Ch-3 - Inheritance and Polymorphism

* Explain Inheritance with its types.

=> ~~In inheritance, Super class~~

In inheritance, subclass can acquire all the property of the Super class.

- Super Class : Super class is called Parenst class.

Super class is the class in which a subclass inherits the Property.

- Sub Class : Sub class is called child class.

Sub class is inherits all the property of Super class.

For inherits Super class property we have to use extends keyword.

Syntax:

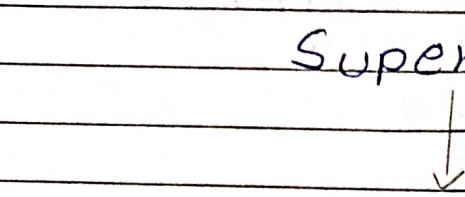
```
class Subclass extends Superclass
      name                name
{
    // body
}
```

There are Five types of inheritances.

1) Single
2) Multilevel
3) Hierarchical
4) Multiple
5) Hybrid

Java does not multiple and Hybrid Inheritances using extend keyword.

1 Single Inheritances :



In Single inheritances Super class inherit Sub class.

Ex.
```
class java
{
  Void display()
  {
    System.out.println("Hello");
  }
}
class java1 extends java
{
  void display1()
  {
    System.out.println("Khushi");
  }
  public static void main
          (String args[])
  {
    java1 obj = new java1();

    obj.display();
    Obj.display1();
  }
}
```

2 Multilevel Inheritance:

Super class
↓
Sub class 2
↓
Sub class 1

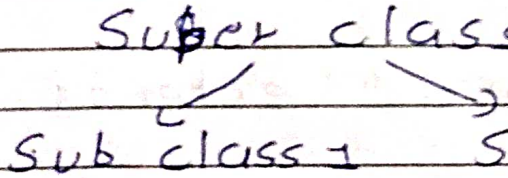
Here, Sub class 1 inherit Sub class property and Sub class inherit Super class property.

Ex.
```
class java0
{
  void display()
  {
    System.out.println("Khushi");
  }
}
class java1 extends java0
{
   void disp()
   {
     System.out.println("Gandhi");
   }
}
class java2 extends java1
{
   void dis()
   {
```

```
        System.out.println("KG");
    }
}
class main
{
    public static void main
            (String args[])
    {
        java2 obj = new java2();

            obj.dis();
            obj.disp();
            obj.display();
    }
}
```

3 Hierachical Inheritance:



Here, Sub class 1 and Sub class 2 inherit Super class property.

Ex.
```
class java
{
    void display()
    {
        System.out.println("Khushi");
    }
```

```
}

class java1 extends java
{
  void disp()
  {
    System.out.println("Gandhi");
  }
}
class java2 extends java
{
  void dis()
  {
    System.out.println("KG");
  }
}
class main()
{
  public static void main(String,
          args[])
  {
    java1 obj = new java1();

      obj.disp();
      obj.display();

  }
}
```

* Explain Constructor and Method Overridding.

=> Constructor Overridding :

* Explain Method Overridding.

=> Method Overridding is a feature that allows us to have a same ~~fune~~ method name in super class and sub class.

It allows to use same name method in super and sub class.

In Overridding, Super class and sub class have method name, return types are same.

Ex.
```
class java
{
   void disp()
   {
      System.out.println("Khushi");
   }
}
```

```
class javal extends java
{
   void dispC)
   {
     System.out.println("Gandhi");
   }
  public static void main(String,
          args[])
  {
     javal obj = new javal();

        obj.dispC);

   }
}
```

* Explain Polymorphism in java.

=> In Polymorphism we can perform a single action in different ways.

Polymorphism means more than one form.

There are two types of Polymorphism.

1) Complie time Polymorphism

2) Run time Polymorphism.

1 Complie time Polymorphism:

Using Method Overloading we can achieve complie time polymorphism.

Method Overloading is used in this polymorphism.

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

```
        obj.sum(5,6);
        obj.sum(5,6,7);
    }
}
```

2 Run time Polymorphism:

Run time Polymorphism can achieve using Overriding.

Using Method Overriding, we can achieve this polymorphism.

Ex.

```
class java
{
    void disp()
    {
        System.out.println("Khushi");
    }
}
class java1 extends java
{
    void disp()
    {
        System.out.println("Gandhi");
    }
}
class main
{
```

```
public static void main
        (String. args[])
{
    java1 obj = new java1();
    obj.disp();
  }
}
```

* Explain Binding in Java.

=> There are two types of Binding.

1) Static Binding
2) Dynamic Binding

1 Static Binding:

Static Binding is called Early binding.

When type of object determined at complie time, this is called static Binding.

Ex.
```
class java
{
  void disp()
  {
```

```
    System.out.println("Khushi");
  }
  public static void main
          (String, args[])
  {
     java obj = new javac();
        obj.disp();
  }
}
```

2 Dynamic Binding :

Dynamic Binding is also called Late Binding.

When type of the object is determined at run time, this is called Dynamic Binding.

Ex.
```
class java
{
  void disp()
  {
    System.out.println("Khushi");
  }
}
class java1 extends java
{
  void disp()
  {
```

```
    System.out.println("Gandhi");
}
public static void main
        (String, args[])
{
    java obj = new java1();
    obj.javadisp();
}
}
```

* Explain Casting Object and Instance of operator.

=> Casting Object:

Using Casting Object, we can create super class object and call the sub class as a object of super class.

Syntax :

```
super Object = new Sub ;
 class name          class
```

Ex, class java
{
  void disp()
  {
    System.out.println("Khushi");
  }
}
class java1 extends java
{
  void disp()
  {
    System.out.println("Gandhi");
  }
}
class main
{
  public static void main(String
        args[])
  {
    java obj = new java1();

    obj.disp();
  }
}

=> Instanceof Operator :

Instanceof is used to check whether the object is an

instance of the specified class or subclass.

Ex.
```
class java
{
    public static void main
            (String, args[])
    {
        java obj = new java();

        System.out.println(obj
            instanceof java);
    }
}
```

* Explain Abstract class.

=> A class which is declared using adstract Keyword this class is called Abstract class

A method which is declared using adstract ~~mc~~ Keyword this method is called Abstract method.

Syntax:

```
abstract class Class name;
{
    abstract method();
}
```

abstract class can extends only one class.

abstract class method does not have body.

Ex.

```
abstract class java
{
  abstract void disp();
}
class java1 extend java
{
  void disp()
  {
    System.out.println("Khushi");
  }
  public static void main(String
          args[])
  {
    java obj = new java1();
```

```
        obj.disp();
    }
}
```

* Explain Interface :

An Interface in java is a blueprint of a class.

Interface consist all the abstract or static method.

Interface method does not contain any type of body.

Using Interface, we can achieve multiple ~~inter~~ inheritance.

For Interface, we have to use implements keyword.

- Syntax of create ~~(class)~~ Interface :

```
interface interface
            name
{
    // method
}
```

Ex. interface java
{
  void disp();
}
interface java1
{
  void display();
}
class main implements java,
                java1
{
  void disp()
  {
    System.out.println("Khushi");
  }
  void display()
  {
    System.out.println("Gandhi");
  }
  public static void main(String,
          args[])
  {
    main obj = new ~~obj~~ main();

      obj.disp();
      obj.display();
  }
}