ADVANCED JAVA PROGRAMMING AJP-P3-2012-2013

INTRODUCTION

This exercise will test your knowledge of the *multiton* and *observer* design patterns. You will also learn about *regular expressions*.

SET-UP

Import the *Netbeans* project folder named 'AJP-P3-2012-2013-STUDENT'. Open it up. You have been given a set of unit tests. Initially, there will be LOTS of syntax errors in these unit tests. This is expected. The errors will disappear as you write the required classes.

INSTRUCTIONS FOR BRONZE TASK

In the *src* folder of your new project, in the package named *uk.ac.tees.bronze.username* create a file called *Share.java*. This class represents one stock market share in a company. Your class should satisfy the following criteria.

- The *Share* class should have a *price* instance variable that records the price of the share on the stock market in pounds and pence. There should be a setter (mutator) and getter (accessor) method for this instance variable.
- Your class will be an implementation of the *multiton* design pattern. This means that there will only ever be one unique *Share* instance per named *key*.
- Instances of the class should be kept in a map.
- The key to the map will be the unique three letter name of the company e.g. BBC, AXA. This name is always capitalised.
- A call to the static method *Share.getInstance()* that passes a three letter string, all capitals, should succeed i.e. the *getInstance*() method will return a *Share* object.
- A call to the static method *Share.getInstance()* that passes an empty string should fail i.e. the *getInstance*() method will return null.
- A call to the static method *Share.getInstance()* that passes a *null* should fail i.e. the *getInstance*() method will return null.
- A call to the static method *Share.getInstance()* that passes a string containing something other than a three latter string, all capitals, should fail i.e. the method will return null.
- If the *getInstance*() method is called for a company that is not currently in the map, create a new *Share* object with a default value of 1.00 and add it to the map.
- If the *getInstance*() method is called for a company that is already in the map, return the pre-existing *Share* object.

The unit tests I have provided will guide your solution. You will probably need to use regular expressions when checking the validity of the company name. Read about them here: http://docs.oracle.com/javase/tutorial/essential/regex/

Make sure that your solution to the BRONZE task does not have any *CheckStyle* errors. You cannot pass this task if your code contains *CheckStyle* errors.

INSTRUCTIONS FOR SILVER TASK

In the *src* folder of your new project, in the package named *uk.ac.tees.silver.username* create a file called *Share.java*. This class represents one stock market share in company X. Your class should satisfy the following criteria.

- The *Share* class should have a *price* instance variable that records the price of the share in company X in pounds and pence. There should be a setter (mutator) and getter (accessor) method for this instance variable. The default value for this variable is 1.00.
- Your *Share* class will be part of an implementation of the *observer* design pattern.
- The state of a *Share* object (i.e. it's price) is watched by other objects who implement the *ShareWatcher* interface. When the unit price of this share changes, these watchers should be informed. The *Share* class should maintain a list of all *ShareWatcher* objects who have registered as interested. There should be two methods for list management – *addShareWatcher(ShareWacther sw)* and *removeShareWatcher(ShareWatcher sw).*
- The *ShareWatcher* interface defines one method – *updatePrice(double price).*
- There are two concrete implementations of the *ShareWatcher* interface – *StockBroker* and *BankManager*. You must write these classes. Both concrete implementations should have an instance variable called *portfolio*, which records the number of shares currently owned. This instance variable should have an accessor method.
- When the *updatePrice*() method is called in a *StockBroker* object, that object must make purchase decisions. A *StockBroker* object will buy shares when the price drops below 2.00. A stock broker will sell shares when the price rises above 3.00. A stock broker buys shares in blocks of 500. These values are passed to the constructor of the *StockBroker* object. When buying/selling shares, the instance variable of this object should be updated.
- When the *updatePrice*() method is called in a *BankManager* object, that object must make purchase decisions. A *BankManager* object will buy shares when the price drops below 1.00. A stock broker will sell shares when the price rises above 4.00. A stock broker buys shares in blocks of 100. These values are passed to the constructor of the *BankManager* object. When buying/selling shares, the instance variable of this object should be updated.

The unit tests I have provided will guide your solution. Make sure that your solution to the SILVER task does not have any *CheckStyle* errors.

INSTRUCTIONS FOR GOLD TASK

In the *src* folder of your new project, in the package named *uk.ac.tees.gold.username,* change your solution to the SILVER task so that

- Fields and methods that are shared by *BankManager* and *StockBroker* are moved into *ShareWatcher*, which changes from an interface to an abstract class.
- In the *ShareWatcher* class, declare a new instance variable called *balance*, which records the current financial status of the *ShareWatcher* object. To calculate the *balance* variable, simple calculate:

  ```
  Total expenditure on shares – total income from selling shares
  ```

  For example, if I bought 100 shares at 1.20, and sold them at 3.10, my balance should be 190.00. Note that a balance can be a minus figure.
- Modify the *BankManager* class so that a *BankManager* object never exceeds a balance of -500.00
- Modify the *StockBroker* class so that a *StockBroker* object never has a portfolio of more than 3000 shares.

The unit tests I have provided will guide your solution. Make sure that your solution to the GOLD task does not have any *CheckStyle* errors.