Unit:II Inheritance - Super classes- sub classes -Protected members - constructors in sub classes- the Object class - abstract classes and methods- final methods and classes - Interfaces - defining an interface, implementing interface, differences between classes and interfaces and extending interfaces - Object cloning -inner classes, Array Lists - Strings

2.1.Inheritance

- Reusability is major concept of OOP paradigm
- Java programs will be reused in different places
- Create new programs by make us of old programs

Definition:

- The process of deriving a new class from an old program is called inheritance.
- Old class of java iscalled as base class or super class or parent class and the new c;as of java is called as subclass/derived class/child class.

Types of Inheritance:

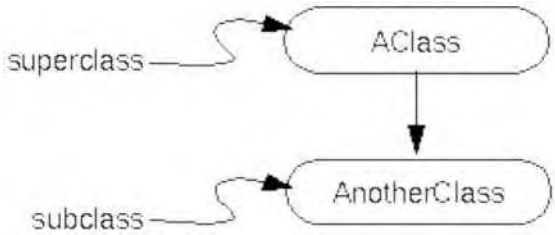
Inheritance can be of any one following types

1. Single inheritance (Only one superclass)
2. Multiple inheritance (Several super classes)
3. Hierarchical inheritance (One superclass, many subclasses)
4. Multilevel inheritance(Derived from a derived class)
5. Hybrid Inheritance(Combination of two inheritances)

Multiple inheritance cannot be used directly in java. This concept is implemented by interface concepts in java

2.1.1. Defining a superclass:

The sub class (the class that is derived from another class) is called a *derived class*. The class from which it's derived is called the *base class or super class*. The following figure illustrates these two types of classes:



2.1.2. Defining a subclass:

Subclass is a class which is formed newly

Syntax for defining a subclass:

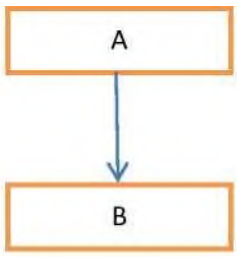
```
Class subclassname extends superclassname
{
        Variable declaration;
        Method declaration;
}
```

Subclassname-subclass or derived class or child class name
Superclassname-superclass or base class or parent class name

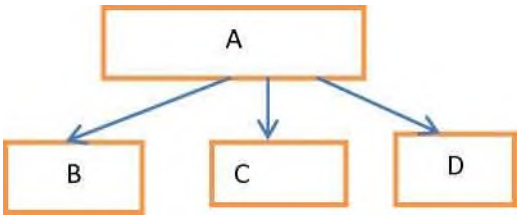
Properties of baseclass are extensive to subclass name.

Pictorial representation of inheritance

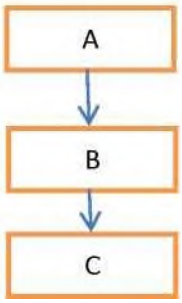
(a)Single inheritance



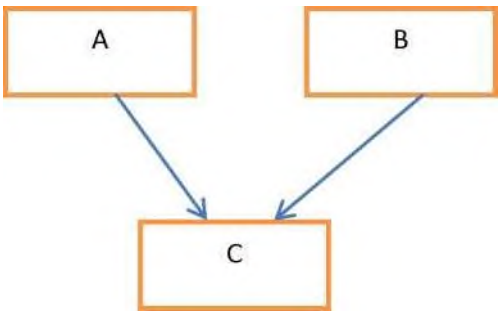
(b)Hierarchical Inheritance



(c)Multilevel inheritance



(d)Multiple inheritance



Single inheritance

The method of inheriting the properties from one super class to one sub class is called single inheritance.

It consists of one base class and one derived class.

