Inheritance

Session 7

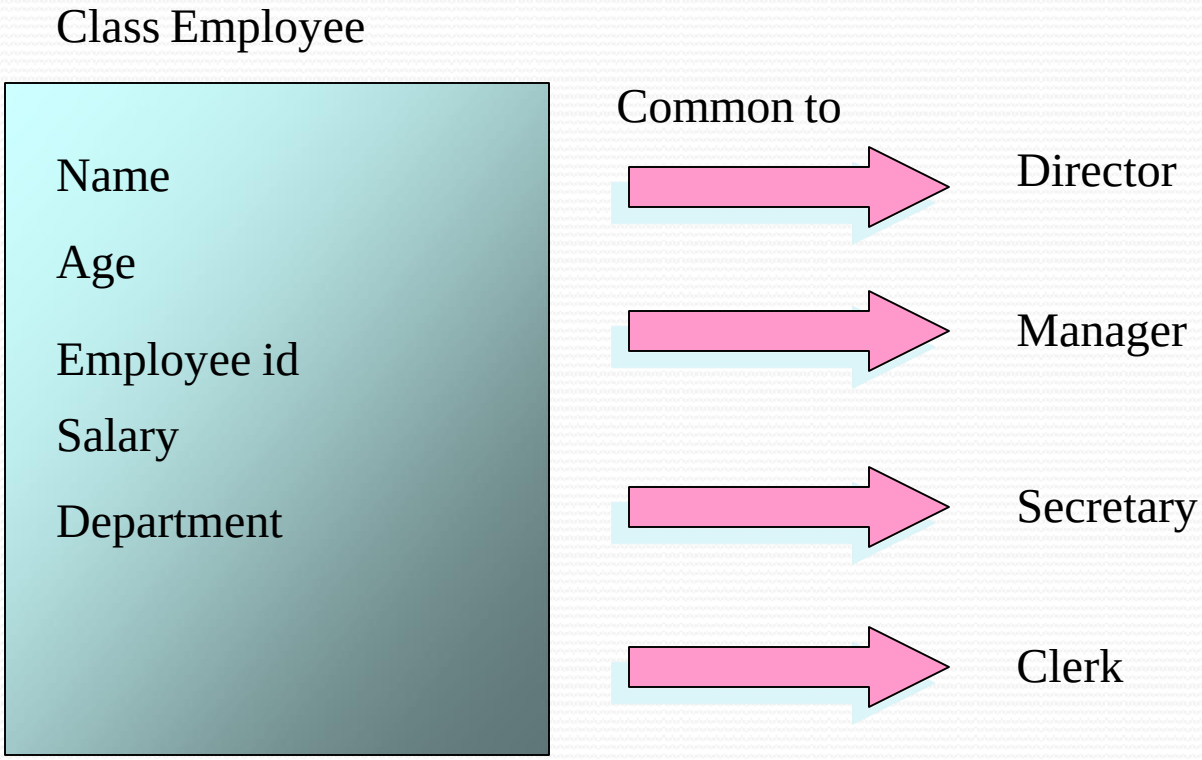
Session Objectives

- Describes inheritance
- Discuss need for inheritance
- Describe types of inheritance
- Implement inheritance using Java
- Describe how to access members in inheritance
- Implement Super()
- Determine the calling sequence of constructors

