Java Programming

Object Oriented Programming - I

Module 3

Agenda

1. Encapsulation and Abstraction
2. Inheritance
3. Multilevel Hierarchy

Objectives

At the end of this module, you will be able to:

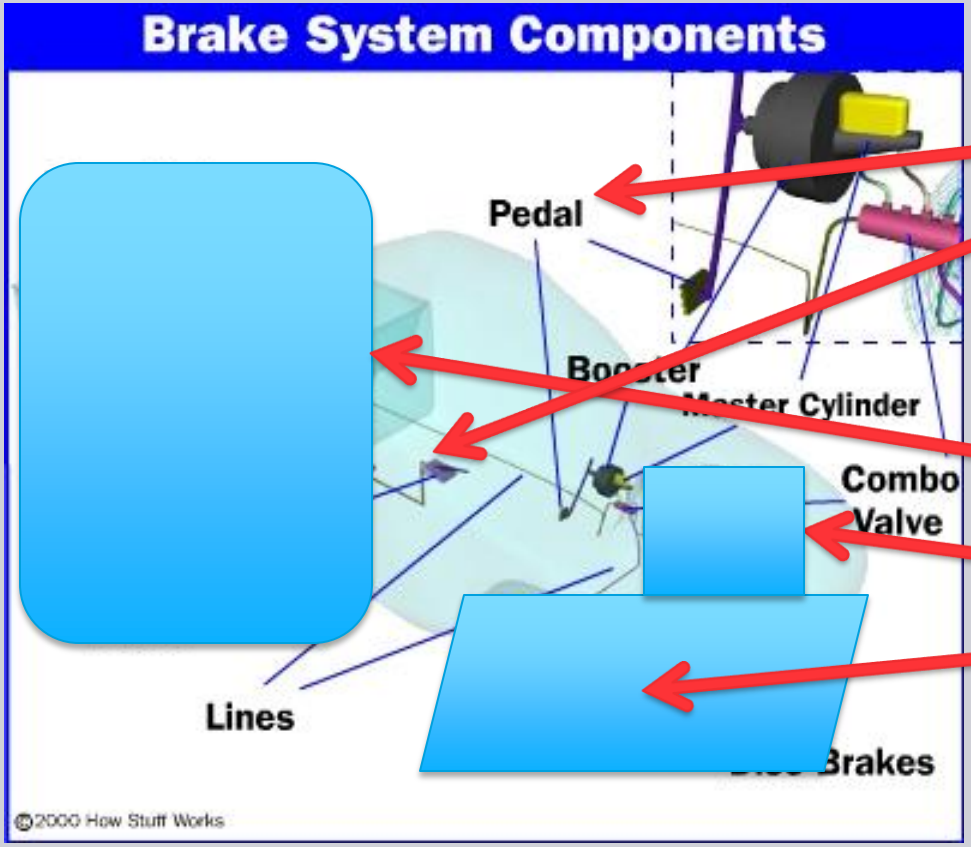
- Apply Encapsulation and Abstraction in java code
- Describe Java's inheritance model and its language syntax
- Describe the usage of the keyword **super**
- Define a multilevel hierarchy

Encapsulation and Abstraction

Introduction to Object Oriented Programming

- Object Oriented Programming is a programming paradigm which uses "Objects" consisting of data fields and methods together with their interactions
- It is used to design applications and computer programs
- Programming technique may include features like encapsulation, abstraction, polymorphism and inheritance

Encapsulation and Abstraction



Abstraction

Encapsulation

Encapsulation is hiding the implementation level details

Abstraction is exposing only the interface

Defining a Sample point Class

```
class Point {
   int x;    int y;
   void setX( int x){
   x = (x > 79 ? 79 : (x < 0 ? 0 :x)); }
  void setY (int y){
   y = (y > 24 ? 24 : (y < 0 ? 0 : y)); }
  int getX( ){ return x; }
  int getY( ){ return y;}
}
```

