ADVANCED JAVA PROGRAMMING AJP-P7-2012-2013

INTRODUCTION

This exercise will test your knowledge of the *decorator*, *adapter* and *factory* patterns.

SET-UP

Import the *Netbeans* project folder named 'AJP-P7-2012-2013-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

EasyFlap is a popular UK airline offering basic flights at low prices.

- India costs £212
- Sri Lanka costs £254.
- Botswana costs £154

Customers purchasing a flight with *EasyFlap* can choose to upgrade their basic flight using the following 'menu'. Each upgrade incurs an additional cost, as follows:

First class upgrade	£217
Priority boarding upgrade	£19
Web check in upgrade	£27
Refreshments upgrade	£12
Entertainment upgrade	£9.50

In this exercise you will develop a set of classes that calculate the correct price for a *flight package*. A flight package is a basic flight to India, Sri Lanka or Botswana, plus any combination of the upgrade options shown above.

Note that the price of an upgrade is the same irrespective of the flight destination (i.e. first class costs £217 whether you are flying to Botswana, India or Sri Lanka.

Use the provided unit tests to guide your solution. You must use the *Decorator* pattern in your solution.

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.

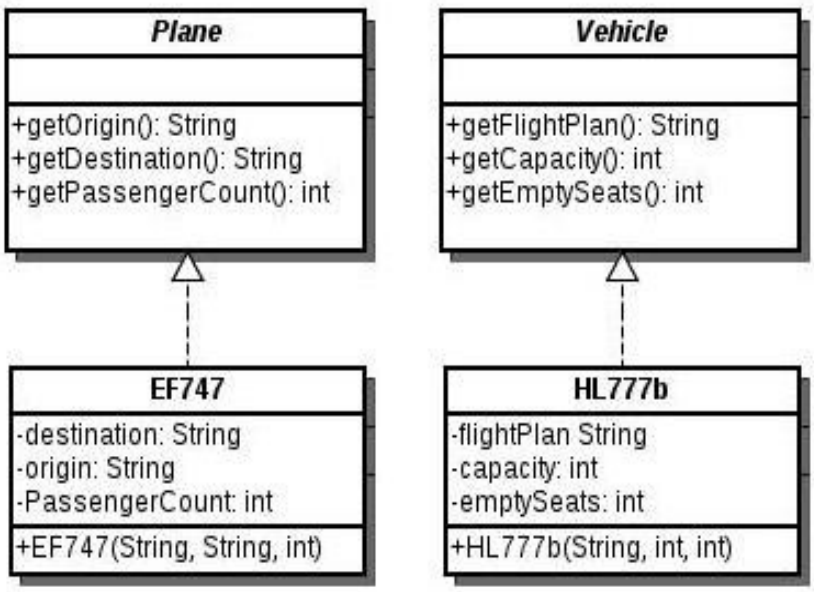
INSTRUCTIONS FOR SILVER TASK

EasyFlap has recently bought out one of its UK competitors, a company called *HairLingers*. Unfortunately, the *HairLingers* software library is not compatible with the *EasyFlap* software library. This is causing all sorts of delays and complications.

In this part of the assignment you will use the adapter design pattern to implement a solution to this problem.

You will now develop the following 2 classes and 2 interfaces:

- **Plane** – All *EasyFlap* classes implement this interface.
- **EF747** – a concrete implementation of the *Plane* interface, describing the EF747 commercial jet plane. All *EasyFlap* planes are initialized using the following 3 parameters:
  - *Origin* – in lower case characters
  - *Destination* – in lower case characters
  - *Passenger count*
- **Vehicle** - All *HairLingers* classes implement this interface.
- **HL777b** – a concrete implementation of the *Vehicle* interface, describing the HL777b commercial jet plane. All *HairLingers* planes are initialized using the following 3 parameters:
  - *Flight plan* – in upper case characters, in the format “X-Y”, where X is the origin and Y is the destination e.g. “SWANSEA-SCUNTHORPE”
  - *Capacity* – The total number of seats on the plane
  - *Empty seats* – The number of empty seats on take off.



The UML diagram above and 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. You cannot pass this task if your code contains *CheckStyle* errors.