Example program1: Single inheritance

```
class Room                    //base class
{
intlength,breadth;
Room(intx,int y)
{
length=x;
breadth=y;
}
int area()
{
return(length*breadth);
}
}
class Bedroom extends Room //derived class using base class named Room {
int height;
        Bedroom(intx,inty, int z)
```

```
{
super(x,y);
height=z;
}
int volume()
{
Return(length*breadth*height);
}
}

classsingleinheritance
{
public static void main(String ars[])
{
Bedroom room1=new Bedroom(14,12,10);
int area1=room1.area();
int volume1=room1.volume();
System.outprmtin("Area1=”+area1);
System.outprmtin("Volume 1=”+volume1);
}
}
```

Output:
Area1=168
Volume1=1680

Example program2:Single inheritance

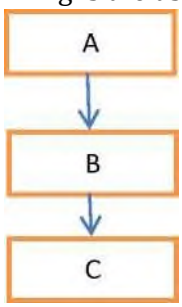
```
class A
{
int x;
int y;
int get(int p, int q)
{
x=p;
y=q;
return(0);
}
void Show()
{
System.out.println(x);
}
}

class B extends A
{
public static void main(String args[])
{
B a = new B();
a.get(5,6);
```

```
a.Show();
}
}
```

Output:
5

Multilevel Inheritance

A generalnecessity in object oriented programming is the use of a derived class as a super class.



A ->B->C is known as inheritance path.

A derived class with multilevel base classes is declared as follows.

```
class A { }
class B extends A {                    //First Level

}
class C extends B                      //Second Level
{
```

Example Program: Multilevel inheritance

```
class students //base class
{
        private int sno;
        private String sname;
        public void setstud(intno,String name) {
                sno=no;
                sname=name;
        }
        public void putstud()
        {
                System.outprmtln("Student No:"+sno);
                System.outprmtln("Student Name:"+sname);
        }
}
class marks extends students //derived or intermediate base class {
        protected int mark1,mark2;
        public void setmarks(int m1,int m2)
        {
                mark1=m1;
                mark2=m2;
        }
        public void putmarks()
```

```
        {
                System.outprmtin("Mark1:"+mark1);
                System.outprmtin("Mark2:"+mark2);
        }
}
class finaltot extends marks // derived class
{
        privateint total;
        public void calc()
        {
                total=mark1+mark2;
        }
        public void puttotal()
        {
                System.outprmtin("Total:"+total);
        }
        public static void main(String args[])
        {
                finaltot f=new finaltot();
                f.setstud(100,"ABC");
                f.setmarks(78,89);
                f.calc();
                f.putstud();
                f.putmarks();
                f.puttotal();
        }
}
```

Example Program2:Multilevel Inheritance

```
class Base
{
        voidbmsg()
        {
        System.outprmtln("Welcome to base class");
        }
}
class Derive1 extends Base
{
        void derive1msg()
        {
        System.outprmtin("Derive1msg”);
        }
}
class Derive2 extends Derive1
{
        void derive2msg()
        {
        System.outprmtin("Derive2msg”);
        }
}
class Multilevel
```

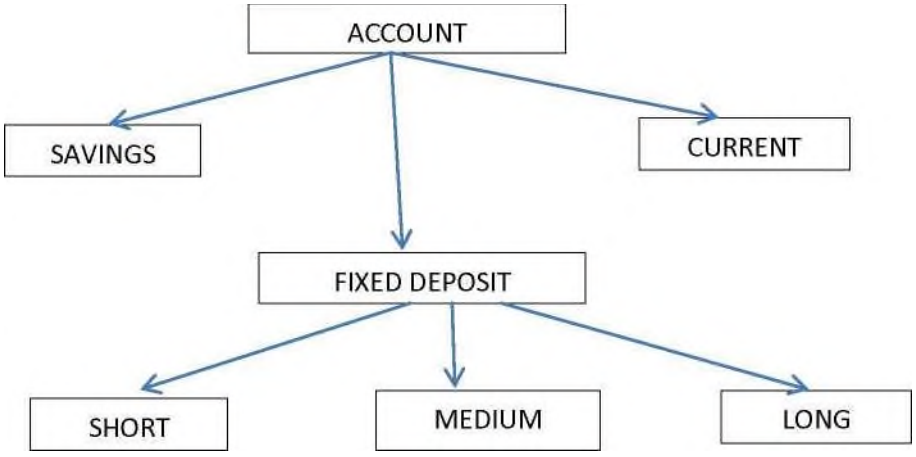
```
{
public static void main(String args[])
{
Derive2 d2=new Derive2();
d2.derive2msg();
d2.derive1msg();
d2.bmsg();
}
}
```

Output:

Derive2msg
Derive1msg
Welcome to base class

Hierarchical inheritance

Class A is a super class of both class B and class C i.e one super class has many sub classes. Some features of one level are shared by many lower level casses