Access Specifiers

- Java provides access specifiers to control access to class members
- Access specifiers help implement:
  - Encapsulation by hiding implementation-level details in a class
  - Abstraction by exposing only the interface of the class to the external world
- The **private** access specifier is generally used to encapsulate or hide the member data in the class
- The **public** access specifier is used to expose the member functions as interfaces to the outside world

Class Declaration for Point

```
class Point{
  private int x;
  private int y;
  public void setX( int x){
    x= (x > 79 ? 79 : (x < 0 ? 0 :x));
  }
  public void setY (int y){
    y= (y > 24 ? 24 : (y < 0 ? 0 : y));
  }
  public int getX( ){
    return x;
  }
public int getY( ){
    return y;
  }
}
```

Class Declaration for Point (Contd.).

```
class PointDemo {
  public static void main(String args[ ]
  ){
    int a, b;
    Point p1 = new Point( );
    p1.setX(22);
    p1.setY(44);
    a = p1.getX( );
    System.out.println("The value of a is
  "+a);
    b = p1.getY( );
    System.out.println("The value of b is
  "+b);
  }
}
```

Expected Output :
The value of a is 22
The value of b is 24

Actual Output :
The value of a is 0
The value of b is 0

?

Class Declaration for Point- modified

```
class Point{
  private int x;
  private int y;
  public void setX( int x){
    this.x= (x > 79 ? 79 : (x < 0 ? 0 :x));
  }
  public void setY (int y){
    this.y= (y > 24 ? 24 : (y < 0 ? 0 : y));
  }
  public int getX( ){
    return x;
  }
public int getY( ){
    return y;
  }
}
```

Class Declaration for Point - modified (Contd.).

```
class PointDemo {
  public static void main(String args[ ]
  ){
    int a, b;
    Point p1 = new Point( );
    p1.setX(22);
    p1.setY(44);
    a = p1.getX( );
    System.out.println("The value of a is
  "+a);
    b = p1.getY( );
    System.out.println("The value of b is
  "+b);
  }
}
```

Output :

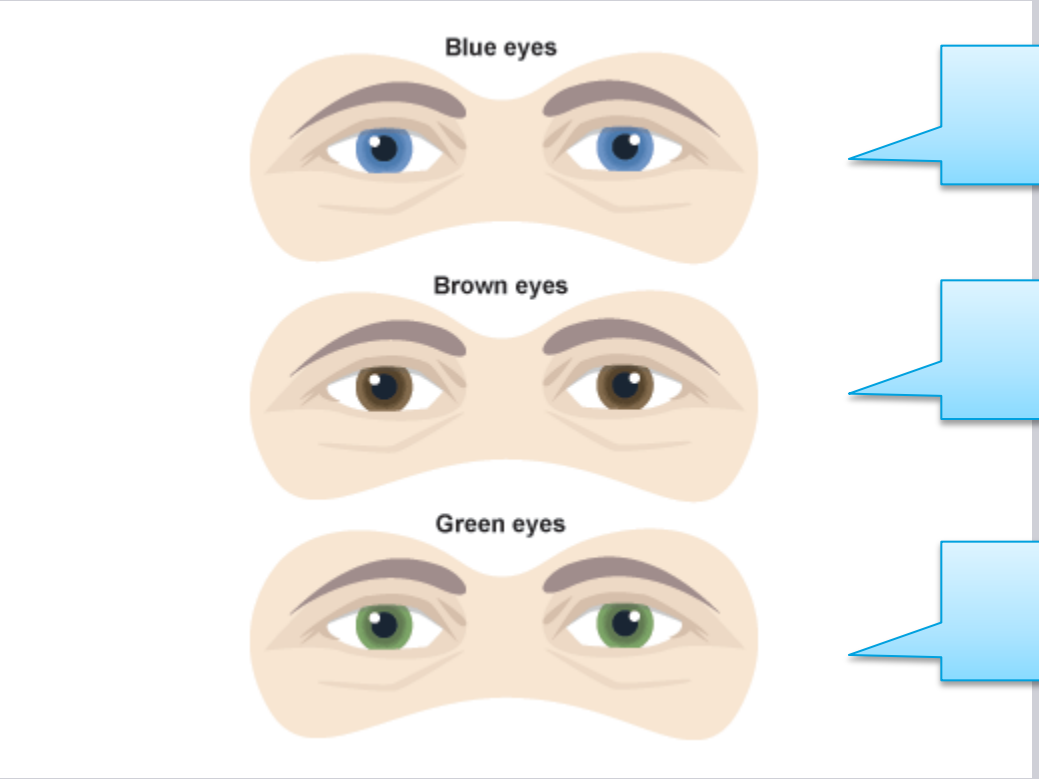
The value of a is 22

The value of b is 24

Inheritance

Inheritance in real world..