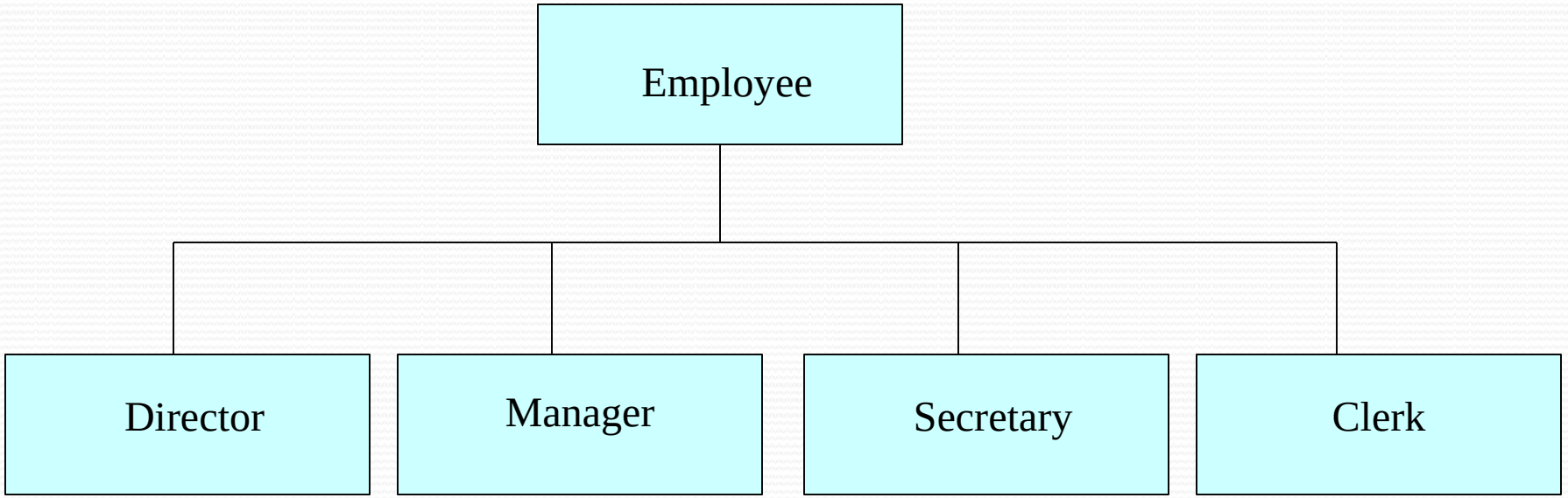
Features of an Object

- An object must combine data and behavior
- Objects must focus on essential properties ignoring any accidental properties
- Object must hide the implementation details from the rest of the world. It may expose certain functionalities to otherobjects
- Object must understand messages from other objects and can pass messages to other objects .
- An object can be reused on any future project

Inheritance



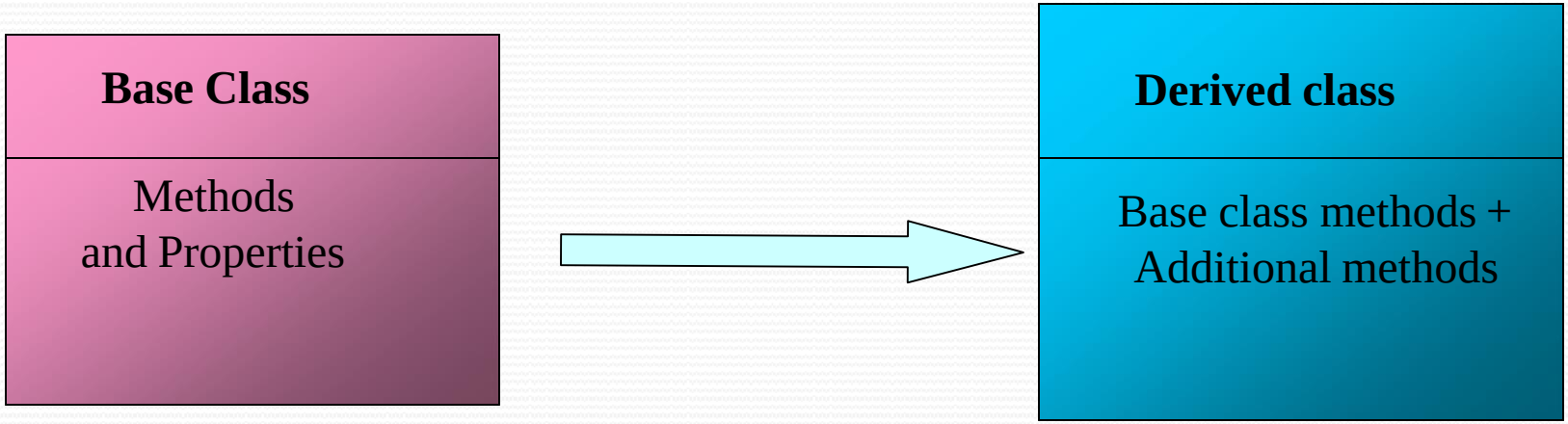
Inheritance



Each of the sub-classes is considered to be derived from the parent class Employee

