ASSIGNMENT 4 – Implementing the Decorator Design Pattern (with Strategy Pattern and Factory Class)

150 pts.

Due Monday, November 14th

PROBLEM

You are to design and implement code based on the Decorator, Strategy and Factory Class design patterns for generating appropriate receipts for customers buying items at a particular Best Buy store location. The general format of a receipt is as follows:

Basic Receipt
Store Header (store street address, state code, zipcode, phone number, 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 that store residing in)
Optional Secondary Heading, e.g.,

- "Happy Holidays from Best Buy"
- "Summer Sales are Hot at Best Buy"

Optional Items printed at the end of the receipt:

- Relevant Rebate Forms
- 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. This information will be stored in a configuration file and read at startup of the system. 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 design pattern**. The possible added items to be displayed at the end of each receipt (i.e., rebates and promotional coupons) will be handled by use of the **Decorator** and **Factory Class design patterns**.

Basic Receipt

The information for the basic receipt should be stored in a **BasicReceipt** object (see below). A BasicReceipt should contain the store header information, date of sale, purchased items, the total sale (without tax,) and the amount due (with added tax). In addition, following the Strategy design pattern, there should be an instance variable of (interface) type **TaxComputationMethod** that can be assigned the appropriate t ax computation object for the state that the store resides in. (For tax purposes, everything purchased from Best Buy is in the category "computer or computer accessory.")

Determining Sales Tax

We will implement the classes so that receipts can be generated for any of the fifty states and the district of Columbia. Note that some states do not have a sales tax (e.g., Delaware). Also, some states have "tax holidays" where certain items on certain dates are not taxed (e.g., school supplies in late summer) -see sales tax holiday.

Because of the existence of sales tax holidays, the determination of sales tax is not just based on the tax rate and purchase amount, but also on the purchase date. That is why the tax computation method is contained in a TaxComputationMethod object. This is a use of the Strategy design pattern. (If there is no applicable sales tax, e.g., in Delaware, then the TaxComputationMethod 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 TaxComputationMethod object for the state that the items were purchased. (We will not be implementing returns.)

Adding Additional Receipt Items

There are a number of different **"add on" items** that may need to be included 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 heading**, to be added to the top of a receipt. In addition, rebate forms may be included at the end of a receipt if a purchased item has a mail-in rebate. Finally, coupons may also be included (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, the applies method 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 specific items, 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 to be used that day.

Factory Class

A **Factory class** will be used to build receipts. The factory class, when constructed, will read the information in the configuration file. From that, it will create a **StoreHeader object** (containing the store's street address, state code, zip code, phone number, and store number) to be attached to each receipt. It will also create and attach the appropriate TaxComputationMethod object for the state that the store resides in. A complete receipt is built and returned by call to **method getReceipt**. The method is passed a **PurchasedItems** object (containing all of the items purchased), and the date of the receipt (either as a simple Date object or a String type). A BasicReceipt object is created with the PurchasedItems and date passed. The previously created StoreHeader and TaxComputationMethod object are each attached (by call to methods setHeader and setTaxComputationMethod of the BasicReceipt class). Within the Factory class is an array of AddOn objects. Finally, each of the AddOn objects are iterated over, and the applies method of each called (each passed the PurchasedItems object). If applies returns true and the type of AddOn object is SecondaryHeader, then a **PreDecorator object** is created for the AddOn object and linked. If, however, the AddOn object is of type Rebate or Coupon, then a **PostDecorator object** is created and linked.

An example receipt is given below. A **UML class diagram** for the design of this program follows.

Happy Holidays from Best Buy!