Have you seen some people who has BLUE EYES?

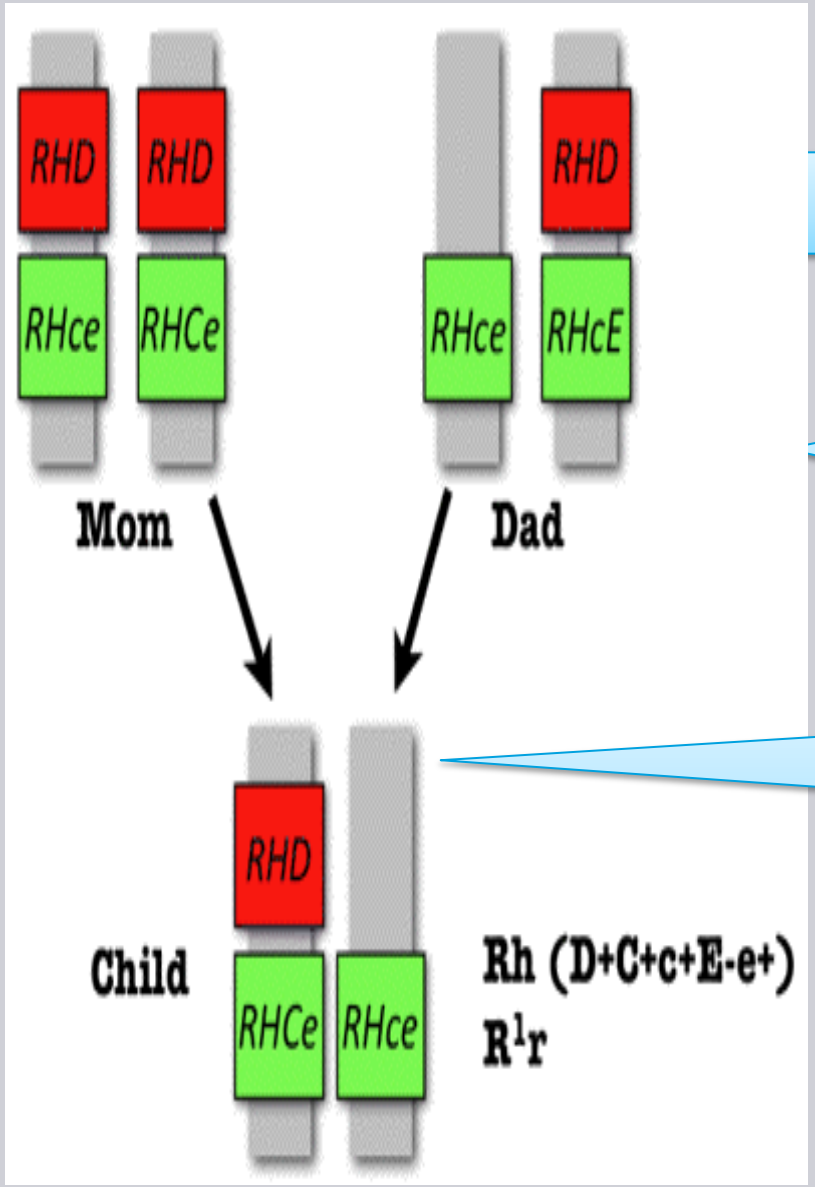


Some people have BLUE EYE: How it is possible?

Some people have BROWN EYE: How it is possible?

Some people have GREEN EYE: How it is possible?

Inheritance in real world (Contd.).