Inheritance

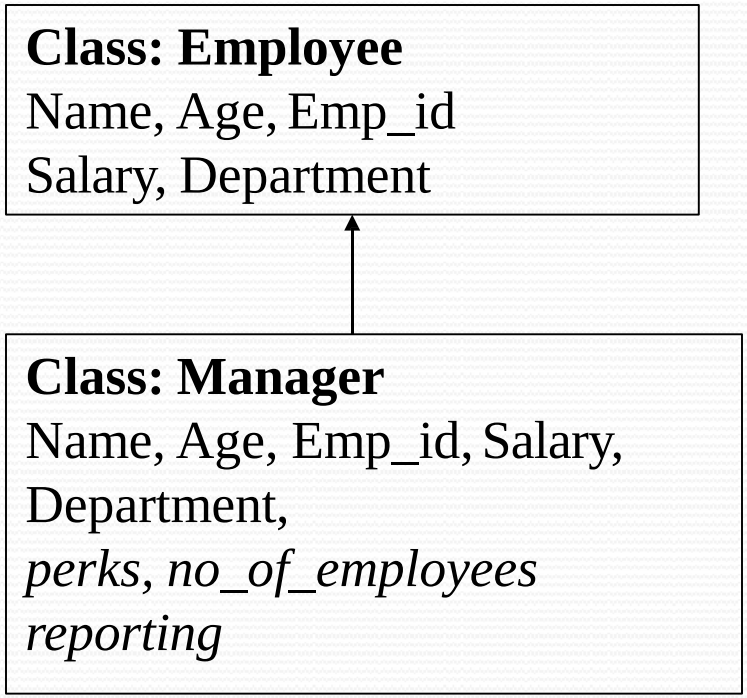
Inheritance is the property that allows the reuse of an existing class to build a new class



Advantages of Inheritance

- Reusability of code
- The base class need not be changed but can be adapted to suit the requirements in different applications
- Saves developers time and effort so that they need not to spend time to know the core technical facts

