Lab-06
Inheritance and Polymorphism in Java

Objectives:
Understanding the concept of inheritance, the superclass and subclass and polymorphism

Theory:
Inheritance
Inheritance is one of the cornerstones of object-oriented programming because it allows the creation of hierarchical classifications. Using inheritance, you can create a general class that defines traits common to a set of related items. This class can then be inherited by other, more specific classes, each adding those things that are unique to it. In the terminology of Java, a class that is inherited is called a **superclass**. The class that does the inheriting is called a **subclass**. Therefore, a subclass is a specialized version of a superclass. It inherits all of the instance variables and methods defined by the superclass and add its own, unique elements.

Multilevel Inheritance
You can build hierarchies that contain as many layers of inheritance as you like. As mentioned, it is perfectly acceptable to use a subclass as a superclass of another. For example, three classes called A, B, and C, C can be a subclass of B, which is a subclass of A. When this type of situation occurs, each subclass inherits all of the traits found in all of its superclasses. In this case, C inherits all aspects of B and A.

Polymorphism
Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.
Any Java object that can pass more than one IS-A test is considered to be polymorphic. In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.

Method Overriding
In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its superclass, then the method in the subclass is said to override the method in the superclass. When an overridden method is called from within a subclass, it will always refer to the version of that method defined by the subclass. The version of the method defined by the superclass will be hidden.

There are situations when a superclass is created that only defines a generalized form that will be shared by all of its subclasses, leaving it to each subclass to fill in the details. Such a class determines the nature of the methods that the subclasses must

implement.

Lab Task:

```
// This program uses inheritance to extend Box.
class Box {
double width;
double height;
double depth;
// construct clone of an object
Box(Box ob) { // pass object to constructor
width = ob.width;
height = ob.height;
depth = ob.depth;
}
// constructor used when all dimensions specified
Box(double w, double h, double d) {
width = w;
height = h;
depth = d;
}
// constructor used when no dimensions specified
Box() {
width = -1; // use -1 to indicate
height = -1; // an uninitialized
depth = -1; // box
}
// constructor used when cube is created
Box(double len) {
width = height = depth = len;
}
// compute and return volume
double volume() {
return width * height * depth;
}
}

// Here, Box is extended to include weight.
class BoxWeight extends Box {
double weight; // weight of box

// constructor for BoxWeight
BoxWeight(double w, double h, double d, double m) {
```

```
width = w;
height = h;
depth = d;
weight = m;
}
}
class DemoBoxWeight {
public static void main(String args[]) {
BoxWeight mybox1 = new BoxWeight(10, 20, 15, 34.3);
BoxWeight mybox2 = new BoxWeight(2, 3, 4, 0.076);
double vol;
vol = mybox1.volume();
System.out.println("Volume of mybox1 is " + vol);
System.out.println("Weight of mybox1 is " + mybox1.weight);
System.out.println();
vol = mybox2.volume();
System.out.println("Volume of mybox2 is " + vol);
System.out.println("Weight of mybox2 is " + mybox2.weight);
}
}
```

Multilevel Inheritance

```
//edit class BoxWeight
 class BoxWeight extends Box {
double weight;
// construct clone of an object
BoxWeight(BoxWeight ob) { // pass object to constructor
super(ob);
weight = ob.weight;
}
// constructor used when all dimensions specified
BoxWeight (double w, double h, double d,double m)
{
super (w,h,d);
weight = m;
}
// constructor used when no dimensions specified
BoxWeight() {
super();
weight = -1; // use -1 to indicate an uninitialized box
}
// constructor used when cube is created
BoxWeight(double len,double m) {
```

```
super(len);
weight=m;
}
}

// Add shipping costs.
class Shipment extends BoxWeight {
double cost;
// construct clone of an object
Shipment(Shipment ob) { // pass object to constructor
super(ob);
cost = ob.cost;
}
// constructor when all parameters are specified
Shipment(double w, double h, double d, double m, double c) {
super(w, h, d, m); // call superclass constructor
cost = c;
}
// default constructor
Shipment() {
super();
cost = -1;
}
// constructor used when cube is created
Shipment(double len, double m, double c) {
super(len, m);
cost = c;
}
}

class DemoShipment {
public static void main(String args[]) {
Shipment shipment1 = new Shipment(10, 20, 15, 10, 3.41);
Shipment shipment2 = new Shipment(2, 3, 4, 0.76, 1.28);
double vol;
vol = shipment1.volume();
System.out.println("Volume of shipment1 is " + vol);
System.out.println("Weight of shipment1 is " + shipment1.weight);
System.out.println("Shipping cost: $" + shipment1.cost);
System.out.println();
 vol = shipment2.volume();
System.out.println("Volume of shipment2 is " + vol);
System.out.println("Weight of shipment2 is " + shipment2.weight);
```

```
System.out.println("Shipping cost: $" + shipment2.cost);
}
}
```

Lab Assignment:

1. Design a class named Person and its two subclasses named Student and Employee. Make Faculty and Staff subclasses of Employee.

(The Person, Student, Employee, Faculty, and Staff classes)

A person has a name, address, phone number, and email address. A student has a class status (freshman, sophomore, junior, or senior). Define the status as a constant. An employee has an office, salary, and date hired. A faculty member has office hours and a rank. A staff member has a title. Override the toString method in each class to display the class name and the person's name.

2. Design a test program that creates a Person, Student, Employee, Faculty, and Staff, and invokes their toString() methods.

Conclusion: