CSCI 3353 Object Oriented Design
Homework Assignment 5
Due Monday, October 19

The first three parts of this assignment ask you to examine the Java API documentation to learn about how to add borders to Swing components. In particular, I want you to focus on the APIs for the interface *Border* and the classes *TitledBorder*, *CompoundBorder*, *LineBorder*, and *BevelBorder*. Note that borders are added to Swing components via the method *setBorder* in class *JComponent*.

1. Write a Java program, called *BorderTest*, that displays four text fields on a panel, with each text field having one of the following borders:

a) A 2-colored border:

b) A 3-colored border:

c) A 2-colored border with a title on the outside border:

d) A 2-colored border with a title across both borders:

2. Consider the classes *Border*, *TitledBorder*, *CompoundBorder*, *LineBorder*, *BevelBorder*, and *JComponent.*

a) Draw a class diagram depicting the connections between these classes.

b) Explain where the *strategy* design pattern is used in this diagram.

c) Explain where the *decorator* design pattern is used. In particular, which classes are the base classes? Which classes are the decorators?

3. There are two ways that Java could have designed borders using the decorator pattern. The first way is to decorate the border: Here, there will be base border classes, and classes that decorate a border to create a fancier border. The second way

is to decorate the component: Here, there will be base components, and classes that decorate a component to create a bordered component.

a) Which way did Java choose? Briefly explain.

b) Give a class diagram corresponding to the other way of doing the design.

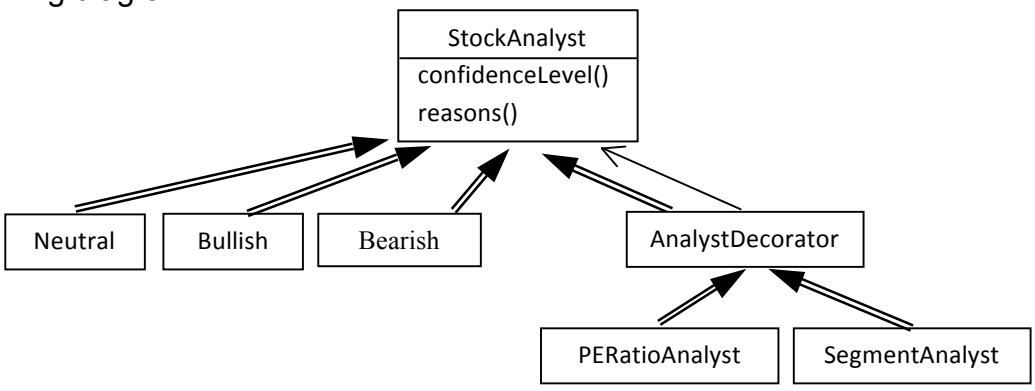
c) Using your diagram for (b), give the code that would be required to create the 2-color bordered text field from problem 1(a). Note that you don't have to write the code to implement the various border classes. You just have to write the code to use them, assuming that they existed. (Consequently, your code will neither compile nor run.)

4. Your task is to write an expert system for analyzing stocks. The system contains numerous classes, called *stock analysts*, whose job is to make predications based on a particular aspect of a company. For example, there might be an analyst that examines the earnings history, one that makes a predication based on the price-earnings ratio, one that understands the broad trends in different market segments (e.g. "auto stocks are bad; technology stocks are good"), and so on.

Each *StockAnalyst* class has two methods:

- the method *confidenceLevel*, which returns a value between 0.0 and 1.0 that denotes the probability that the stock will rise in value;
- the method *reasons*, which returns a string explaining the reasons for this value.

The stock analyst classes are organized according to the decorator pattern, as in the following diagram:



The box for *StockAnalyst* indicates that it should support the methods *confidenceLevel* and *reasons*. The base classes are *Neutral*, *Bullish*, and *Bearish*. Their confidence levels are based on the market as a whole. Assume that *Neutral* always gives a confidence level of 50%, *Bullish* 60%, and *Bearish* 40%.

Each decorator class derives its confidence level by examining a particular aspect of the company. For example, the *PERatioAnalyst* class calculates the Price-Earnings ratio (by dividing the current price by current earnings). Assume that a PE ratio of 12 returns a confidence level of 50%; a smaller ratio returns a higher confidence level, and a higher ratio returns a lower confidence level. You can decide how to do this.

The *SegmentAnalyst* class has a preset confidence level for certain market segments. For example, assume that auto stocks have a confidence of 20% and technology stocks have a confidence of 80%. For other stocks, the analyst has no opinion.

In general, a decorator analyst may not be supplied with all of the information about a company that it needs. In that case, the analyst contributes nothing to the overall confidence level.

When a decorator analyst is able to compute a confidence level, it combines that value with the confidence level of the analyst it decorates. In this assignment, assume that it takes the average of its confidence with that of its component confidence. (What this means is that if you have 3 analysts chained together, the order in which they are composed can affect the final confidence level. Consider this a feature, not a bug.)

The relevant information about a company should be stored in a text file. For an example, download the file *appleInfo.txt*. It has the following contents:

```
name apple
hq california
earnings 8.5
marketsegment  technology
shareprice 110
```

Each line of this file denotes a (keyword, value) pair. The first word of the line is the key, and the remaining word denotes the value of that key. You should download the class file *StockInfo.java*, which processes such a file. Its constructor takes the name of this file as an argument. It reads the entire file and save the (key,value) pairs in a map. The class has an accesssor method that you can use, which returns the value associated with a specified key.

The *StockInfo* class assumes that the company file is located in the java class path. If you are using Eclipse, the class path is defined to be home directory of the project. This means that if your code is in a package (say, "hw.hw5"), then the company file will not be in the same directory as your source code. In particular, suppose that your project is

named "oodesign". Then your source code will be in the folder "hw5" which is in the folder "hw" which is in folder "oodesign"; but your file it will be in the folder "oodesign".

Your main class should be named *HW5Analyst*. It has three tasks. First, it creates a *StockInfo* object corresponding to a specified stock-information file. Second, it constructs a chain of one or more analyst objects, passing the input data into the constructor of the base class. Finally, it calls the *confidenceLevel* and *reasons* methods of the outermost decorator object, and prints them. For an example, download my *HW5Analyst.java* file. You should be able to execute this code using the classes you wrote.

WHAT TO SUBMIT: You should submit three files to Canvas:

- the file *BorderTest.java*, for problem 1;
- a file containing the answers to problems 2 and 3;
- a zip file containing the java files needed to execute problem 4.