In real life, what is meant by inheritance?

What you inherit from your parent? What is your own behavior which is not found in your parent?

Child inherits some properties from its parent. Apart from that, child has its own properties….

Inheritance

- Inheritance is one of the cornerstones of OOP because it allows for the creation of hierarchical classifications
- Using inheritance, you can create a general class at the top
- This class may then be inherited by other, more specific classes
- Each of these classes will add only those attributes and behaviors that are unique to it

Generalization/Specialization

- In keeping with Java terminology, a class that is inherited is referred to as a superclass
- The class that does the inheriting is referred to as the subclass
- Each instance of a subclass includes all the members of the superclass
- The subclass inherits all the properties of its superclass

Association, Aggregation, Composition

- These terms are used to signify the relationship between classes
- They are the basic building blocks of OOPS

Association

- Association is a relationship between two objects
- The association between objects could be
  - one-to-one
  - one-to-many
  - many-to-one
  - many-to-many
- Types of Association
  - Aggregation
  - Composition

- ***Example:*** A Student and a Faculty are having an association

Aggregation

- Aggregation is a special case of association
- A directional association between objects
- When an object ‘has-a’ another object, then you have got an aggregation between them
- Aggregation is also called a “Has-a” relationship.
- Example: College has a Student Object

Composition

- Composition is a special case of aggregation
- In a more specific manner, a restricted aggregation is called composition
- When an object contains the other object, if the contained object cannot exist without the existence of container object, then it is called composition
- ***Example:*** A class contains students. A student cannot exist without a class. There exists composition between class and students

IS-A relationship: Manager IS-A Employee

4 Employees of a department

Their Manager

HAS-A relationship

- ***HAS-A*** relationship is expressed with containership
- Containership simply means using instance variables that refer to other objects
- Example:
- The class House will have an instance variable which refers to a Kitchen object
  - It means that, House HAS-A Kitchen
  - Note that, something like Kitchen HAS-A House is not valid in this context

HAS-A relationship (Contd.).

- Let us take one personal computer.
- It has a monitor, CPUbox, keyboard and mouse, etc.
- Technically we can say that,
  - Personal Computer class HAS-A monitor.
  - Personal Computer class HAS-A CPUbox
  - Personal Computer class HAS-A keyboard.
  - Personal Computer class HAS-A mouse.
  - The most important point is : the 4 independent components like monitor, keyboard, CPUbox and mouse cannot function separately on its own.
  - But, by combining them, we are creating a new type of useful class called Personal Computer.

Java's Inheritance Model

- Java uses the single inheritance model
- In single inheritance, a subclass can inherit from **one (and only one)** superclass

Code Syntax for Inheritance:

```
class derived-class-name extends base-class-name {
// code goes here
}
```

Inheritance – A Simple Example

```
class A{
  int m, n;
  void display1( ){
    System.out.println("m and n are:"+m+" "+n);
  }
}
class B extends A{
  int c;
  void display2( ){
    System.out.println("c :" + c);
  }
  void sum(){
    System.out.println("m+n+c = " + (m+n+c));
  }
}
```

Inheritance – A Simple Example (Contd.).

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

    A s1 = new A();  // creating objects
    B s2 = new B();
    s1.m = 10; s1.n = 20;

    System.out.println("State of object A:");
    s1.display1();
    s2.m = 7; s2.n = 8; s2.c = 9;
    System.out.println("State of object B:");
    s2.display1();
    s2.display2();
    System.out.println("sum of m, n and c in object B
  is:");
    s2.sum();
  }
}
```

Accessing Superclass Members from a Subclass Object

- A subclass includes all of the members of its superclass

- But, it cannot directly access those members of the super class that have been declared as **private**.

```
class A{
  int money;
  private int pocketMoney;

  void fill  (int money, int pocketMoney)
  {
    this.money = money;
    this.pocketMoney = pocketMoney;
  }
}
```

Accessing Superclass Members from a Subclass Object (Contd.).

