Assignment 2 - Shopping Cart Calculator

Due: May 23, 2019 10:00

The purpose of this assignment is

- design, implement and documenting classes in Java

Shopping Cart Calculator

After a long night working on Assignment 1 in the lab, you walked out into the chilling weather and suddenly surrounded by blinding white light. Before you realize, you are sitting at an unfamliar workstation with many Starbucks coffee cups lining the desk, talking to bald man with beard who looks like Amazon's CTO.

Turns out in the parallel universe SC1181, Amazon just started launching their online bookstore a week ago and it went viral! With all hands on deck, their engineers have been overworked and suffering carpal tunnel syndromes from calculating order invoices by hand. Knowing this won't scale, their CTO is tasking you to develop an innovative OOP solution to generate order invoices.

Part 1

READ THE FULL ASSIGNMENT (INCLUDING APPENDIX) BEFORE YOU START

(Hint: The following are all classes that you need to implement)

LineItem

A `LineItem` consists of a **product name**, a **quantity** and a **unit price**.

Anyone should be able to

1. construct a LineItem by providing a **product name**, a **quantity** and a **unit price**
2. to read the product name (`getProductName`) and the quantity (`getQuantity`)

No one outside the class should be able to

1. modify the **product name**, **quantity** or **unit price**

Implement this in `LineItem.java` . Make sure you implement the following methods

```
// LineItem.java
public LineItem(String productName, long quantity, double unitPrice)
{}
public String getProductName(){}
public long getQuantity(){}
public double getUnitPrice(){}
```

Order

An `Order` consists of an `ArrayList` of `LineItem` s.

See **Appendix** section on some notes about `ArrayList` .

Anyone should be able to

1. construct an empty Order
   - when an order is constructed, initialize `lineItems` with an empty `ArrayList` (Hint: See Appendix)

2. add a LineItem (`addLineItem`) by providing a `productName` , a `quantity` and a `unitPrice`
   i. if `lineItems` contains a `LineItem` with the same name
      i. remove that `LineItem` from the list
      ii. create a new `LineItem` with the provided `productName` , `quantity` and `unitPrice` and add to `lineItems`

   ii. otherwise, create a new `LineItem` with the provided `productName` , `quantity` and `unitPrice` and add to `lineItems`

3. add a LineItem(`addLineItem`) by providing a `productName` and a `unitPrice`
   - it should follow the same logic as `addLineItem` above
   - and the **quantity** would default to 1

4. remove a LineItem (`removeLineItem`) by providing a `productName`
   - if `lineItems` contains a `LineItem` with the same name, remove that `LineItem` from the list

5. get all the `LineItem` s within the Order (`getAllLineItems`)

Implement this in `Order.java` . Make sure you implement the following methods

```
// Order.java
public Order(){}
public void addLineItem(String productName, long quantity, double un
itPrice){}
public void addLineItem(String productName, double unitPrice){}
public void removeLineItem(String productName){}
public LineItem[] getAllLineItems(){}
```

OrderPrinter

Create a class called `OrderPrinter`

No one outside of the class can

1. `printHeader` which prints `"Product Name, Quantity, Unit Price"` on its own line

Anyone can

1. create a `OrderPrinter`
2. use `print(Order order)` to print an order
   - First it'll print the header using `printHeader`
   - For each `LineItem` , you should print `"{item.productName}, {item.quantity}, {item.unitPrice}"` on each line
     - For example, `Lord of the Rings, 30, 20.5` .

```
// OrderPrinter.java
public void print(Order order){}
```

Test1

Create a class called `Test1`

Then create the `public static void main(String[] args) {}` method

Inside, write the code to do the following

1) Create an order called `order1`

2) Create an order called `order2`

3) Add the following into `order1` , please follow the order

Operation	Product Name	Quantity	Unit Price
add	Lord of the Rings	30	20.5
add	Bible	50	5
add	Lord of the Rings	unspecified	22.5

Operation	Product Name	Quantity	Unit Price
add	Harry Potter	300	10
remove	Harry Potter		

4) Create a `OrderPrinter`

5) Use the `orderPrinter` to print out `order1`

6) Use the `orderPrinter` to print out `order2`

Make sure you implement the following

```
// Test1.java
public static void main(String[] args) {}
```

Part 2

TaxCalculator

A `TaxCalculator` calculates tax of a particular `Order`

Anyone can

1. construct a `TaxCalculator` by providing a `taxRate` in `double`
2. construct a `TaxCalculator`, with a default `taxRate` of 0.2 (i.e. 20%)
3. use the `taxCalculator` to calculate the tax (`calculateTax`) by providing an `Order`
   - if the `Order` has more than 6 `LineItem` s, then the order is tax free (0)
   - otherwise, the tax is charged at 20%
     - E.g., if an `Order` consists of 1 `LineItem` with quantity 5 and unit price 10. The tax should be `5 * 10 * 0.2 = 10` .