BEST BUY — Store # 2014
123 Main St., SomeTown, MD 21455 — 410-704-5555

4/16/18 5:51pm

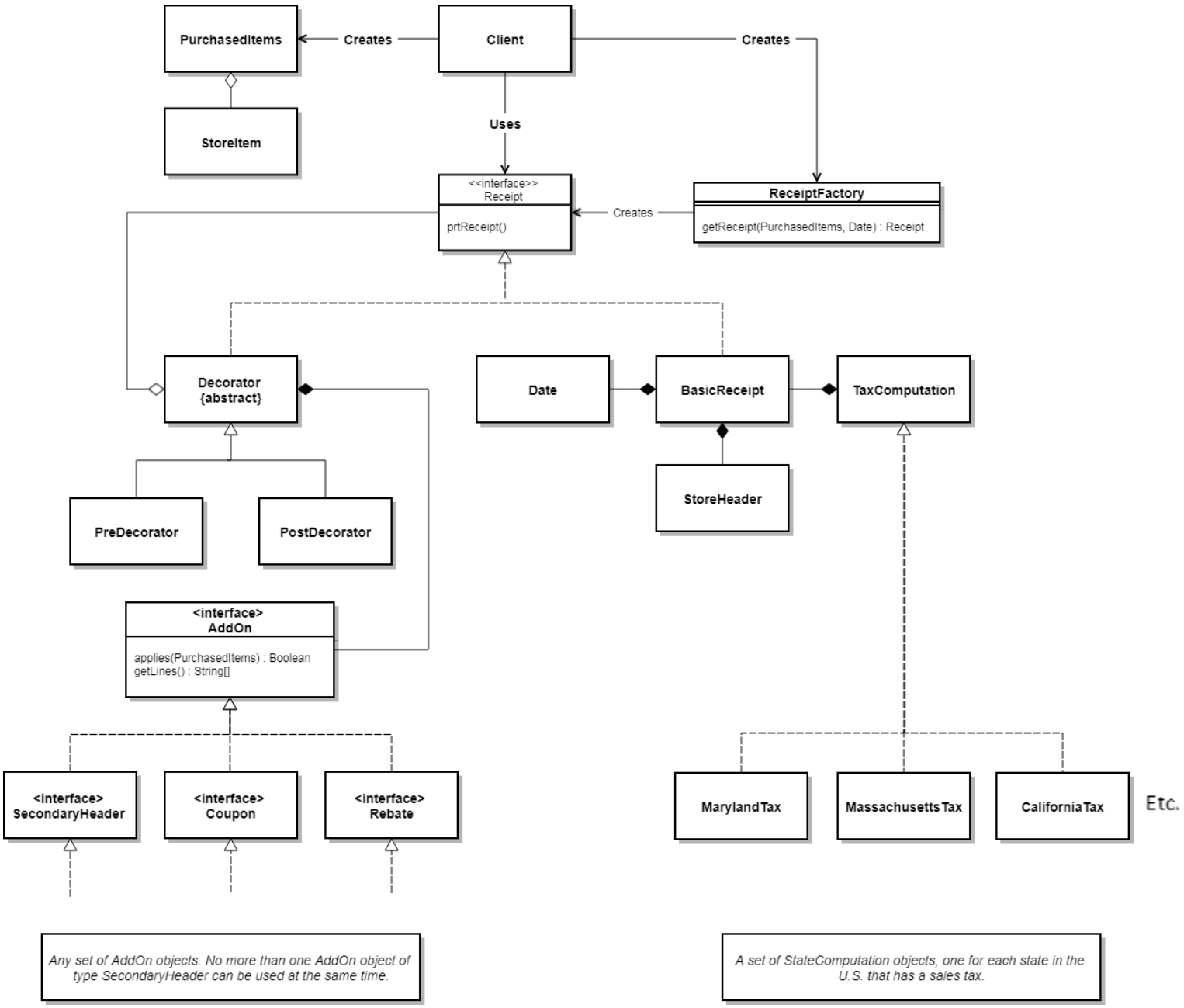
ITEM #	
4028 Samsung S9 (unlocked) Midnight Black 64 GB	719.99
0915 Samsung S9 Case (black)	39.99
2428 Anker External Battery (20,000 mAh)	49.99
Total Sale	$ 809.97
MD Sales Tax (6%)	$ 48.60
TOTAL SALE	$ 138.57

BEST BUY COUPON 10% off next purchase
Good until 12/31/2018

$10 REBATE Item 0915 Samsung S9 Case

Mail ORIGINAL receipt and Proof of Purchase from package to:
BEST BUY / SAMSUNG REBATES
1000 Industry Way
Owings Mills, MD 21117

Please allow 3-4 weeks for processing


PurchasedItems
Creates
Client
Creates
StoreItem
Uses
<<interface>>
Receipt
prtReceipt()
Creates
ReceiptFactory
getReceipt(PurchasedItems, Date) : Receipt
Decorator
{abstract}
Date
BasicReceipt
TaxComputation
StoreHeader
PreDecorator
PostDecorator
<interface>
AddOn
applies(PurchasedItems) : Boolean
getLines() : String[]
<interface>
SecondaryHeader
<interface>
Coupon
<interface>
Rebate
MarylandTax
MassachusettsTax
CaliforniaTax
Etc.
Any set of AddOn objects. No more than one AddOn object of type SecondaryHeader can be used at the same time.
A set of StateComputation objects, one for each state in the U.S. that has a sales tax.

