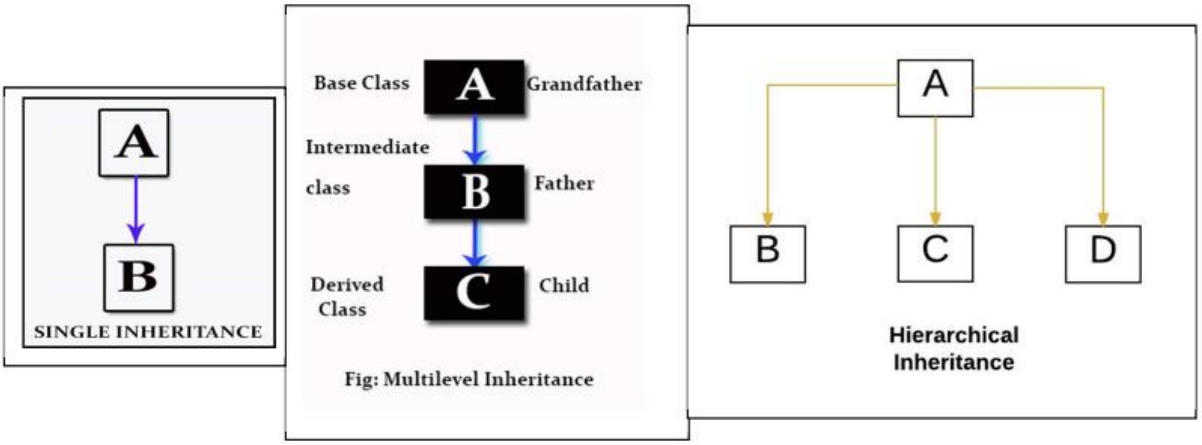
In single inheritance, subclasses inherit the features of one superclass.

2. Multilevel Inheritance:

In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class.

3. Hierarchical Inheritance:

In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one sub class.



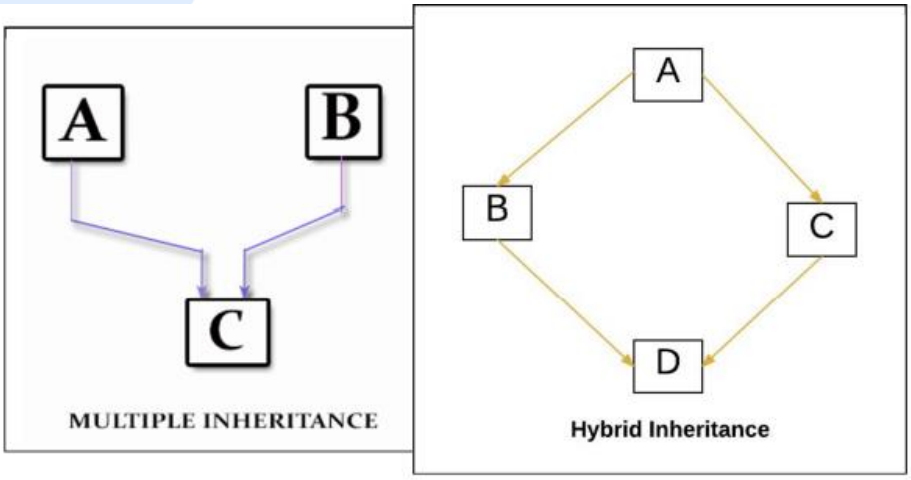
java programming, multiple and hybrid inheritance is supported through interface only.

4. Multiple Inheritance (Through Interfaces):

In Multiple inheritance, one class can have more than one superclass and inherit features from all parent classes. Please note that Java does not support multiple inheritance with classes. In java, we can achieve multiple inheritance only through Interfaces.

5. Hybrid Inheritance (Through Interfaces):

It is a mix of two or more of the above types of inheritance. Since java doesn't support multiple inheritance with classes, the hybrid inheritance is also not possible with classes. In java, we can achieve hybrid inheritance only through Interfaces.



Static keyword:

The **static keyword** in Java is used for memory management mainly. We can apply static keyword with variables, methods, blocks and nested classes. The static keyword belongs to the class than an instance of the class.

Static variable:

If you declare any variable as static, it is known as a static variable.

The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.