```
class B extends A{
  int total;
  void sum( ){
    total = money + pocketMoney;
  }
}
class AccessDemo
{
  public static void main(String args[ ])
  {
    B subob = new B();
    subob.fill(10,12);
    subob.sum();
    System.out.println("Total: " + subob.total);
  }
}
```

Will this compile now?

A Possible Solution To The Program

```
class A{
  int money;
  private int pocketMoney;
  void fill(int money, int pocketMoney)
  {
    this.money = money;
    this.pocketMoney = pocketMoney;
  }
  public int getPocketMoney(){
    return pocketMoney;
  }
}
```

A Possible Solution To The Program (Contd.).

```
class B extends A{
  int total;
  void sum( ) {
    total = money + getPocketMoney();    }
}
class AccessDemo {
  public static void main(String args[ ]) {
        B subob = new B();
        subob.fill(10,12);
        subob.sum();
        System.out.println("Total: " + subob.total);
  }
}
```

Will this compile now?

Using super

- The creation and initialization of the superclass object is a prerequisite to the creation of the subclass object.
- When a subclass object is created,
  - It creates the superclass object
  - Invokes the relevant superclass constructor.
    - The initialized superclass attributes are then inherited by the subclass object
  - finally followed by the creation of the subclass object
    - initialization of its own attributes through a relevant constructor subclass

Using super (Contd.).

- The constructors of the superclass are never inherited by the subclass
- This is the only exception to the rule that a subclass inherits all the properties of its superclass

A Practical Example

Employee class should be put inside the mypack directory..

```
package mypack;
 class Employee {
  int Employeeno;    String Empname;
    Employee()       {
        System.out.println(" Employee No-arg Constructor
  Begins");
        Employeeno =0; Empname= null ;
        // the above assignments are unnecessary .. Why?
        System.out.println(" Employee No-arg Constructor
  Ends");
    }
    Employee(int Employeeno)      {
    System.out.println(" Employee 1-arg Constructor Begins");
    this.Employeeno=Employeeno;
    this.Empname= "UNKNOWN";
    System.out.println(" Employee 1-arg Constructor Ends");
    }
```

A Practical Example (Contd.).

```
  Employee(int Employeeno, String s) {
      System.out.println(" Employee 2-arg Constructor
Begins");
      this.Employeeno = Employeeno;
      this.Empname  = s;
      System.out.println(" Employee 2-arg Constructor
Ends");
  }
  void display()      {
      System.out.println(" Employee Number =
"+Employeeno);
      System.out.println(" Employee Name = "+Empname);
  }

} // End of the Employee class
```

A Practical Example (Contd.).

```
class Manager extends Employee
{
   String deptname;
   Manager(int Employeeno, String name, String
     deptname )
   {
      super(Employeeno, name);
      // parent class 2-arg constructor is called

      System.out.println(" Manager 3-arg Constructor
  Begins");
      this.deptname = deptname;
      System.out.println(" Manager 3-arg Constructor
  Ends");
       }
```

A Practical Example (Contd.).

```
 void display()   {
       super.display();
       // parent class display() function is called
       System.out.println(" Deptname = "+deptname);
 }
public static void main( String a[])    {
     System.out.println(" [Main function Begins--------------------
     --] ");
     System.out.println(" Creating an object for manager class ");
     Manager mm = new Manager(10,"Gandhi","Banking");
     System.out.println(" Printint the manager details .... : ");
     mm.display();
       System.out.println(" [Main function Ends----------------------
     -] ");
     }
 }
```

Using super to Call Superclass Constructors

- super() if present, must always be the first statement executed inside a subclass constructor.

- It clearly tells you the order of invocation of constructors in a class hierarchy.

- **Constructors are invoked in the order of their derivation**

Constructors – Order of Invocation

Constructors in a class hierarchy are invoked in the order of their derivation.

