Unit - II (Concept of class and Inheritance)

① Introduction of Class

- A class is a set of object which shares common characterstics and common properties. It is a user - defined blueprint or prototype from which objects are created.

- Properties of class
  - ↳ A class is not real world entity.
  - ↳ Class does not occupy memory
  - ↳ Class is a group of variables of different data type

- A class can contain
  - ↳ Data member
  - ↳ Method
  - ↳ Constructor
  - ↳ Nested class
  - ↳ Interface

- Class declaration in Java

```
Class <Class_name>
{
  data member ;
  method ;
  Constructor ;
  nested class ;
  interface ;
}
```

Object

- An object is a basic unit of object-oriented programming and represent real-life entities. Object are the instance of class that are created to use the attribute and method of class

- An object consist of
  - ↳ state
  - ↳ behaviour
  - ↳ Identity

- Declaring an object
  - ↳ Using new keyword
  - ↳ Using class method
  - ↳ using clone() method
  - ↳ Deserilization

Difference between Class And Object

Class
- Class is a blueprint of object
- No memory is allocated when class is declared
- A class is group of similar object
- Class is logical entity
- A class can be declared once
- Ex - Car

Object
- An object is instance of class
- Memory is allocated as soon as object is created
- An object is real-world entity
- An object is a physical entity
- An object can be created many times as per requirement
- Ex - Class Car can be BMW, Mercedes, ferrari etc.

Method

- A method is a block of code which only runs when it is called.
- You can pass data, known as parameter into a method.
- Methods are used to perform certain action, and they are also known as function.
- A method must be declared within a class, followed by parentheses ( ).
- Java provide some pre-defined method such as System.out.print( ).

Sub Class and Super Class

- A subclass is a class derived from the superclass. It inherits the properties of the superclass and also contain the attributes of its own.
- A subclass is known as derived class, child class.
- A superclass is the class from which many subclass can be created. The subclass inherits the properties of superclass.
- A superclass is known as parent class or base class.

Method Overloading And Overriding

Method Overloading

- Method overloading is compile-time polymorphism
- Helps to increase the reliability of program
- It occurs within a class
- It may require or may not require inheritance
- Method must have same name and different signature
- Return type can or cannot be the same
- Static binding is being used for overloaded methods
- Private and final methods can be overloaded
- Argument list should be different

Method Overriding

- Method overriding is run-time polymorphism
- Used to grant the specific implementation of method
- It is performed in two classes
- It always needs inheritance
- Methods must have same name and same signature
- Return type must be the same
- Dynamic binding is being used for overriding methods
- Private and final methods can't be overriden
- Argument list should be the same

Abstract Class

- Abstract class is a class that can not be instantiated by itself, it need to be subclassed by another class to use its properties.
- An abstract class is declared using the "abstract" keyword in its class definition
- We can have abstract class without any abstract method
- An instance of an abstract class can not be created.

Syntax:

```
abstract class
{
    int color;
    // An abstract function
    abstract void draw();
}
```

#A Constructor

A constructor is a block of codes similar to method. It is called when an instance of class is created. At the time of calling the constructor, memory is allocated in the memory.

- Constructor are called only once at a time of Object creation while methods can be called any number of times.
- Constructor are used to assign values to the class variable at the time of object creation.

Ex :-

```
Public class Main {
    int x;

    public Main () {
        x = 5;
    }
    public static void main (String [] args) {
        Main myObj = new Main ();
        System.out.print (myObject);
    }
}
```

Output - 5.

Types of Constructor

① Default Constructor

A constructor that has no parameter is known as default constructor. A default constructor is invisible and if we write a constructor with no argument then compiler does not create a default constructor. The default constructor can be implicit or explicit. If no constructor is defined in a class java compiler automatically provide a default constructor.

② Parametrized Constructor

A constructor that has parameter is known as parametrized constructor. If we want to initialize the class with our own values, then we use a parametrized constructor.

③ Copy Constructor

Unlike other constructor copy constructor is passed with another object which copies the data available from the passed object to the newly created object.

Inheritance

- Inheritance is an important pillar of object-oriented programming. It is a mechanism by which one class inherit the features (fields and method) of another class.
- Inheritance means creating new class based on existing ones. A class that inherits from another class can reuse the methods and fields of that class.
- The "extend" keyword is used for inheritance. using the extend keyword indicate you are derived from an existing class.

Syntax

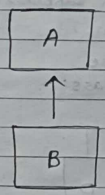
```
Class derived class extend Baseclass
{
  // methods and fields
}
```

Types of Inheritance

- ↳ Single Inheritance
- ↳ Multilevel Inheritance
- ↳ Hierarchical Inheritance
- ↳ Multiple Inheritance
- ↳ Hybrid Inheritance

① Single Inheritance

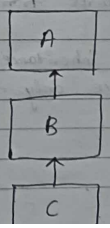
In single Inheritance, a sub-class is derived from only one super class. It inherits the properties and behaviour of single-parent class. It is also known as simple inheritance.