INSTRUCTIONS FOR GOLD TASK

a. In a sub package called uk.ac.tees.username.gold.simplefactory, write a solution to the following problem.

Techno-Byte is a small start-up company that retails computer hardware to local businesses. *Techno-Byte* has a telephone order line which is used by customers. When taking a phone order, *Techno-Byte* employees greet the customer and open the ordering system. Then the employee

- Asks the customer for their maximum spend per machine
- Asks the customer for the number of units required
- Enters these values into the ordering system
- Talks the customer through the deal suggested by the ordering system

The 'deal' suggested by the ordering system will be the *Techno-Byte* machine that is as close as possible to the maximum spend per machine specified by the customer, without exceeding it. For example, if the customer can spend £450 per unit and *Techno-Byte* have three different grades of machine, priced at £300, £400 and £500 respectively, the ordering system will suggest a deal based on the £400 units.

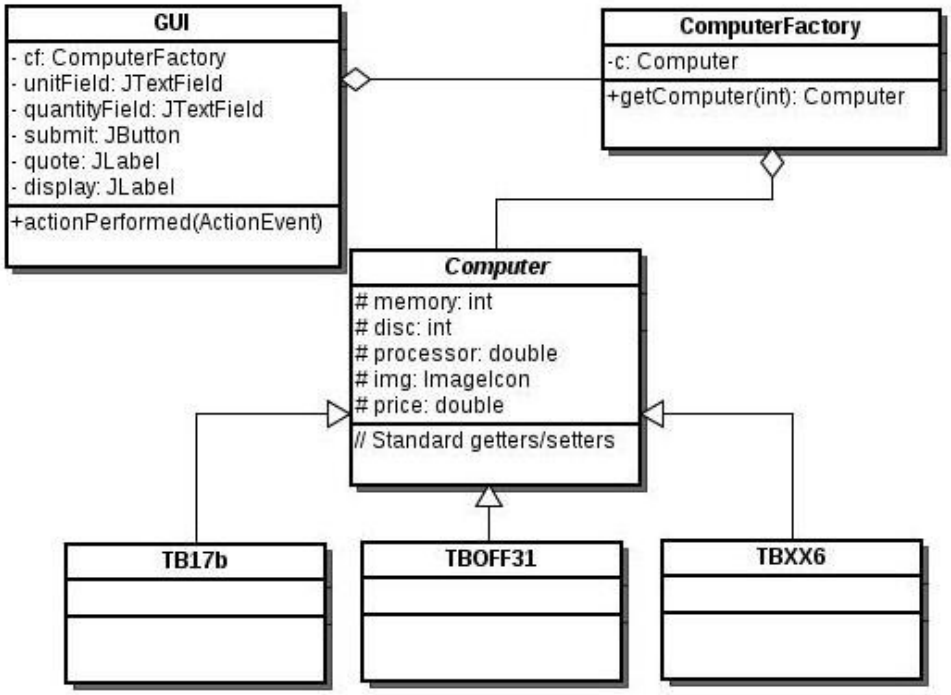
In this part of the exercise you will design and build the following:

- An entry point for the application called *TestGUI*, which has a main method.
- A graphical user interface for the ordering system, called *GUI*
- The back-end of the ordering system, comprised of the following classes
  - *Computer*, an abstract class representing all computers sold by Techno-Byte
  - *TB17b*, *TBOFF31* and *TBXX6*, three concrete classes representing thee different grades of computer sold by Techno-Byte
  - *ComputerFactory*, a simple factory that creates *Computer* objects. This class has a factory method that takes an int (maximum spend), and returns a *Computer* object (as described above). If the maximum spend is lower than the lowest price machine, it returns null.

