COP 3330 Suggested Exercises for Week 8

Polymorphism

1) Imagine adding a ThreeDCC that extended ColorCoordinate, and adding two equals methods to it: one that takes in a Coordinate and another that takes in a ThreeDCC. (You can actually add this fairly easily.)

In this situation, identify the equals methods that get called in the following sequence of code:

```
Coordinate test = new ColorCoordinate(3,'a',"Green");
ColorCoordinate red = new ThreeDCC(3,'a',10,"Red");
ThreeDCC blue = new ThreeDCC(3,'a',79,"Blue");
Coordinate nocolor = new Coordinate(3,'a');

test.equals(red);
red.equals(test);
blue.equals(red);
red.equals(blue);
nocolor.equals(red);
nocolor.equals(test);
blue.equals(test);
```

2) Actually write a small ThreeDCC class and test out your answers. You don't need to write a meaningful equals method, just enough to test the method calls above.

3) Add in yet another method called equals to the ThreeDCC class that takes in a ColorCoordinate. Write a line of code in main that runs differently when you add this method versus when you omit it from your code base.

Interfaces

4) Pick another shape (maybe a regular hexagon) and have it implement the Shape2D interface. Test your shape out by adding it as one of the shapes in TestShape2D.java.

5) Create an InfiniteSequence interface which requires the following method:

```
public int getNextTerm(int curTerm);
```

Any class that implements this interface must implement a method with the signature above, which, when given an integer, returns the next term in that object's InfiniteSequence that follows curTerm.

Create your own class that implements this interface and define your own getNextTerm method for this class. Test your object out.

6) Create an object from the class you created and write a segment of code that generates 1000 terms of your sequence, starting with a seed of curTerm = 1 and computes the sum of those 1000 terms.

Comparable Interface

7) The Time_V1 class already has a compareTo method. Just make this class implement Comparable<Time_V1> and test out sorting several Time_V1 objects.

8) Take GiftCard class and add a compareTo method to it and have the class implement Comparable<GiftCard>. Have the compareTo method return a negative integer if this object is worth less money than other (formal parameter) and a positive integer if this object is worth more money than other. If both are worth the same, decide how you want to break the tie. Test your method out by creating an array of GiftCard, sorting it and printing out the results to prove that your compareTo code was used.

9) Take the Coordinate class and add a compareTo method to it and have it implement Comparable<Coordinate>. Sort Coordinates by num (smaller value of num comes first), breaking ties with the Ascii values of the character c. Here is a list of Coordinates in sorted order via this metric: `(3, 'Z'), (3, 'a'), (3, 'p'), (4, ' '), (4, 'A'), (1000, 'C'), (1001, 'B')`. Test your code!

10) Come up with your own class and give its objects an ordering based on the compareTo method you define. Test your method by sorting an array of objects of the class that you created.