'A' is a parent class and 'B' is a child class. The class 'B' inherits all the properties of class 'A'.

② Multilevel Inheritance

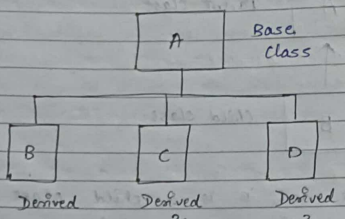
In multilevel inheritance, a derived class will be inheriting a base class and as well as the derived class also act as base class for other classes.



class 'A' serve as a base class for derived class 'B', which in turn serve as a base class for derived class 'C'.

③ Hierarchical Inheritance

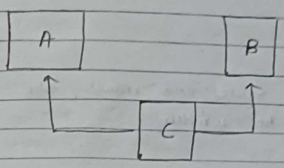
In hierarchical inheritance, one class serves as a superclass (base class) for more than one subclass



Class 'A' serves a base class for the derived class B, C and D.

④ Multiple Inheritance

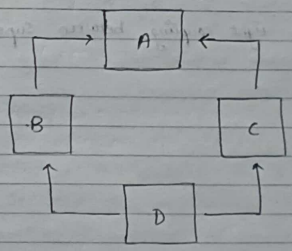
- In Multiple Inheritance, one class can have more than one superclass and inherit features from all parent class
- Java does not support multiple inheritance with class, we can achieve multiple inheritance only through Interface.



Class 'C' is derived from interface 'A' and 'B'

⑤ Hybrid Inheritance

It is a mix of two or more types of inheritance. Since Java doesnot support multiple inheritance with classes, so we can achieve hybrid inheritance only through interface if we want to involve multiple inheritance to implement hybrid inheritance



Advantages of Inheritance

- Inheritance allows for code reuse and reduce the amount of code that needs to be written.
- Inheritance allows for creation of abstract class.
- Inheritance allows for creation of class hierarchy.
- Inheritance allows for polymorphism.

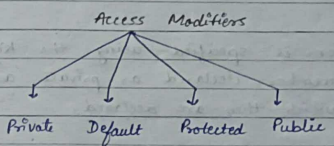
Disadvantages of Inheritance

- Inheritance can make the code more complex and harder to understand.
- Inheritance create a tight coupling between Superclass and Subclass.

Access Specifiers / Access Modifiers

- Access modifiers helps to restrict the scope of a class, constructor, variable, method or data member. It provide security, accessibility to user depending upon the access modifiers used with the element.
- It play a crucial role in encapsulation, which is one of the fundamental principal of object-oriented programming.

Types of access modifiers



① Default Access Modifiers

- When no access modifiers is specified for a class, method, or data member, it is said to be having the default access modifiers by default.
- Default access modifiers are accessible within the same package.

Ex :-

```
Package p1 ;

Class Modifiers
  {
    Void display ()
     {
       System.out.print ("Hello world");
     }
  }
```

② Private Access Modifiers

The private access modifier is specified using the keyword private. The methods or data member declared as private are accessible within the class in which they are declared

- Private means "only visible within the enclosing class"

Ex :-

```
Package p1 ;

Class A {
    private void display ()
    {
      System.out.print ( " Private access modifiers" );
    }
}
```

③ Protected Access Modifiers

The protected access modifiers is specified using the keyword protected. The methods or data member declared as protected are accessible within the same package or subclass in different packages

Ex :-

```
package p1 ;

Public class A {
      protected void display ()
      {
        System.out.print {" Protected access modifiers")
      }
}
```

④ Public access modifiers

The public access modifier is specified using the keyword Public. The methods or data member that are declared as public are accessible from every where in the program.

- There is no ~~scope~~ restriction on the scope of public data member.

Ex:-

```
Package p1;

Public class A {
    public void display()
    {
      System.out.print("Public access modifiers")
    }
}
```

* Algorithm to use Access Modifiers

↳ Define a class
↳ Define instance variables
↳ set an access modifier
  - use private
  - use protected
  - use public
↳ Use getter and setter methods

(#) Using Final with Inheritance

- Final classes

When a class is declared as final it cannot be subclassed. This means no other class can inherit from it

```
Public final class My class {
        //....
    }
```

- Final Method

When a method is declared as final, it cannot be overriden in any sub class

```
Public class Parent class {
    Public final void My Method ( ) {
            //.....
        }
    }
    Public class child class {
            // cannot override my method ( ) here
        }
```

- Variables

↳ Instance Variable

When an instance variable is declared as final, its value must be assigned during declaration in a constructor

```
Public class My class {
    private final int x = 10;
      //.....
  }
```

↳ class Variable

When a class variable is declared as final, it must assinged during declaration.

```
Public class My Class {
    Public static final double Pi = 3.14159;
    //...
}
```