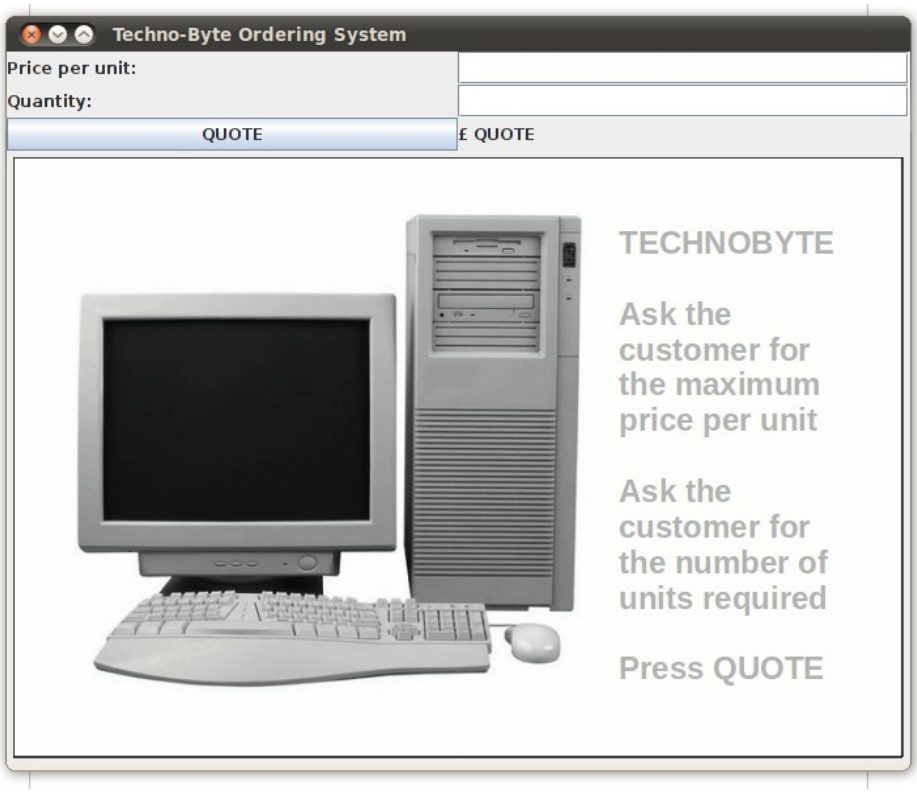
The specifications and prices of the three computers are as follows:

TB17b: 2GB SDRAM, 500GB HDD, 2.60 GHz £299
TBOFF31: 4GB SDRAM, 750GB, 3.00 GHz £449
TBXX6: 8GB SDRAM, 1TB, 3.40GHZ £599

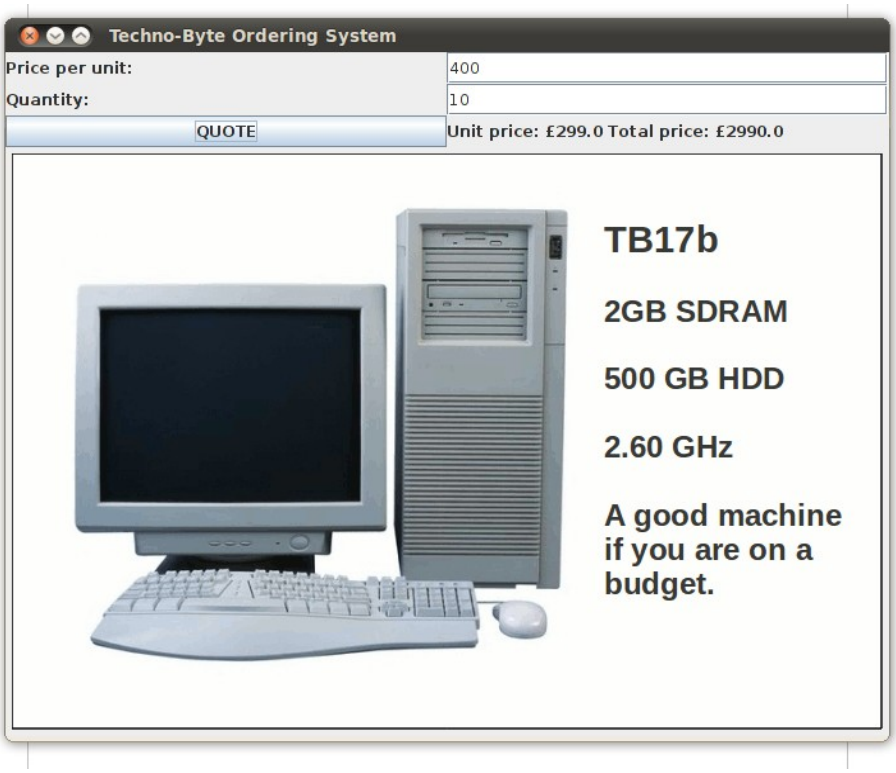
Use the following UML diagram as a guide:



The GUI should initially look like this:



If the user types in numeric values for 'price per unit' and 'quantity' then the display should change as follows:



You should validate the data in the text fields using the Integer wrapper class. If the user types in non-numeric values for either 'price per unit' or 'quantity', set both text fields to read “Please enter numerical value”.