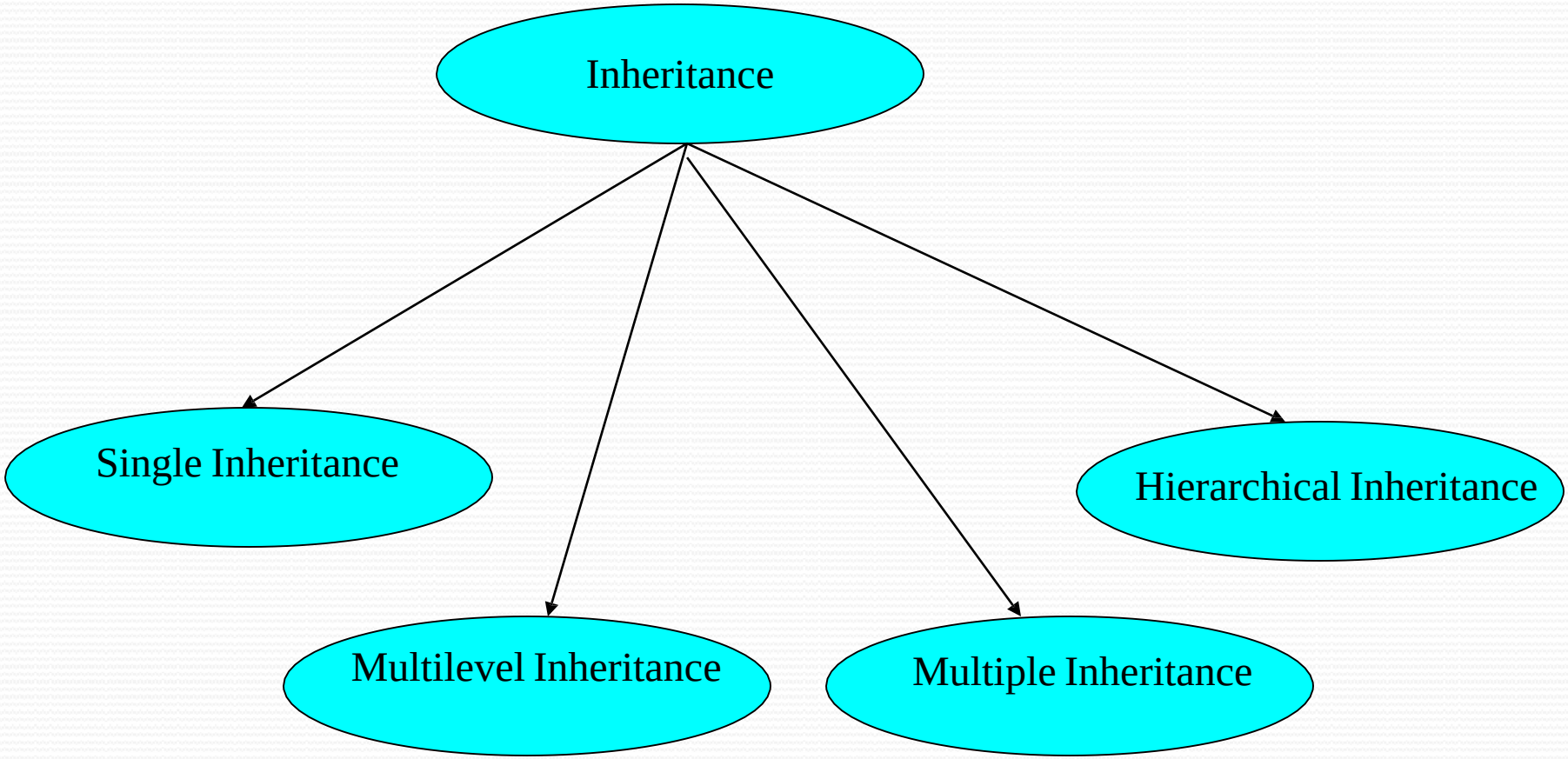
Generalization and Specialization



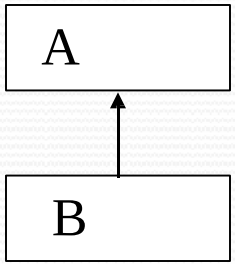
Base Class and Derived Class

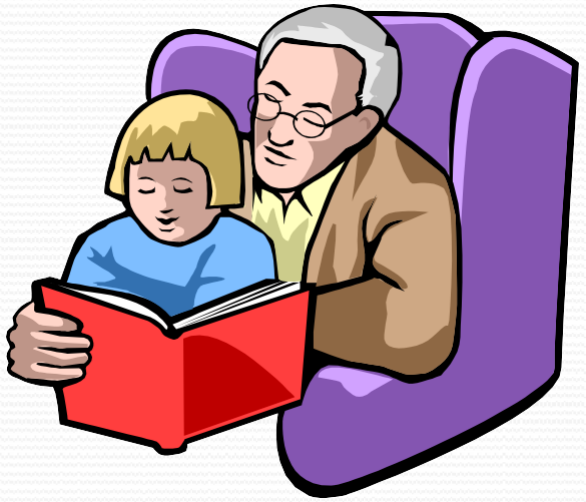
```
class Employee{    //Base Class
        String Name;
        int Age;
        String emp_id;
        Float Salary;

        public void create()
        { ….. }
        public void resign()

        { ……. }
}
class Manger extends Employee{  //Subclass

        float perks;

        int no_of_employees;

        public void report()     { …… }

}
```

Types of Inheritance



Single Inheritance

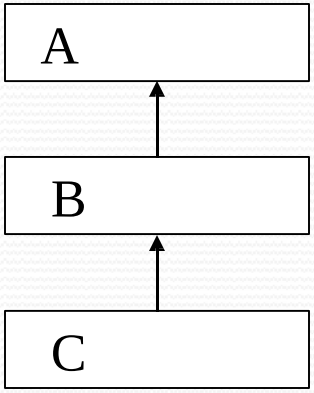




The son will have his own features as well as features of the father

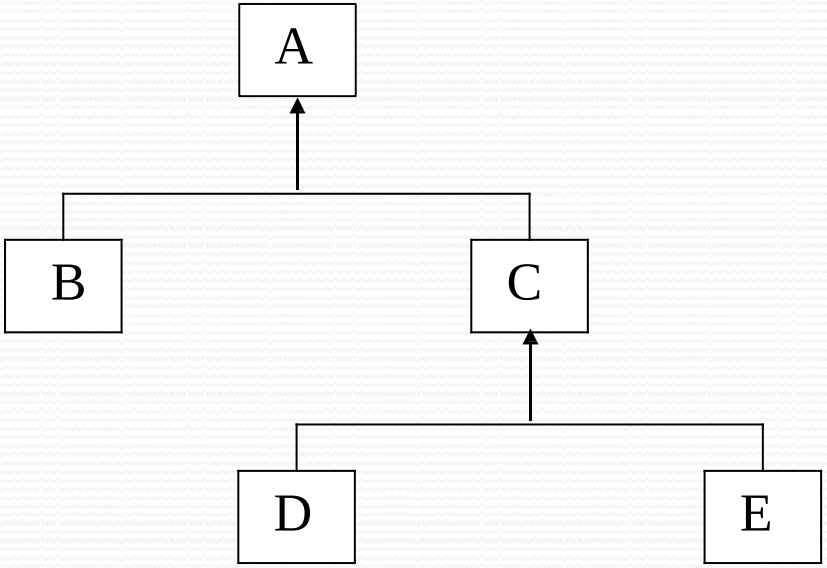
```
class A

{          ……          }
class  B  extends  A
{ …. }
```

Multilevel Inheritance




```
class A
{ …… }
class B extends A
{ ……..}
class C extends B{
……}
```

Hierarchical Inheritance




```
class A
{ …… }
class B extends A
{ ……..}
class C extends A
{ ……}

class D extends C

{ ……}


class E extends C

{ ……}
```

Single Level Inheritance in Java

```
class Container{
        int width;
        int height;
        int depth;
     Container() //default constructor
{
       System.out.println("Default object created with zero
       dimension");
          width=0;
          height=0;
          depth=0;
}
Container(Container C) //Pass object to constructor
```

Single Level Inheritance

```
{
          width=C.width;
          height=C.height;
          depth=C.depth;
      }
Container(int W, int H, int D)//Pass values to
constructor
{
         width=W;
          height=H;
          depth=D;
      }
Container(int C) //Constructor for Cube
{
 // In Cube width=height=depth
            width=C;
            height=C;
            depth=C;
      }
```

Single Level Inheritance

```
long getVolume()
{
         return width*height*depth;
        }
}
class Containerweight extends Container {
        int filledweight;

Containerweight(int W, int H, int D, int WT)
{
         width=W;
         height=H;
         depth=D;
         filledweight=WT;
         System.out.println("This is from the derived
class");
     }

}
```

Single Level Inheritance

```
class DemoSingleInh
{
   public static void main(String args[])
  {
    //Creating Base class object created
Containerweight containerX = new
Containerweight(10,20,15,35);
        long vol;
        vol = containerX.getVolume();
        System.out.println("Volume of container = " +
vol);
        System.out.println("Weight of container = " +
     containerX.filledweight);
  }

}
```

Multi-level Inheritance

```
class Container{
        int width;
        int height;
        int depth;
Container() //default constructor
{
      System.out.println("Default object created with zero
      dimension");
         width=0;
         height=0;
         depth=0;
}
Container(Container C) //Pass object to constructor
{
        width=C.width;
         height=C.height;
         depth=C.depth;
```

Multi-level Inheritance

```
Container(int W, int H, int D)//Pass values to  constructor
{
          width=W;
          height=H;
          depth=D;

}
Container(int C) //Constructor for Cube
{
 // In Cube all width=height=depth
         width=C;
          height=C;
          depth=C;
 }
long getVolume()
{         return width*height*depth;    }
```

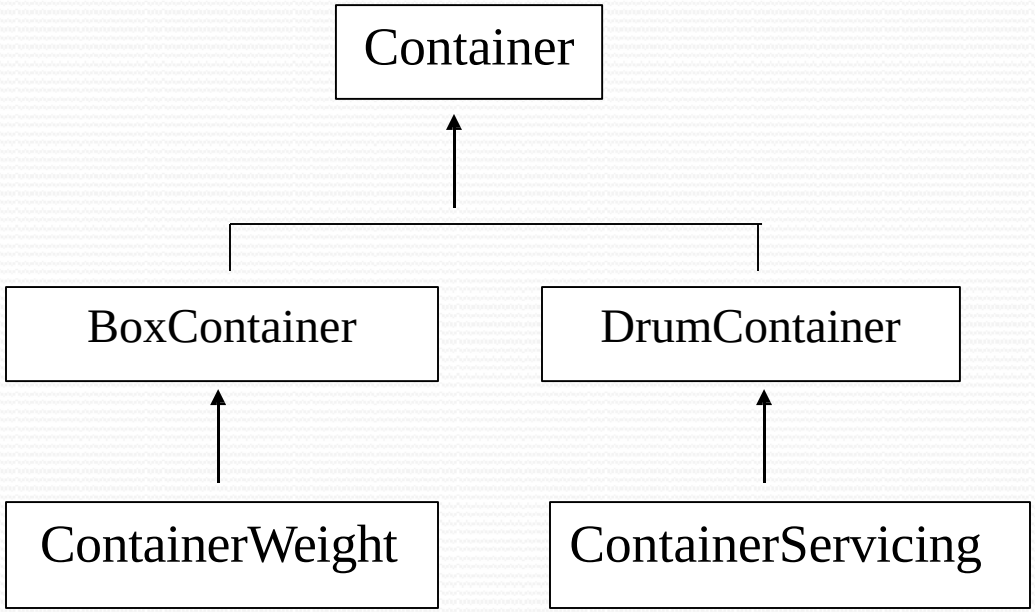
Multi-level Inheritance

```
}
class Containerweight extends Container
{
int filledweight;

Containerweight(int W, int H, int D, int WT)
{
        width=W;
        height=H;
        depth=D;
        filledweight=WT;
        System.out.println("This is from the derived class");
}

}
class Shipment extends Containerweight {
       int cost;
Shipment(int W, int H, int D, int WT, int C) {
```

Multi-level Inheritance

```
            width=W;
            height=H;
            depth=D;
            filledweight=WT;
            cost=C;
            System.out.println("This is from the derived class");
}
}
class DemoMultilevelInh{
public static void main(String args[])
{
          Shipment shipcontainerX = new Shipment(5,10,7,20,4);
          long vol;
          vol = shipcontainerX.getVolume();
          System.out.println("Volume of container = " + vol);
          System.out.println("Weight of container = " +
      shipcontainerX.weight);

 }

}
```

Hierarchical Inheritance



Hierarchical Inheritance

```
class Container
{
        String containerNo;
        Container(String S)
        {……}
        SetContainerNo(String S){      ……
        }

                getContainerNo()
{

        return containerNo;
        }

}
```

Hierarchical Inheritance

```
class BoxContainer extends Container{
        int width;
        int height;
        int depth;
     Boxcotainer() { …… }//default constructor
Boxcontainer(int W, int H, int D,String S)//Pass values to
constructor
{……..}
long getVolume()
{

          ……..   }

}
```

Hierarchical Inheritance

```
}
class Containerweight extends BoxContainer
{
        int filledweight;
Containerweight(int W, int H, int D, int WT)

{ …….. }
}
class ContainerServicing extends BoxContainer {
        int servicvecost;

                    Shipment(…….)
{       ……..                                                    }
int getServiceCost(){……..}}

                            ……..
……
```

Rules for Access modifiers

The methods of the derived class can access members of the base class if its members are public.