Make sure you implement the following

```
public TaxCalculator(double taxRate)
public TaxCalculator()
public double calculateTax(Order order)
```

Invoice

An `Invoice` consists of 3 pieces of information

1. Total number of line items (`totalNumberOfLineItems`)
2. Total quantity (`totalQuantity`)
3. Total price (`totalPrice`) **(not including tax)**
4. Total tax (`totalTax`)

Create a constructor and the `getters` . See below for their signature

```
public Invoice(long totalNumberOfLineItems, long totalQuantity, doub
le totalPrice, double totalTax)
public long getTotalOfLineItems()
public long getTotalQuantity()
public double getTotalPrice()
public double getTotalTax()
```

InvoiceCalculator

An `InvoiceCalculator`

Anyone can

1. construct it by providing a `TaxCalculator`
2. `generateInvoice` by providing an `Order`
    - you can get `totalProducts` , `totalQuantity` , `totalPrice` from the order, and get `totalTax` using `TaxCalculator`

Make sure you implement the following

```
public InvoiceCalculator(TaxCalculator taxCalculator)
public Invoice generateInvoice(Order order)
```

Test2

Then create the `public static void main(String[] args) {}` method

Inside, write the code to do the following

1) Create a `TaxCalculator` with the default tax rate

2) Create an `InvoiceCalculator`

3) Create an `OrderPrinter

4) Create an order called `order1`

5) Add the following into `order1` , please follow the order

Operation	Product Name	Quantity	Unit Price
add	Lord of the Rings	30	20.5
add	Bible	5	5
remove	Harry Potter		
add	Lord of the Rings	unspecified	22.5
add	Harry Potter	3	10

6) Use the `invoiceCalculator` to calculate the `Invoice`

7) Use the `orderPrinter` to print out `order1`

8) Then print the invoice as follow,

```
Total number of Products: 3
Total Quantity: 9
Total Price: 77.5
Total Tax: 15.5
```

Please round all price and tax to 2 decimal places before printing them.

For example, if the price is `7.567` then print `7.57` . If the price is `7.5` , then print `7.5`

Bonus

InvoicePrinter

Implement `InvoicePrinter` to do **Step 8** of `Test2`

```
public void print(Invoice invoice)
```

CheckoutService

Create a class called `CheckoutService`

Anyone can

1. construct `CheckoutService` by providing an `InvoiceCalculator` , an `OrderPrinter` and an `InvoicePrinter`
2. `checkout(Order order)` to checkout an order, which
   i. generate an invoice using `InvoiceCalculator`
   ii. print the order using `OrderPrinter`
   iii. print the invoice using `InvoicePrinter`

After you are done, replace the logic in `Test2` with `CheckoutService`

```
public CheckoutService(InvoiceCalculator invoiceCalculator, OrderPri
nter orderPrinter, InvoicePrinter invoicePrinter)
public void checkout(Order order)
```

Code Documentation

Please document all `public` methods other than **getters**.

- 1 to 2 lines descirption of what the method does
- for each argument, use `@param argumentName` to describe what it is
- for any method that's not `void` , use `@return` to describe what it returns

Here's an example

```
/**
 * Calculates the tax of a home
 *
 * @param home A home, address must be initialized
 * @return     returns the property tax based on 2019 BC guidelines
 *             rounded to 2 decimal place
 */
 public double calculateHomeTax(Home home) {
   //...
 }
```

Implementation Notes and Hints

IMPORTANT For this assigment, `long` is sufficient for quantity and use `double` for any price/tax related values.

IMPORTANT You should not implement any additional `public` methods besides the ones listed above.

Experiment with `ArrayList` before you start coding

See Appendix below with sample programs. Experiment with them to see how `ArrayList` works. It's crucial you understand

Take small steps!

- implement a method, then use `public static void main(String[] args)` to verify that it works before moving on to next step
- implement the classes in order, make sure a class works before attempting the next
- feel free to implement helper methods in the classes, but make sure you implement the required methods as specified

Document your code, it helps the marker understand your code.

DO NOT copy any Java code from the Internet or from anyone else. If you do, you will get zero and you be reported to the Dean of Student Services for Academic Dishonesty (see the course outline).

Example Compilation

```
$> javac -d out Test1.java
```

```
$> javac -d out Test2.java
```

Example Run

```
$> java -cp out Test1
<Some output>
```

```
$> java -cp out Test2
<Some output>
```

Submission

Submit to BrightSpace as a single ZIP file (*<your student ID>.zip*). Inside contains

1. A directory named `assignment02` , inside contains
   i. All the source files specified above
   ii. **DO NOT** submit `.class` files, **DO NOT** submit the `out` directory

Marking Scheme [70]

- [30] Programming Style
    - [5] Broken down into smaller methods
    - [10] Proper use of encapsulation
    - [10] Documentation
    - [10] Coding Style
    - Descriptive variable names
    - Consistent variable and method naming

- [30] Code Correctness
- [10] Bonus

Appendix

ArrayList

We will go into `ArrayList` over the next few weeks with more details.

In the mean time, I've provided an example of common operations with `ArrayList` for this assignment on D2L.

To use `ArrayList` in your `Order` class, remember to add

```
import java.util.ArrayList;
```

as the first line of `Order.java` .

Then you'll need to declare an instance variable as such, and instantiate `lineItems` in the constructor.

```
public class Order {
  private ArrayList<LineItem> lineItems;

  public Order() {
    lineItems = new ArrayList<LineItem>();
  }
}
```