If the factory method returns a *null* value (i.e. the maximum spend is lower than the lowest priced machine), then change the 'unit price' *JLabel* to “No machines in that price range” and reset the image to DEFAULT.gif.

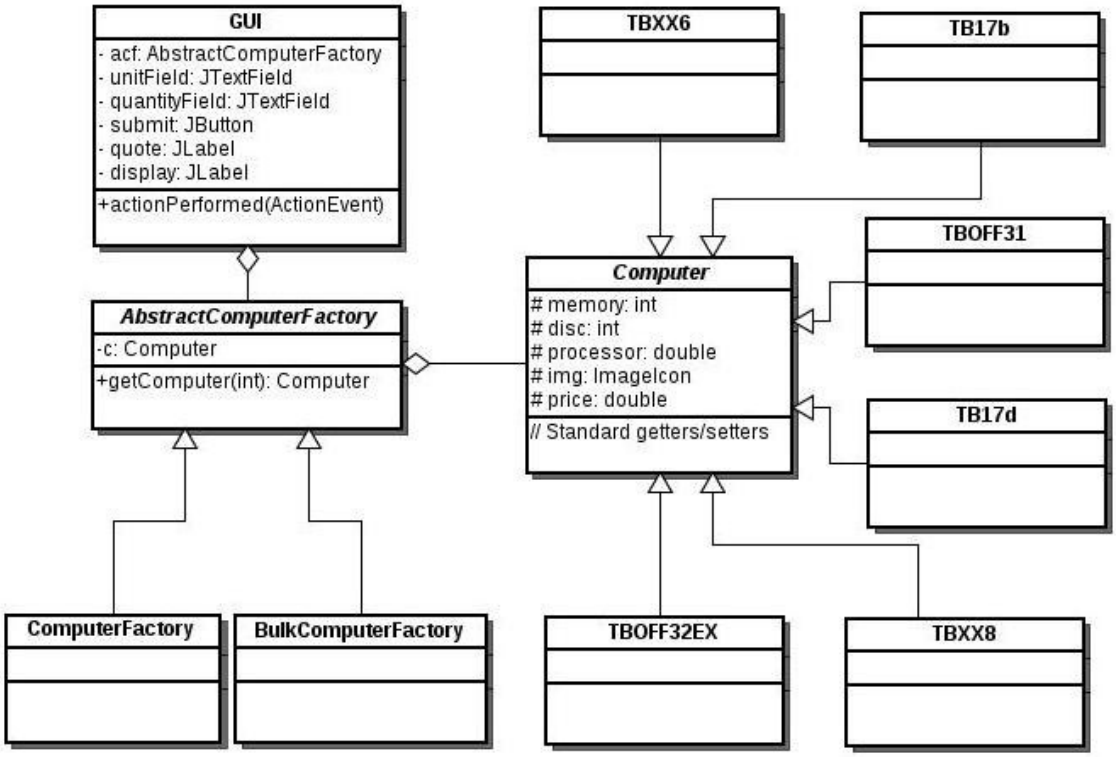
There are no unit tests for this exercise. Mark will check your work manually. The images toy need have been provided in the ZIP file named *gold-images.zip.*

b. In a sub package called uk.ac.tees.username.gold.factorymethod, write a solution to the following problem.

