COSC 436 Dierbach
Fall 2015

ASSIGNMENT 7 – Implementing the Decorator Design Pattern (and Associated Strategy Pattern)

100 pts.

Due Thursday, October 29th (by class time)

PROBLEM

You are to design and implement code based on the Decorator pattern for generating an appropriate receipt for a customer buying items at Best Buy. The general format of a receipt is as follows:

Basic Receipt
Store Header ("Best Buy", store address, store number, phone number)
Date/Time of Sale
Itemized Purchases
Total Sale (without sales tax)
Amount Due (with added sales tax)

Dynamically-Added Items

Sales Tax Computation (based on state residing in)
Optional Secondary Headers,
- "Happy Holidays from Best Buy"
- "Summer Sales are Hot at Best Buy"

Relevant Rebate Forms (to be printed at the end of the receipt)
Promotional Coupons (e.g., "10% off next purchase")

APPROACH

We will assume that the code is written as part of the software used by all Best Buy stores around the country. Therefore, the information in the Store Header will vary depending on the particular store and its location. In addition, the amount of sales tax (if any) is determined by the particular state that the store resides in. It will be implemented by use of the Strategy pattern. Finally, the added items specific to each receipt will be handled by use of the Decorator pattern.

Basic Receipt

The information for the basic receipt should be stored in a Receipt object. The instance variables of a receipt should contain the date of sale as a Java Date type (java.util); an array of PurchaseItem objects; the total sale (without tax) and amount due (with added tax) each of type float. In addition, following the Strategy design pattern, there should be an instance variable of (interface) type StateTax that can be assigned to a specific StateTax object (e.g., MarylandStateTax) if there is a sales tax for the state that the store resides in.

Determining Sales Tax

We will implement the classes so that the receipts can be generated for one of four possible store locations, each in a different state: Maryland (6% sales tax), California (7.5%), Massachusetts (6.25%), and Delaware (no sales tax). Note the following "sales tax holidays" of individual states:

Maryland

Has a sales tax holiday, but does not include computers or computer accessories.

California

No sales tax holidays.

Massachusetts

For the benefit of back-to-school shoppers, there is a sales tax holiday on the second weekend in August (for two days) which includes school supplies, computers, sports equipment, and health and beauty aids. The tax-free days on these items for 2015 will be August 8th and 9th.

Because the determination of sales tax is not just based on the tax rate, but also on the purchase date and purchase amount, is why the Strategy pattern is being used here. If there is no applicable sales tax (e.g., in Delaware) then the reference to a SalesTax object is set to null. Thus, if an item is returned to a store in a different state from which the item was purchased in, the Receipt object retrieved from the system will have associated with it the SalesTax object (method) for the state that the items were purchased.

Adding Additional Receipt Items

During particular times of the year, a receipt header may begin with a special greeting (e.g., "Happy Holidays from Best Buy"), called *secondary headers*. Each possible secondary header is stored as an object containing the header. We assume that each Best Buy store downloads the current decorator objects each morning. Therefore, if a secondary greeting object exists, we assume that it should be used.

Each item in the store has associated with it a unique item code. Since rebates only apply to specific purchases, each Rebate object will contain a Boolean method `applies()` that returns true if the particular rebate applies. If so, then a `getRebateForm` method would be called which returns a single string (containing any number of newline characters) to be displayed at the end of the receipt. A decorator object would therefore be added to the receipt for this.

A similar approach should be taken for the promotional coupons mentioned above. They only apply when a customer has spent a certain amount of money.

ClassLoader in Java

Following is a description of the ClassLoader in Java. It explains how our program could *theoretically* retrieve the needed decorator classes from the Best Buy web site.

> Among commercially popular programming languages, the Java language distinguishes itself by running on a Java virtual machine (JVM). This means that compiled programs are expressed in a special, platform-independent format, rather than in the format of the machine they are running on. This format differs from traditional executable program formats in a number of important ways.
> In particular, a Java program, unlike one written in C or C++, isn't a single executable file, but instead is composed of many individual class files, each of which corresponds to a single Java class. Additionally, these class files are not loaded into memory all at once, but rather are loaded on demand, as needed by the program.
>
> The ClassLoader is the part of the JVM that loads classes into memory … one of the most innovative things about the Java language is that it makes it easy for the JVM to get classes from places other than the local hard drive or network. For example, browsers use a custom ClassLoader to load executable content from a Web site.
>
> -- IBM DeveloperWorks (http://www.ibm.com/developerworks/java/tutorials/j-classloader/section2.html)

Installing the Current Rebate Objects

We will place all the rebate classes in a separate package called `rebates`. We will assume that these classes are retrieved from the Best Buy web site. The program will create an instance of each (using the default constructors) to be added to its list of Rebate objects. Similarly, there will be a separate package called `secondaryHeaders` that are also assumed to be retrieved from the Best Buy web site. The program will also create and maintain a list of these objects. Finally, a separate package, `coupons`, of Coupon classes will be assumed retrieved, and object instances of each stored. Each of these classes should implement an `Applicable` interface, containing only one method: `boolean applies(String[])`. Note that this method is passed an array of strings, so that all the information needed can be provided.

Configuration File

A configuration file will be read by the program at start up, to configure the system for the particular store location, containing the following information:

-- store number
-- store address
-- store phone number
-- stateCode

Factory Class

You must utilize a factory class to properly construct Best Buy receipts based on the information read from the configuration file.

PROGRAM TO CREATE
Create a program that will create of and display Best Buy receipts. The program should provide a main menu as in the following,

1 – Start New Receipt
2 – Add Sales Items
3 – Display Receipt

WHAT TO TURN IN: Each of the source files, submitted as one zipped file.

DUE: Thursday, October 29th by class time. **Late assignments will be marked off 10% for each 24-hour period past the time due.**