COSC 436

Fall 2018

Noah Day

Adam Lynch

# ASSIGNMENT 4 – Implementing the Decorator Design Pattern (with Strategy Pattern)

70 pts.

**Due Tuesday, November 20th**

**Note:** As specified in the course syllabus, you are encouraged to work with one partner on the assignments. Collaborative assignments will receive a 5% bonus credit. If you work collaboratively, you must list your collaborator’s name and provide a short summary of what each person did.

**PROBLEM**

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

Basic Receipt

Store Header (store street address, phone number, state code, store number)

Date of Sale

Itemized Purchases

Total Sale (without sales tax)

Amount Due (with added sales tax)

Dynamically-Added Items

Tax Computation object (based on state residing in)

Optional Secondary Headers (“Greetings”), e.g.,

- “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's location. In addition, the amount of sales tax (if any) is determined by the state that the store resides in. The calculation of tax will be implemented by use of the Strategy pattern. The added items to be displayed on each receipt will be handled by use of the Decorator pattern.

Basic Receipt

The information for the basic receipt should be stored in a BasicReceipt object (see below). The instance variables of a BasicReceipt should contain the date of sale, a PurchasedItems object, 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 TaxComputation that can be assigned the appropriate tax computation object (e.g., MDTaxComputation) 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](http://en.wikipedia.org/wiki/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 2016 will be August 13th and 14th.

The determination of sales tax is not just based on the tax rate and purchase amount, but also on the purchase date (because of the possible existence of tax holidays). That is why the method of tax computation is contained in a TaxComputation object by use of the Strategy pattern. (If there is no applicable sales tax, e.g., in Delaware, then the TaxComputation variable is set to null.) 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 TaxComputation object for the state that the items were purchased.

Adding Additional Receipt Items

There are a number of different “add on” items that may need to be printed with the basic receipt.

During particular times of the year, a receipt header may begin with a special greeting (e.g., “Happy Holidays from Best Buy”), called a secondary header, to be added to the top of a receipt. In addition, rebate forms may be printed at the end of a receipt if a purchased item has a mail-in rebate. Finally, coupons may be printed (also at the end of a receipt) if the total purchases exceeds a certain dollar amount (e.g., if spend over $100, get a 10% off coupon for the next visit).

Objects of interface type AddOn are used to store the added printout for a receipt. The interface has two methods - applies (which is passed a PurchasedItems object containing all of the items for the current receipt), and getLines (which returns the added lines of text to be printed as a single String with embedded newline characters). For AddOn objects of type SecondaryHeader, applies always returns true. This is because if a SecondaryHeader exists, it is always added to the (top) of the receipt. Since rebates only apply to a specific item, method applies returns true if and only if the particular item is found in PurchasedItems. A similar approach is taken for adding coupons to the end of a receipt, except that coupons apply if and only if the customer has spent over a certain amount.

We assume that each Best Buy store downloads the current set of decorator objects each morning.

