ADVANCED JAVA PROGRAMMING AJP-P5-2012-2013

INTRODUCTION

This exercise will test your knowledge of the *strategy* and *state* design patterns. You will also gain some experience using third party libraries (an important skill for software engineers).

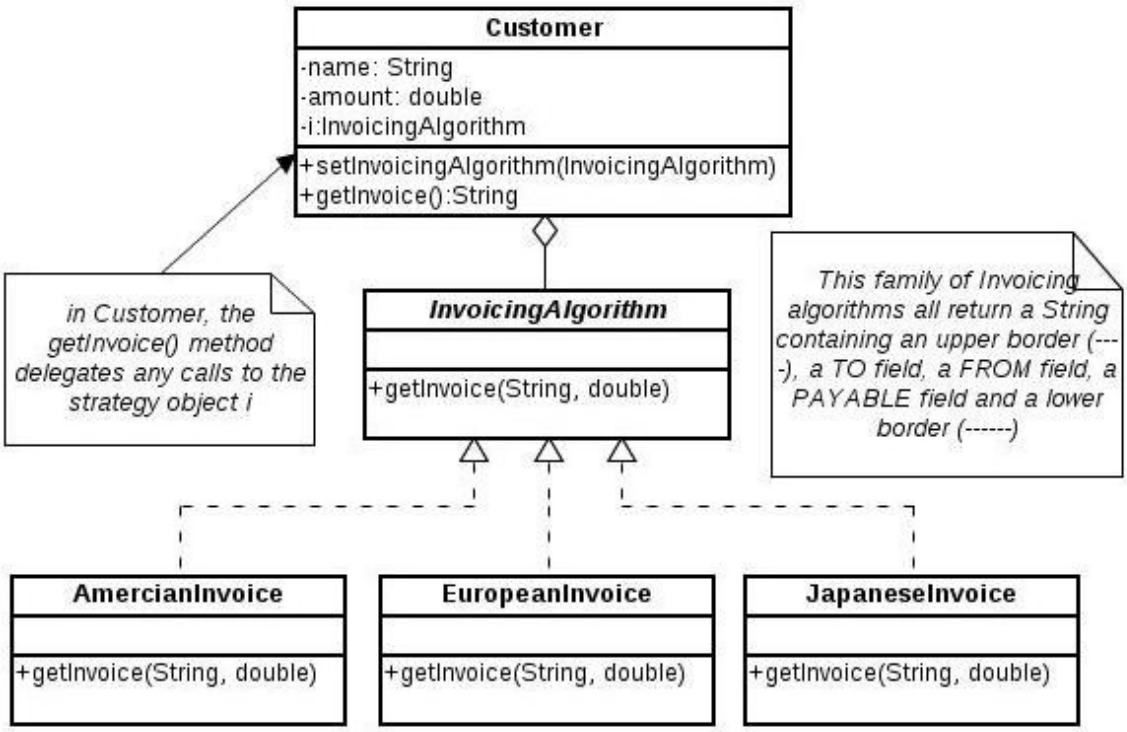
SET-UP

Import the *Netbeans* project folder named 'AJP-P5-2012-2013-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

You work in the invoice department at *EasyFlap* airways. Your job involves sending invoices to commercial customers who use the airport. Most of these customers are international companies. The invoices you send them should be *localised* (i.e. an invoice sent to America should be in dollars).

In the *src* package *uk.ac.tees.bronze.username,* write a *Customer* class with a *getInvoice*() method that can produce invoices in different currencies. Your solution must use the *Strategy* pattern. Use the unit tests and the UML diagram below as a guide:



In your application, use the following currency conversion rates:

UK / AMERICA	1: 1.57
UK/ EUROPEAN	1: 1.14
UK/ JAPAN	1: 121

You should use '\u00a5' to produce the Yen symbol. You should use '\u20AC' to produce the Euro symbol.

You will need to use the *DecimalFormat* class to format your currency values to two decimal places.

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,* modify your solution to the BRONZE task so that the *getInvoice*() method produces neat, business-like invoices in PDF format.

You will need to use this 3rd party library http://itextpdf.com/ The JAR you need is itextpdf.5.3.4.pdf

Here are some instructions for adding a JAR file to your NetBeans project.

Once you have added the iText JAR file to your project, you will be able to use classes from that JAR file in your solution e.g. *PdfWriter*. Import statements will have to be added to your code as normal.

You can look at some example code here. I recommend you start with *HelloWorld*.

There are no unit tests for this task. The output of your application will be checked manually by Mark.

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

INSTRUCTIONS FOR GOLD TASK

A railgun is an electrically powered electromagnetic projectile launcher. In this exercise you will develop a java class that describes a rail gun, and other classes which describe the various states a railgun can be in. The states are as follows:

Normal– The railgun can move and fire without restriction.

Damaged– The rail gun cannot move and can only fire at a reduced rate (50% of requested rounds).

NeedAmmo– The rail gun can move without restriction, but it has no ammunition and therefore cannot fire.

In the *src* folder of your new project, in a package named uk.ac.tees.gold.username, create the following files:

- Railgun.java
- RailgunState.java
- NormalState.java
- DamagedState.java
- NeedAmmoState.java

Please note the following points:

- *RailgunState* is an interface which is implemented by *NormalState*, *DamagedState* and *NeedAmmoState*
- *RailgunState* defines two methods, *fire*() and *move*(). You must determine the parameters for these methods.
- A newly instantiated *Railgun* should be in a *NormalState* with full ammunition (10 rounds).
- When a *Railgun* in a *NormalState* has fired all of its ammunition, it should move to the *NoAmmoState*.
- When a *Railgun* in a *DamagedState* has fired all of its ammunition, it should move to the *NoAmmoState*.
- When a *Railgun* in a *NoAmmoState* visits the ammo dump, it should move to a *NormalState* and its ammunition level should be replenished.
- When a *Railgun* in a *NormalState* visits the ammo dump, its ammunition level should be replenished.
- The location of the ammo dump is (320,43)
- The Railgun class has the following instance variables
  - *int* ammo
  - *Point* position
  - *RailGunState* state
- The *Railgun* class should have a method called *fire(Point p, int rounds)* where
  - *Point* p is the coordinates of the target
  - *int* rounds is the number of rounds HQ has requested
  - This method should return a *String* object indicating Success, Partial Success or Failure
  - *Success* occurs when the requested number of rounds are fired at the target.
  - *Failure* occurs when zero rounds are fired at the target.
  - *Partial* success is anything in-between.
- The *Railgun* class should have a method called *move(Point p)* where
  - *Point* p is the destination
  - This method should return a *String* object indicating Success or Failure

    - *Success* occurs when a *Railgun* can move to the specified location
    - *Failure* occurs when a *Railgun* cannot move to the specified location
- All state transitions must occur within other states. For example, the *Railgun* can only move from *NormalState* to *NeedAmmoState* state within *NormalState.fire(),*

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. You cannot pass this task if your code contains *CheckStyle* errors.