The static variable gets memory only once in the class area at the time of class loading.

It makes your program memory efficient (i.e., it saves memory).

Static method:

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

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

Final Keyword:

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

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only.

Abstract Keyword:

The abstract keyword is used to achieve abstraction in Java. It is a non-access modifier which is used to create abstract class and method.

The role of an abstract class is to contain abstract methods. However, it may also contain non-abstract methods. The method which is declared with abstract keyword and doesn't have any implementation is known as an abstract method.

Syntax:-

```
abstract class Employee

{

abstract void work();

}
```

Note - We cannot declare abstract methods in non-abstract class.

Access Specifiers:

Access modifiers (or access specifiers) are keywords in object-oriented languages that set the accessibility of classes, methods, and other members.

There are four types of access modifiers available in java:

1. **Default (No keyword required):** When we do not mention any access modifier, it is called default access modifier. The default modifier is accessible only within package.
2. **Private:** Private Data members and methods are only accessible within the class. Class and Interface cannot be declared as private. If a class has private constructor, then you cannot create the object of that class from outside of the class.
3. **Protected:** Protected data member and method are only accessible by the classes of the same package and the subclasses present in any package.
4. **Public:** The members, methods and classes that are declared public can be accessed from anywhere. Public are also called universal access modifiers.

Packages in Java:

Packages in Java are groups of similar types of classes, interface and sub packages.

Syntax:-

```
package;
```


Advantage of Java Package:

1. **Reusability:** The classes contained in the packages of another program can be easily reused.
2. **Better Organization:** Java package is used to categorize the classes and interfaces so that they can be easily maintained.
3. **Protection:** Java package provides access protection.
4. **Name Conflicts:** Java package removes naming collision.

Types of Packages:

1. **Built-in Package:** Existing Java package for example java.lang, java.util etc.
2. **User-defined-package:** Java package created by user to categorize their project's classes and interface.

Steps to create a Java package:

1. Create a package name.
2. Pick up a base directory.
3. Make a subdirectory from the base directory that matches your package name.
4. Place your source files into the package subdirectory.
5. Use the package statement in each source file.
6. Compile your source files from the base directory.
7. Run your program from the base directory.

Example of Java packages

//save as FirstProgram.java

```
package learnjava;
public class FirstProgram{
public static void main(String args[]) {
System.out.println("Welcome to package");
}
}
```

Output: Welcome to package

Access package from another package:

Import keyword:

import keyword is used to import built-in and user-defined packages into your java source file so that your class can refer to a class that is in another package by directly using its name.

There are 3 different ways to refer to any class that is present in a different package:

1. Using fully qualified name

If you use fully qualified name to import any class into your program, then only that particular class of the package will be accessible in your program, there is no need to use the import statement.

2. Using packagename.classname

To import only the class/classes you want to use.

3. Using packagename.*

To import all the classes from a particular package but the classes and interface inside the subpackages will not be available for use.

Subpackage:

A package created inside another package is known as a subpackage. When we import a package, subpackages are not imported by default. They have to be imported explicitly. In the following statement:

import java.util.*;

util is a subpackage created inside java package.

Example of Subpackage:

```
package com.javatpoint.core;

class Simple{

public static void main(String args[]){

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

```
}
}
```

Output: Hello subpackage

Abstract class:

A class that is declared with abstract keyword, is known as abstract class in java.

It can have abstract and non-abstract methods.

It cannot be instantiated. Abstract classes can have Constructors, Member variables and Normal methods.

When you extend Abstract class with abstract method, you must define the abstract method in the child class, or make the child class abstract.

Syntax:

abstract class class_name { }

Abstract method:

A method that is declared as abstract and does not have implementation is known as abstract method.

Syntax:

abstract return_type function_name ();

Interface:

- Interface is a pure abstract class.
- They are syntactically similar to classes, but you cannot create instance of an Interface and their methods are declared without any body.

Properties:

- An interface does not contain any constructors.
- The methods declared in interface are by default abstract (only method signature, no body)
- The variables declared in an interface are public, static & final by default.

Use of interface:

- Interface is used to achieve complete abstraction in Java.
- By using Interface, you can achieve multiple inheritance in java.
- It can be used to achieve loose coupling.

Syntax:

interface interface_name { }

Declaring Interfaces

The interface keyword is used to declare an interface.

Example of Interface