Example Program: Hierarchical inheritance

```
public class A
{
void DisplayA()
{
System.outprmtin("I am in A”);
} }

public class B extends A
{
void DisplayB()
{
System.outprmtin("I am in B”);
} }

public class C extends A
{
void DisplayC()
{
```

```
System.outprmtin("I am in C”);
} }

public class Mainclass
{
System.outprmtin("CaHmg for subclass C”);
C c=new C();
c.DisplayA();
c.DisplayC();
System.outprmtin("CaHmg for subclass B”);
B b=new B();
b.DisplayA();
b.DisplayB();
} }
```

Output:
Calling for subclass C
I am in A
I am in C
Calling for subclass B
I am in A
I am in B

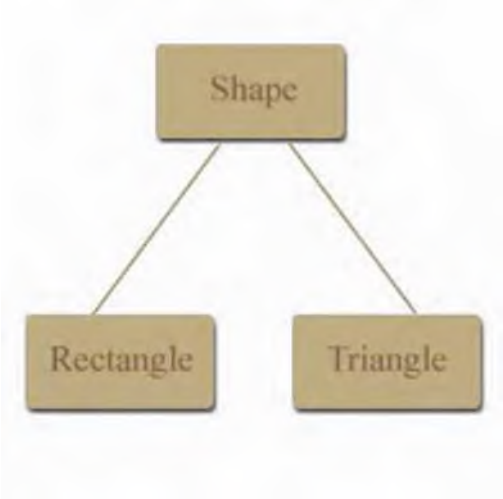
2.1.3. Protected Member:

The private members of a class cannot be openly accessed external class. Only functions of that class can access the private data fields directly. As discussed previously, however, occasionally it may be essential for a subclass to access a private member of a base class. If you make a private member public, then someone can access that member. So, if a member of a base class wants to be (directly) accessed in a subclass and yet still stop its direct access external class, you must declare that member as **protected**.

Following table gives the difference

Modifier	**Class**	**Subclass**	**World**
public	Y	Y	Y
protected	Y	Y	N
private	Y	N	N

Following program illustrates how the functions of a subclass can directly access a protected member of the base class



For example, let's consier a series of classes to describe two types of shapes: rectangles and triangles. These two shapes have definite general properties height and a width (or base).

This could be depicted in the world of classes with a class Shapes from which we can derive the two other ones : Rectangle and Triangle public class Shape

```
{
  protected double height; // To hold height.
  protected double width; //To hold width or base
  public void setValues(double height, double width)
  {
    this.height = height;
    this.width = width;
  }
}

public class Rectangle extends Shape
{
public double getArea()
  {
    return height * width; //accessing protected members
  }
}
```

```
public class Triangle extends Shape {
public double getArea()

    return height * width / 2; //accessing protected members

 }

 public class TestProgram
 {
   public static void main(String[] args)
   {
    //Create object of Rectangle.
    Rectangle rectangle = new Rectangle();

    //Create object of Triangle.
    Triangle triangle = new Triangle();

    //Set values in rectangle object rectangle.setValues(5,4);

    //Set values in trianlge object triangle.setValues(5,10);

    // Display the area of rectangle.
    System.out.println("Area of rectangle : " +
                rectangle.getArea());

    // Display the area of triangle.
    System.out.println("Area of triangle : " +
                triangle.getArea());

}
```

Output :
Area of rectangle : 20.0
Area of triangle : 25.0

2.1.4. Subclass Constructor

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

A **Subclass constructor** is used to build the instance variables of both the subclass and the superclass.

The subclass constructor uses the keyword super to call up the constructor method of the superclass. Keyword super is used subject to the subsequent conditions.

1 ."Super" may only be used within a subclass constructor method.
2 . The call to super class constructor must show as the first statement inside the subclass constructor.
3 .The parameters in the super class must equal to the order and type of the instance

variable declared in the base class

```
class Animal
{
Animal()
{
System.out.println("animal is created");
}
}
class Dog extends Animal{
Dog()
{
super();
System.out.println("dog is created");
}
}
class TestSuper3{
public static void main(String args[]){
Dog d=new Dog();
}
}
```

Output:

```
animal is created
dog is created
```