Cmp Sci 187: Programming with Data Structures

Discussion section: **Object Class Hierarchy**

1. Constructors, Typecast, Extension:

```
public class X {
 private int capacity;
 public X()      { capacity = 16;}
 public X(int i) { capacity = i;}
 public int getCapacity() { return capacity; }
}

public class Y extends X {
 private double loadFactor;
 public Y(double d) { this.loadFactor = d;}
 public getLoad() {return loadFactor;}
}

public static void main(String [] args) {
  X x = new X(32);              //1
  Y y = new Y();                //2
  Y y2 = new Y(0.25);           //3
  x = (X) y;                    //4
  x.getCapacity();              //5
  x.getLoad();                  //6
  y = (Y) x;                    //7
}
```

In the main method what happens when the constructor Y() is invoked?

What happens when the constructor Y(0.25) is invoked?

Is the typecast used in line 4 correct? will it succeed?

Will the main method compile correctly? (lines 5, 6)

Is the typecast in line 7 correct, will it succeed?

Polymorphism: Overloading and Overriding

```
public class X {
 private int [] data;
 public X(int [] data) { this.data = data;}
 public int [] sort() {...}
 public void print() {}
}

public class Y extends X {
 public Y(boolean direction) {}
 public int [] sort(boolean descending) {...}
 public void print() {}
}
```

Does the sort method in class Y override the sort method in class X?

Does the print method in class Y override the print method in class X?

Is the print method in class X polymorphic? In class Y?

Is the sort method in class X polymorphic? In class Y?

Visibility:

```
public class X {
 public void u (p());
 private void p();
}
public class Y extends X {
 private void p();
}
Y y = new Y();

y.u();
```

Which p() method is invoked when u() is called? What would happen if p() were declared public both in X and Y ?

Hierarchy:

```
public class X extends Y {}
public class Y extends X {}
```

Would the above code compile? Why?

```
public class X {}
public class Y extends X {}
public class Z extends X { private Y; }
```

What pattern does the above fragment implement? Why?

```
public class X {}
public interface Y {}
public class Z extends X implements Y {}
```

What is the difference between the above fragment and the one before it?

```
public interface X {}
public interface Y extends X {}
public class Z implements Y {}
```

What is the difference between the above fragment and the one before it?

2. Single Inheritance: Consider the case of the accounts department of a business. It uses a software package to manage travel expenses of the employees. To provide for different modes of travel it models a Travel interface, from the perspective of the accounts department, and two concrete classes that implement the interface for travel by air and by car. Why do you need a Travel interface? Sketch a possible Travel interface and outlines of the two classes. What makes this design flexible? What happens if we have to make changes to the Travel interface ? It allows the employees to fill in travel details using a web page and internally uses the model to compute the compensation due.

3. IS-A and Has-A Relationship*:* A mail order business sells catalog merchandise all over the country. The rules for taxation on the merchandise vary from state to state. Also, the rate of taxation can be different based on the type of merchandise. For simplicity, we will consider three types of merchandise, clothing, pharmaceutical, and beauty products. Suggest a class hierarchy to model the tax on a merchandise.

2. *Travel Hierarchy*

The reason why we need the Travel interface is so that all the clients of this functionality, (the ability to compute the reimbursement to the employee) can utilize the interface and do not have to worry about the complexity involved in computing it for various modes.

Code Snippet for Travel Hierarchy

```
public abstract class Travel {
     public Travel (String source, String Destination) {}
     public double cost() {} // cost of travel, fare
     public int    time() {} // travel time for meals, days  away
     public int    compensation() {} // compute dollar amount
}
public class AirTravel extends Travel {
   public AirTravel(String source, String destination,
                    String airline) {
     super (source, destination);
   }
   public AirTravel(String source, String destination,
                    String airline,  String economyOrFirst) {
        super (source, destination);
   }
   public double cost() {
     // verify cost from airline website
   }
   public int time() {
   }

   public int compensation () {
     // compute allowable parking based on num days at airport
     // or allow taxi cost to airport
   }
}
public class CarTravel extends Travel {
   public CarTravel(String source, String destination, String route) {
        super (source, destination);
   }
   public double cost() {
     // compute mileage
     // compute parking cost
     // based on a per mile cost, compute total cost
   }
   public int time();
   public int compensation () {
   // compute allowable parking based on num days spent a destination
   }
}
```

3. Solution based on Composition:

```
public abstract class Merchandise {
 Tax tax;
 public int getCost() {}
 public int getTax(int zipCode) {return tax.getTax(zipCode);}
}
public class Clothing extends Merchandise {
  public Clothing () { tax = new ClothingTax(this);}
  public int getCost() {}
}
public abstract class Tax {
 Merchandise article;
 public int getTax(int zipCode);
}
public class ClothingTax extends Tax {
 // imagine a zipcode indexed table for looking up taxation
 public ClothingTax(Clothing article) { this.article = article;}
 public int getTax(int zipCode);
}
public class PharmaTax extends Tax {
    public int getTax(int zipCode);
}
```

We may want to model zip code explicitly using a Location class.