The private keyword makes a member of a class really private. Because the derived class members cannot access the private members of the base class.

The scope of the base class has to be broader than that of derived class. For example, If the base class has private as its access specifier then the derived class cannot have the access specifier as public.

Class members can always be accessed by methods of their own class, whether the members are private or public. But the inherited class's object can access base class members only if the members are public or protected.

Rules for Access modifiers

Protected members of a class can be accessed by objects or functions from its sub-classes. However, the difference between them appears only in derived classes.

Private members of a class cannot be derived, only public and protected members of a class can be derived”

Access modifiers

```
class Employee{                                  //base class
   private int privA;
   protected int protA;
   public int pubA;

public static void main(String args[])
{

  Employee emp = new Employee();                 //Object of base class type

        emp.privA = 1;                               //valid

        emp.protA = 1;                               //valid

        emp.pubA = 1;                                //valid
```

Access modifiers

```
Manager mgr = new Manager();   //object of derived class
         mgr.privA = 1;        //error:not accessible
         mgr.protA = 1;        //valid
         mgr.pubA = 1;         //valid

}}
class Manager extends Employee{       //derived class

        void fn()
        { int a;
         a = privA;            //error:not accessible

         a = protA;            //valid

         a = pubA;             //valid

        }

}
```

Using Super

To initialize data members of the superclass which are hidden from the subclass (because of their private access modifier) we can use the Super keyword.

Using Super we can save extra lines of coding

Data hiding principles can be restored using Super.

Using Super

```
class Container{ int width;
    int height;
     int depth;
     Container() //default constructor
     {
       ……..}
Container(Container C) {
 width=C.width;
 height=C.height;
 depth=C.depth;
 }
Container(int W, int H, int D) {
 width=W;
 height=H;
 depth=D;
}
```

Using Super

```
Container(int C) {
width=C;
height=C;
depth=C;
}
long getVolume()
{ …….}
}
}
class Containerweight extends Container {
 int filledweight;
Containerweight() {
 super();
     filledweight=0;
     System.out.println("Calling default constructor  of Base
   class from derived class");
}
```

Using Super

```
Containerweight(int W, int H, int D, int WT) {
 super(W,H,D);
   filledweight=WT;
   System.out.println("Calling          parameterised
   constructor of Base class from derived class");
  }
Containerweight(Containerweight cobj) {
 super(cobj);
    filledweight=cobj.filledweight;
    System.out.println("Calling constructor of Base
     class passing object from derived class");
 }
}
```

Using Superclass variable

```
class DemoSingleInh {
public static void main(String args[])  {
Containerweight containerX =  new Containerweight(10,20,15,35);
Container Cobj=new Container();
long vol;
vol = containerX.getVolume();
System.out.println("Volume of container = " + vol);
System.out.println("Weight of container = " +
containerX.filledweight);
Cobj=containerX; //assign object of subclass to a super class reference.
vol = Cobj.getVolume();
System.out.println("Volume of container(duplicate)= "+ vol);
/*System.out.println("Weight of container(duplicate) ="  +
Cobj.filledweight); */
}
}
```

Using Super keyword

```
public class Room  {
double height;
double length;
double width;
}
class bedroom extends Room
{
          double height;
           double length;
 double width;
bedRoom() {
    super.height //height in Room
    //main method
…..
}
}
```

The super can be used in another way – as reference. The super calls members of superclass from a derived class.

`super.member`

Calling sequence of Constructor

```
class ShowSequence
{
  public static void main(String args[])

 {
    //Creating Inherited class object
   Shipment shipcontainerX =
          new Shipment(5,10,7,20,4);
  }
}
```