```
class X  {
  X( ){
    System.out.println("Inside X's Constructor");  }  }

class Y  extends X {
    Y( ) {
    System.out.println("Inside Y's Constructor");  }  }

class Z extends Y {
    Z(){
    System.out.println("Inside Z's Constructor");  }  }

class OrderOfConstructorCallDemo{
  public static void main(String args[]){
            Z z = new Z();
  }
}
```

You can easily find the output of this program..

Constructors – Order of Invocation (Contd.).

- When we invoke a super() statement from within a subclass constructor, we are invoking the immediate super class' constructor
- This holds good even in a multi level hierarchy
- Remember, **super( )** can only be given as the first statement within a constructor

Using this() in a constructor

- this(argument list) statement invokes the **constructor** of the **same class**

- first line of a constructor must EITHER be a super *(call on the super class constructor)* OR a this *(call on the constructor of same class)*

- If the first statement within a constructor is NEITHER super() NOR this(), then the compiler will automatically insert a super(). (That is, invocation to the super class' no argument constructor)

What is the result, if we try to compile & execute the following code :

```
class A1 {
   A1(){ System.out.println("A1's no arg constructor"); }
   A1(int a){ System.out.println("A1's constructor "+ a); }
}
class B1 extends A1{
   B1(){  System.out.println("B1's no arg constructor"); }
   B1(int b){  super(1000);
               System.out.println("B1's constructor "+ b); }
}
class C1 extends B1{
   C1() {System.out.println("C1's no arg constructor"); }
   C1(int c){ super(100);
               System.out.println("C1's constructor "+ c); }
}
class TestingInheritance{
   public static void main(String args[]){
       C1 ca = new C1();
   }
}
```

The participants are expected to answer this question during session

What is the result, if we try to compile & execute the following code (Contd.).

```
class A1 {
   A1(){ System.out.println("A1's no arg constructor"); }
   A1(int a){ System.out.println("A1's constructor "+ a); }
}
class B1 extends A1{
   B1(){  System.out.println("B1's no arg constructor"); }
   B1(int b){  super(1000);
               System.out.println("B1's constructor "+ b); }
}
class C1 extends B1{
   C1() {System.out.println("C1's no arg constructor"); }
   C1(int c){ super(100);
               System.out.println("C1's constructor "+ c); }
}
class TestingInheritance{
   public static void main(String args[]){
       C1 ca = new C1(10);
   }
}
```

The participants are expected to answer this question during session

What is the result, if we try to compile & execute the following code (Contd.).

```
class A1 {
   A1(){ System.out.println("A1's no arg constructor"); }
   A1(int a){ System.out.println("A1's constructor "+ a); }
}
class B1 extends A1{
   B1(){  System.out.println("B1's no arg constructor"); }
   B1(int b){  super(1000);
               System.out.println("B1's constructor "+ b); }
}
class C1 extends B1{
   C1() {System.out.println("C1's no arg constructor"); }
   C1(int c){ System.out.println("C1's constructor "+ c); }
}
class TestingInheritance{
   public static void main(String args[]){
      C1 ca = new C1(10);
   }
}
```

The participants are expected to answer this question during session

What is the result, if we try to compile & execute the following code (Contd.).

```
class A1 {
   A1(){ System.out.println("A1's no arg constructor"); }
   A1(int a){ System.out.println("A1's constructor "+ a); }
}
class B1 extends A1{
   B1(){  System.out.println("B1's no arg constructor"); }
   B1(int b){ System.out.println("B1's constructor "+ b); }
}
class C1 extends B1{
   C1() { super(100);
      System.out.println("C1's no arg constructor"); }
   C1(int c){ System.out.println("C1's constructor "+ c); }
}
class TestingInheritance{
   public static void main(String args[]){
      C1 ca = new C1(10);
   }
}
```

The participants are expected to answer this question during session

What is the result, if we try to compile & execute the following code (Contd.).

```
class A1 {
   A1(){ System.out.println("A1's no arg constructor"); }
   A1(int a){ System.out.println("A1's constructor "+ a); }
}
class B1 extends A1{
   B1(){ super(50);
       System.out.println("B1's no arg constructor"); }
   B1(int b){  super(1000);
              System.out.println("B1's constructor "+ b); }
}
class C1 extends B1{
   C1() {System.out.println("C1's no arg constructor"); }
   C1(int c){ System.out.println("C1's constructor "+ c); }
}
class TestingInheritance{
   public static void main(String args[]){
       C1 ca = new C1(10);
   }
}
```

The participants are expected to answer this question during session

What is the result, if we try to compile & execute the following code (Contd.).