INTERFACES AND CLASSES TO UTILIZE

Following are the interfaces 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 extends AddOn { // marker interface, i.e., nothing to implement
}

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

public interface Coupon extends AddOn { // marker interface, i.e., nothing to implement
}
```

Abstract Classes

```
public abstract class Decorator implements Receipt {
	private Receipt trailer;
	private AddOn;

	public Decorator(Receipt r, AddOn a) {
		trailer = r;
		addon = a;
	}

	protected void callTrailer() {
		trailer.prtReceipt();
	}

	public abstract void prtReceipt();
}


public abstract class TaxComputationMethod {
	public abstract double computeTax(PurchasedItems items, ReceiptDate date);
	private abstract boolean taxHoliday();
}
```

StoreHeader Class

```
public class StoreHeader {
	private String street_addr;
	private String zip_code;
	private String state_code;
	private String phone_num;
	private String store_num; // e.g., #1004

	public StoreHeader(String street_addr, String zip_code, String state_code, String phone_num,
				String store_num)
	{ ... }

	public String getStateCode() { ... }
	public String toString() { .. }
}
```

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 StoreHeader store_header; // street address, state code, phone number, store number
    private TaxComputationMethod tc;
    private Date; // may also be a String type
    private PurchasedItems items;

    public BasicReceipt(PurchasedItems items, Date date) { // Date may also be a String type
        this.items = items;
    }

    public void setStoreHeader(StoreHeader h) {store_header = h;}
    public void setTaxComputationMethod(TaxComputationMethod tc) { this.tc = tc; }

    public void prtReceipt() {
        // to implement
    }
}
```

AddOn Classes

There are **three types of "add on" classes**: SecondaryHeading, Rebate and Coupon. Each of these classes is implemented to provide the information for a particular secondary heading, rebate or coupon. The following are *examples only* of what these classes may be.

```
public class HolidayGreeting implements 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 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 Coupon { // similar to rebate class }
```

Decorator Classes

There are two Decorator types – **PreDecorator** (for displaying text at the top of a receipt) and **PostDecorator** (for displaying text at the bottom of a receipt). Each is **constructed to contain an AddOn object**, which provides the added information to be displayed on the receipt, and a Receipt type object, which is the next object (either another decorator or the basic receipt) that this decorator object should "link" to.

```
public class PreDecorator extends Decorator {

    public PreDecorator(Receipt r, AddOn a) {
        super(r, 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 TaxComputationMethod {

    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
```

Factory Class

```
public class ReceiptFactory {

    StoreHeader store_header; // contains street_addr, zip_code, state_code, phone num, store num
    private TaxComputationMethod[] taxComputationsObjs; // tax computation objs (for each state)
    private AddOn[] addOns; // secondary heading, rebate and coupon add-ons (hardcoded here)

    public ReceiptFactory() { // constructor

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

    // 2. Reads config file to create and save StoreHeader object (store_num, street_addr, etc.) to be
         used on all receipts.

    // 3. Based on the state code (e.g., "MD") creates and stores appropriate StateComputation object
         to be used on all receipts.

    }

    public getReceipt(PurchasedItems items, Date date) {

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

    // 2. Sets StoreHeader object of the BasicReceipt (by call to SetStoreHeader of BasicReceipt)

    // 3. Sets the TaxComputationMethod object of the BasicReceipt (by call to the
         setTaxComputationMethod of BasicReceipt).

    // 4. 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.

    // 5. Links in the decorator object based on the Decorator design pattern.

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

CLIENT CODE

```
public static void main(String[] args)
{
    // 1. Creates a Data object (either from Java API or date entered by user)
    // 2. Creates a PurchasedItems object (selections made by user)
    // 3. Constructs a 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 call to method prtReceipt.

    // get receipt date
    // (prompt user for current date)

    // display all available store items to user
    (to be implemented)

    // get all user selections
    (to be implemented)

    ReceiptFactory factory = new ReceiptFactory();
    Receipt = factory.getReceipt(items, date);
    receipt.prtReceipt();
}
```

PROGRAM TO CREATE

Create a program that will create and display Best Buy receipts. The program should provide a simple text-based interface as follows:

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 SecondaryHeading AddOn exists; and in which various combinations of Coupon and Rebate AddOns exist). Also try various dates for checking that the sales tax is correctly determined for certain states that have tax holidays

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