Techno-Byte is flourishing, and now sells computers to larger customers. These customers are offered significant discounts provided they purchase in bulk (i.e. > 50 units). Here are the specifications of the bulk buy models available at Techno-Byte -

TBOFF32EX - 4GB SDRAM, 750GB, 3.20 GHz £400
TBXX6 - 8GB SDRAM, 1TB, 3.40GHZ £500

Create a second version of your GUI that switches between *ComputerFactory* and *BulkComputerFactory* depending on the number of units requested by the customer. Your solution should implement the GoF factory method pattern. Use the following UML diagram as a guide:



There are no unit tests for this exercise. Mark will check your work manually.

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.

INSTRUCTIONS FOR PLATINUM TASK

This task involves material that we have not covered in class and will require research.

a. Implement a Java class that represents an *EasyFlap* passenger. The information that must be encapsulated by the *Passenger* class is as follows:

- First name of the passenger
- Last name of the passenger
- Passport number (a series of 9 numbers).
- Passenger class.
- Nationality of the passenger, expressed as a three letter code.

EasyFlap has three passenger classes – Economy, Business and First Class.

Due to legal restrictions, EasyFlap only sells tickets to people from the United Kingdom (3-letter code GBR), America (USA), Germany (GER), Botswana (BOT) and India (IND).

Your *Passenger* class should have the normal accessor/mutator methods.

Your class should use enumerations.

In *Passenger*, implement the *Comparable* interface so that you can sort a collection alphabetically by last name.

Now write a unit test that creates 10 *Passenger* objects as follows:

```
ArrayList<Passenger> a = new ArrayList<Passenger>();
a.add(new Passenger("Mark", "Truran", 691192317, PassengerClass.F, Nationality.GBR));
a.add(new Passenger("Derek", "Simpson", 557213131, PassengerClass.F, Nationality.GER));
a.add(new Passenger("Matt", "Damon", 547712126, PassengerClass.E, Nationality.GER));
a.add(new Passenger("Erika", "Downs", 329982763, PassengerClass.B, Nationality.IND));
a.add(new Passenger("Mike", "Lockyer", 459174268, PassengerClass.B, Nationality.USA));
a.add(new Passenger("Simon", "Lynch", 117790344, PassengerClass.E, Nationality.BOT));
a.add(new Passenger("Gwenn", "Bosser", 654626722, PassengerClass.E, Nationality.BOT));
a.add(new Passenger("Zafar", "Khan", 978965446, PassengerClass.E, Nationality.GBR));
a.add(new Passenger("Simon", "Stobart", 564253227, PassengerClass.B, Nationality.GBR));
a.add(new Passenger("Jackie", "Dawes", 112442317, PassengerClass.E, Nationality.USA));
```

Sort the collection using *Collections.sort(List)*, then write the contents of the sorted collection to console using the *toString*() method of the *Passenger* class.

Your unit test should fail unless you get this result:

```
Gwenn Bosser            654626722       E       BOT
Matt Damon              547712126       E       GER
Jackie Dawes            112442317       E       USA
Erika Downs             329982763       B       IND
Zafar Khan              978965446       E       GBR
Mike Lockyer            459174268       B       USA
Simon Lynch             117790344       E       BOT
Derek Simpson           557213131       F       GER
Simon Stobart           564253227       B       GBR
Mark Truran             691192317       F       GBR
```

b. In a seperate class, implement the *Comparator* interface so that you can sort a collection of *Passenger* objects by nationality and passport number. The desired sort order for nationalities is alphabetical (i.e. BOT, GBR, GER, IND, USA). The desired sort order for passport numbers is numerical (i.e. 1, 2, 3 etc.). You should initially sort by nationality, then by passport number.

Now write a unit test that creates 10 *Passenger* objects (use the code above). Sort the collection using *Collections.sort(List, Comparator)*, then write the contents of the sorted collection to console using the *toString*() method of the *Passenger* class.

Your unit test should fail unless you get this result:

```
Simon Lynch             117790344       E       BOT
Gwenn Bosser            654626722       E       BOT
Simon Stobart           564253227       B       GBR
Mark Truran             691192317       F       GBR
Zafar Khan              978965446       E       GBR
Matt Damon              547712126       E       GER
Derek Simpson           557213131       F       GER
Erika Downs             329982763       B       IND
Jackie Dawes            112442317       E       USA
Mike Lockyer            459174268       B       USA
```

There are no unit tests provided for this exercise. You will write your own unit tests. Mark will check your work manually.

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