```
class A1 {
   A1(){ System.out.println("A1's no arg constructor"); }
   A1(int a){ System.out.println("A1's constructor "+ a); }
}
class B1 extends A1{
   B1(String x){ super(50);
      System.out.println("B1's no arg constructor"); }
   B1(int b){  super(1000);
            System.out.println("B1's constructor "+ b); }
}
class C1 extends B1{
   C1() {System.out.println("C1's no arg constructor"); }
   C1(int c){ super(100);
            System.out.println("C1's constructor "+ c); }
}
class TestingInheritance{
   public static void main(String args[]){
      C1 ca = new C1(10);
   }
}
```

The participants are expected to answer this question during session

What is the result, if we try to compile & execute the following code (Contd.).

```
class A1 {
   A1(){ System.out.println("A1's no arg constructor"); }
   A1(int a){ System.out.println("A1's constructor "+ a); }
}
class B1 extends A1{
   B1(){  System.out.println("B1's no arg constructor"); }
   B1(int b){  this("x");
               System.out.println("B1's constructor "+ b); }
   B1(String b){  super(1000);
               System.out.println("B1's constructor "+ b);  }
}
class C1 extends B1{
   C1() {System.out.println("C1's no arg constructor"); }
   C1(int c){ super(100);
              System.out.println("C1's constructor "+ c); }
}
class TestingInheritance{
   public static void main(String args[]){
       C1 ca = new C1(10);
   }
}
```

The participants are expected to answer this question during session

Multilevel Hierarchy

Defining a Multilevel Hierarchy

- Java allows us to define multiple layers in an inheritance hierarchy
- We can define a superclass and a subclass, with the subclass in turn becoming a superclass for another subclass
- Consider the following example…
  - Employee
  - Manager **is a** Employee
  - Director **is a** Manager
    - This is an example for multilevel inheritance

Draw the inheritance tree for this example..

Defining a Multilevel Hierarchy (Contd.).

```
class Employee extends Object { }
class Manager extends Employee { }
class Director extends Manager { }
public class Test_Multi_Level_Inheritance
{
public static void salary(Object obj)
  {
  // Here, Object obj will accept the following:
  //  Object class objects
  //  Employee class objects
  //  Manager class objects
   //  Director class objects
```

Defining a Multilevel Hierarchy (Contd.).

// The following block decides what type of object is passed to this function.

// We test whether the object obj is really an instance of Director class or Manager class or Employee class.

```
if (obj instanceof Director)
    System.out.println  (" Director Salary 30000$");
else if (obj instanceof Manager)
    System.out.println   (" Manager Salary 20000$");
else if (obj instanceof Employee)
    System.out.println   (" Employee Salar  10000$");
else  System.out.println(" INVALID");
}
```

What will happen,
if it is tested like this ?:
First Employee,
then Manager,
then Director.

Defining a Multilevel Hierarchy (Contd.).

```
  public static void main(String ss[])
  {
  System.out.println(" Employee object e is created
");
  Employee e =  new Employee();
  System.out.println("  Manager object m is created
");
  Manager m = new Manager();
  System.out.println("  Director object d is
created  ");
  Director d = new Director();
```

Defining a Multilevel Hierarchy (Contd.).

```
  System.out.println("  salary(e) is called ; ");
salary(e);
System.out.println("  salary(m) is called ; ");
salary(m);
System.out.println("  salary(d) is called ; ");
salary(d);
} // end of main
} // end of class
```

What is the output?

Summary

In this session, you were able to:

- Apply Encapsulation and Abstraction in java code
- Describe Java’s inheritance model and its language syntax
- Describe the usage of the keyword **super**
- Define a multilevel hierarchy

References

1. Gosling, J and others. (1996). *Java Language Specification.* Ed 3. Sun Microsystems, Inc. Retrieved on Feb 25, 2012, from, http://java.sun.com/docs/books/jls/third_edition/html/lexical.html

Thank You