```
interface Person
{
datatype variablename=value;
//Any number of final, static fields
returntype methodname(list of parameters or no parameters)
//Any number of abstract method declarations
}
```

Implementing Interfaces:

A class uses the implements keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

Example:


```
interface Person
{
void run(); // abstract method
}
class A implements Person
{
public void run()
{
System.out.println("Run fast");
}
public static void main(String args[])
{
A obj = new A();
obj.run();
```

```
}
}
```

Output: Run fast

String Handling:

The basic aim of String Handling concept is storing the string data in the main memory (RAM), manipulating the data of the String, retrieving the part of the String etc. String Handling provides a lot of concepts that can be performed on a string such as concatenation of string, comparison of string, find sub string etc.

String: String is a sequence of characters enclosed within double quotes (" ") is known as String.

Example: "Java Programming".

How to create String object:

There are two ways to create String object:

1. By string literal: Java String literal is created by using double quotes.

For Example: String s="welcome";

2. By new keyword: In such case, JVM will create a new string object.

For Example: String s=new String("Welcome");

In java programming to store the character data we have a fundamental datatype called char. Similarly, to store the string data and to perform various operations on String data, we have three predefined classes they are:

String class: It is a predefined class in java.lang package can be used to handle the String. String class is immutable that means whose content cannot be changed at the time of execution of program.String class object is immutable that means when we create an object of String class it never changes in the existing object.

Example:

```
class StringHandling
{
public static void main(String arg[])
{
String s=new String("java"); s.concat("software"); System.out.println(s);
}
}
```

Output: java

Methods of String class

length(): This method is used to get the number of character of any string.
charAt(): This method is used to get the character at a given index value.
toUpperCase(): This method is use to convert lower case string into upper case.
toLowerCase(): This method is used to convert lower case string into upper case.
concat(): This method is used to combined two string.
equals(): This method is used to compare two strings, It return true if strings are same otherwise return false. It is case sensitive method.

equalsIgnoreCase(): This method is case insensitive method, It return true if the contents of both strings are same otherwise false.

compareTo(): This method is used to compare two strings by taking unicode values, It return 0 if the string are same otherwise return +ve or -ve integer values.

compareToIgnoreCase(): This method is case insensitive method, which is used to compare two strings similar to compareTo().

startsWith(): This method return true if string is start with given another string, otherwise it returns false.

endsWith(): This method return true if string is end with given another string, otherwise it returns false.

subString(): This method is used to get the part of given string.
indexOf(): This method is used find the index value of given string. It always gives starting index value of first occurrence of string.

lastIndexOf(): This method used to return the starting index value of last occurence of the given string.

trim(): This method remove space which are available before starting of string and after ending of string.

split(): This method is used to divide the given string into number of parts based on delimiter (special symbols like @ space ,).

replace(): This method is used to return a duplicate string by replacing old character with new character.

StringBuffer:

It is a predefined class in java.lang package can be used to handle the String, whose object is mutable that means content can be modify. StringBuffer class is working with thread safe mechanism that means multiple thread are not allowed simultaneously to perform operation

of StringBuffer. StringBuffer class object is mutable that means when we create an object of StringBuilder class it can be change.

Example:

```
class StringHandling
{
public static void main(String arg[])
{
StringBuffer sb=new StringBuffer("java");
sb.append("software");
System.out.println(sb);
}
}
```

Output: javasoftware

Methods of StringBuffer class

reverse(): This method is used to reverse the given string and also the new value is replaced by the old string.

insert(): This method is used to insert either string or character or integer or real constant or boolean value at a specific index value of given string.

append(): This method is used to add the new string at the end of original string.

replace(): This method is used to replace any old string with new string based on index value.

deleteCharAt(): This method is used to delete a character at given index value.

delete(): This method is used to delete string form given string based on index value.

toString(): This method is used to convert mutable string value into immutable string.

StringBuilder:

It is a predefined class in java.lang package can be used to handle the String. StringBuilder class is almost similar to to StringBuffer class. It is also a mutable object. The main difference StringBuffer and StringBuilder class is StringBuffer is thread safe that means only one threads allowed at a time to work on the String whereas StringBuilder is not thread safe that means multiple threads can work on same String value.