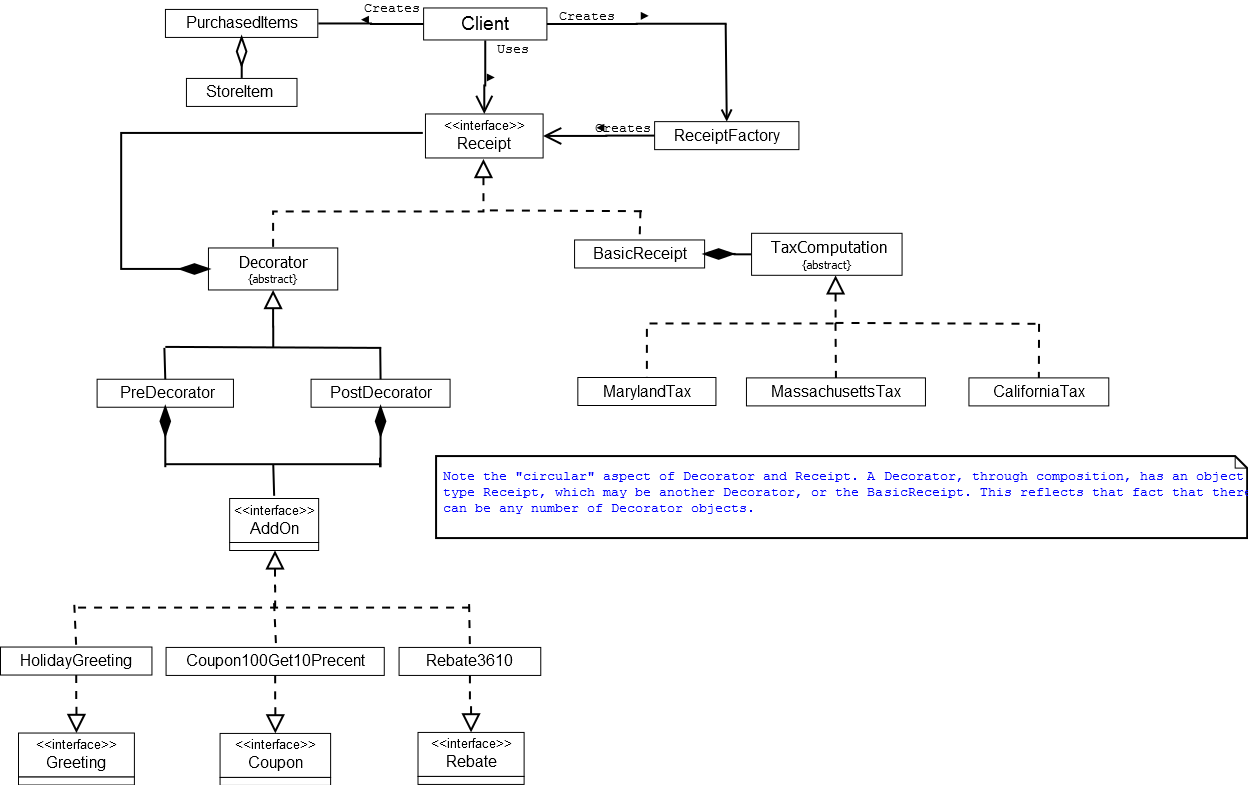
Configuration File

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 street address, store phone number, store number, and state code.

Factory Class

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

A UML class diagram for the design of this program is given below.



**INTERFACES AND CLASSES TO UTILIZE**

Following are the interface and classes to be used in the design of the program.

Interfaces

public interface Receipt { // type of all receipt components (i.e., BasicReceipt and receipt decorators)

public void prtReceipt();

}

public interface AddOn { // the type of added info to a BasicReceipt (e.g., greetings, rebates, coupons)

public boolean applies(PurchasedItems items);

public String getLines();

}

public interface SecondaryHeading { // marker interface, i.e., nothing to implement

}

public interface Rebate { // marker interface, i.e., nothing to implement

}

public interface Coupon { // marker interface, i.e., nothing to implement

}

Abstract Classes

public abstract class Decorator implements Receipt {

private Receipt trailer;

public Decorator(Receipt r) {

trailer = r;

}

protected void callTrailer() {

trailer.prtReceipt();

}

public abstract void prtReceipt();

}

public abstract class TaxComputation {

public abstract double computeTax(PurchasedItems items, ReceiptDate date);

protected abstract boolean taxHoliday();

}

StoreItem Class

public class StoreItem {

private String itemCode; // e.g., 3010

private String itemDescription; // e.g., Dell Laptop

private String itemPrice;

public StoreItem(String code, String descript, String price)

{ ... }

// appropriate getters and setters

}

PurchasedItems Class

public class PurchasedItems {

private ArrayList<StoreItem> items;

public PurchasedItems() {

items = new ArrayList();

}

public void addItem(StoreItem item) { ... }

public double getTotalCost() { ... }

public boolean containsItem(String itemCode) { ... }

}

BasicReceipt Class

public class BasicReceipt implements Receipt {

private String storeInfo; // store number, store address, phone number

private String stateCode; // MD, DE, CA or MA

private PurchasedItems items;

private Date date;

private TaxComputation tc;

public BasicReceipt(PurchasedItems items) {

this.items = items;

}

public void setTaxComputation(TaxComputation tc) { this.tc = tc; }

public void setDate(Date date) { this.date = date; }

public void prtReceipt() {

// to implement

}

}

AddOn Classes

Each of these classes are implemented to provide the information for a particular secondary header, rebate or coupon. The following are *examples only* of what these classes may be.

public class HolidayGreeting implements AddOn, SecondaryHeading {

public boolean applies(PurchasedItems items) {

return true; // SecondaryHeading decorators always applied

}

public String getLines() {

return “\* Happy Holidays from Best Buy \*”;

}

}

public class Rebate1406 implements AddOn, Rebate {

public boolean applies(PurchasedItems items) {

return items.containsItem(“1406”);

{

public String getLines() {

return “Mail-in Rebate for Item #1406\n” + “Name:\n” + “Address:\n\n” +

“Mail to: Best Buy Rebates, P.O. Box 1400, Orlando, FL”;

}

}

public class Coupon100Get10Percent implements AddOn, Coupon { // similar to rebate class }

Decorator Classes

There are two decorator types - one for displaying text at the top of a receipt (PreDecorator), and another for displaying information at the bottom of a receipt (PostDecorator). Each is constructed to contain an AddOn object, which provides the added information to be displayed on the receipt.

public class PreDecorator extends Decorator {

private AddOn a;

public PreDecorator(AddOn a, Receipt r) {

super(r);

this.a = a;

}

public void prtReceipt() {

System.out.println(a.getLines());

callTrailer();

}

}

public class PostDecorator extends Decorator

// similar, except that prtReceipt print the add on information

// *after* the other parts of the receipt are printed

Tax Computation Classes

public class MDTaxComputation extends TaxComputation {

public double computeTax(PurchasedItems items, ReceiptDate date) {

// calls private method taxHoliday as part of this computation

}

private boolean taxHoliday(ReceiptDate date);

// returns true if date of receipt within the state’s tax free holiday,

// else returns false. Supporting method of method computeTax.

}

// tax computation objects for other states are similar (to be completed)

Factory Class

public class ReceiptFactory {

String header; // contains line with “Best Buy”, store\_num, street\_addr, phone

String state\_code;

private TaxComputation[] taxComputationsObjs; // tax computation objects (for each state)

private AddOn[] addOns; // secondary header, rebate and coupon add-ons

public ReceiptFactory() { // constructor

// 1. Populates array of StateComputation objects and array of AddOn objects (as if downloaded  
 from the BestBuy web site).

// 2. Reads config file to assign store\_num, street\_addr, etc.

// 3. Based on the state code (e.g., “MD”, from the config file) stores appropriate   
 StateComputation object to be used on all receipts.

}

public getReceipt(PurchasedItems items) {

// 1. Sets the current date of the BasicReceipt.

// 2. Attaches the StateComputation object to the BasicReceipt (by call to the setComputation   
 method of BasicReceipt).

// 3. Traverses over all AddOn objects, calling the applies method of each. If an AddOn object   
 applies, then determines if the AddOn is of type SecondaryHeader, Rebate, or Coupon.   
 If of type SecondaryHeader, then creates a PreDecorator for othe AddOn. If of type Rebate or   
 Coupon, then creates a PostDecorator.

// 4. Links in the decorator object based on the Decorator design pattern.

// 5. Adds current date to

// 6. Returns decorated BasicReceipt object as type Receipt.

}

}

**CLIENT CODE**

public static void main(String[] args)

{

// 1. Creates a PurchasedItems object.

// 2. Constructs ReceiptFactory object.

// 3. Prompts user for items to purchase, storing each in PurchasedItems.

// 4. Calls the getReceipt method of the factory to obtain constructed receipt.

// 5. Prints receipt by method call to constructed receipt.

// get receipt date

// (prompt user for current date)

// display all available products to user

(to be implemented)

// get all user selections

(to be implemented)

ReceiptFactory factory = new ReceiptFactory();

Receipt receipt = factory.getReceipt(items, date);

receipt.prtReceipt();

}

**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

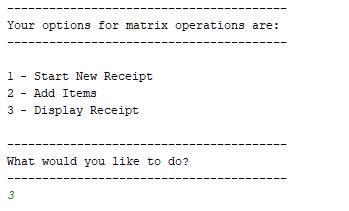
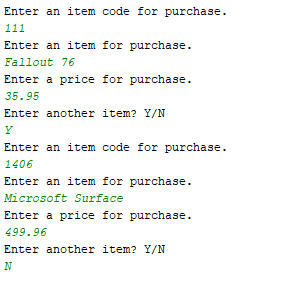
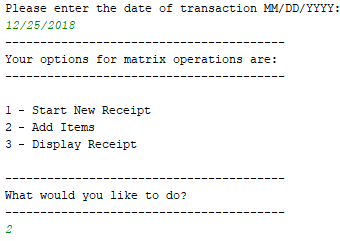
2 – Add Items

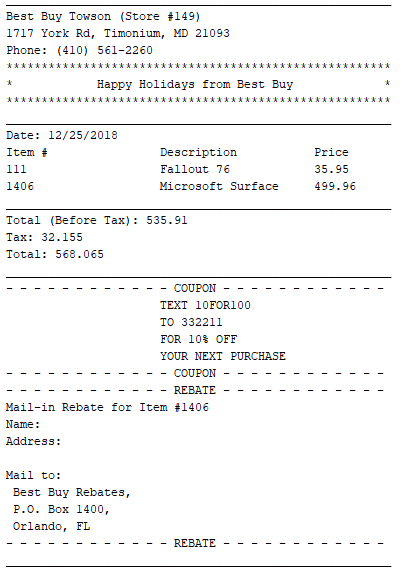
3 – Display Receipt

Vary the AddOn objects stored in the ReceiptFactory to check that the factory builds the correct receipts for various situations (e.g., in which only a BasicReceipt is created; in which a Greeting AddOn exists; and in which various combinations of Coupon and Rebate AddOns exist).

**WHAT TO TURN IN:**

Each of the source files, submitted as one zipped file. Please include sufficient screenshots and introduction to demonstrate how your programs run. 40% deduction if screenshots and introduction are not provided.





**Classes:**

**Interface: AddOn**

Interface for adding any AddOn items to the receipt

**Class: BasicReceipt**

Class for constructing a basic receipt

**Class: CATaxComputation**

Class for calculating tax in California

**Interface: Coupon**

Interface for adding a coupon to the receipt

**Class: Coupon100Get10Percent**

Used for adding a specific coupon for the receipt

**Class: Decorator**

Adds any decoration to the receipt

**Class: HolidayGreeting**

If applicable, will add a holiday greeting to the receipt

**Class: Main**

The main client class, drives the creation and modification of receipts

**Class: MassTaxComputation**

Class for calculating tax in Massachusetts

**Class: MDTaxComputation**

Class for calculating tax in Maryland

**Class: PostDecorator**

Adds decorations to the end of the receipt

**Class: PreDecorator**

Adds decorations to the beginning of the receipt

**Class: PurchasedItems**

Class used to keep track of what was being purchased by the user

**Interface: Rebate**

Interface used for rebates

**Class: Rebate1406**

Class used to generate a rebate response if the area code matches 1406

**Interface: Receipt**

Interface for the creation of a receipt

**Class: ReceiptDate**

Creates a date for the receipt

**Class: ReceiptFactory**

Factory class for the receipt

**Interface: SecondaryHeading**

Interface for any headings underneath the top of the receipt

**Class: StateComputation**

Class to fill all state codes

**Class: StoreItem**

Class used for the creation and storage of any products within the store

**Class: TaxComputation**

Class used for calculating taxes on the bill