COMP2396 Object-oriented programming and Java

Assignment 3: Vending Machine

Due date: 18 Mar, 2020 23:59

Notes about submitting this assignment

- Please submit your solution to Moodle.
- Please double check your submission. Please check the assignment page again after submission to ensure all files are submitted.
- You are also required to write JavaDoc for all non-private classes and non-private class members.
- The main programs shown on this sheet are only for your reference to develop the classes. We will use different test cases to test your program outputs against standard outputs.
- Please contact us as soon as possible if you encounter any problem regarding this submission

Description:

Consider a vending machine, which vends soft drinks, that accept coins only. The vending machine allows customers to insert coins, purchase a product and reject inserted coins. In this assignment, you need to build a system to simulate the operations of vending soft drinks in a vending machine. Please read this assignment sheet carefully before starting your work.

Typical vending machine consist of these components:

- A Coin Slot allows customers to insert coins into the machine. It also serves as temporality storage for inserted coins and accumulates the amount of face value.
- A button to reject all inserted coins.
- A Coin Changer stores coins for giving change. Change is the money that is returned to the customer who has paid for the product that costs less than the amount that the customer has given.
- A Soft Drink Slot holds the same products of soft drinks in a column. When a transaction is made, the slot will drop a can of drink into the dispenser.

Standard workflow for vending a soft drink in a vending machine is as follow:

1. The customer inserts coins into the Coin Slot.
2. The customer selects a product.
3. If the customer has inserted enough amount of money for the product, the vending machine firstly drops the product and return change (if necessary). Then, all coins in the Coin Slot are moved to the Coin Changer.

Task: Implement this system. Below shows a simple run-down.

Sample main

```
public class Main {
  public static void main(String[] args) {
    VendingMachine v = new VendingMachine();
    v.addCoinToCoinChanger(Integer.valueOf(2));
    v.addCoinToCoinChanger(Integer.valueOf(2));
    v.addCoinToCoinChanger(Integer.valueOf(1));
    v.addSoftDrinkSlot(new SoftDrinkSlot("Cocacola", 4, 1)); // Price: $4, Quantity: 1
    v.addSoftDrinkSlot(new SoftDrinkSlot("Pepsi", 5, 3)); // Price: $5, Quantity: 3

    System.out.println(((Command) new CmdInsertCoin()).execute(v, "10"));
    System.out.println(((Command) new CmdInsertCoin()).execute(v, "2"));
    System.out.println(((Command) new CmdRejectCoins()).execute(v, ""));
    System.out.println();
    System.out.println(((Command) new CmdRejectCoins()).execute(v, ""));
    System.out.println();
    System.out.println(((Command) new CmdInsertCoin()).execute(v, "5"));
    System.out.println(((Command) new CmdPurchase()).execute(v, "Pepsi"));
    System.out.println();
    System.out.println(((Command) new CmdInsertCoin()).execute(v, "10"));
    System.out.println(((Command) new CmdInsertCoin()).execute(v, "5"));
    System.out.println(((Command) new CmdPurchase()).execute(v, "Pepsi"));
    System.out.println();
    System.out.println(((Command) new CmdInsertCoin()).execute(v, "5"));
    System.out.println(((Command) new CmdInsertCoin()).execute(v, "2"));
    System.out.println(((Command) new CmdPurchase()).execute(v, "Pepsi"));
    System.out.println();
    System.out.println(((Command) new CmdRejectCoins()).execute(v, ""));
    System.out.println();
    System.out.println(((Command) new CmdInsertCoin()).execute(v, "2"));
    System.out.println(((Command) new CmdPurchase()).execute(v, "Pepsi"));
    System.out.println();
    System.out.println(((Command) new CmdInsertCoin()).execute(v, "2"));
    System.out.println(((Command) new CmdPurchase()).execute(v, "Cocacola"));
    System.out.println();
    System.out.println(((Command) new CmdInsertCoin()).execute(v, "2"));
    System.out.println(((Command) new CmdInsertCoin()).execute(v, "2"));
    System.out.println(((Command) new CmdPurchase()).execute(v, "Cocacola"));
    System.out.println(((Command) new CmdRejectCoins()).execute(v, ""));
  }
}
```

Sample output

```
Inserted a $10 coin. $10 in Total.
Inserted a $2 coin. $12 in Total.
Rejected $2, $10. $12 in Total.

Rejected no coin!

Inserted a $5 coin. $5 in Total.
Purchasing Pepsi... Success! Paid $5. No change.

Inserted a $10 coin. $10 in Total.
Inserted a $5 coin. $15 in Total.
Purchasing Pepsi... Success! Paid $15. Change: $1, $2, $2, $5.

Inserted a $5 coin. $5 in Total.
Inserted a $2 coin. $7 in Total.
Purchasing Pepsi... Insufficient change!

Rejected $2, $5. $7 in Total.

Inserted a $2 coin. $2 in Total.
Purchasing Pepsi... Insufficient amount! Inserted $2 but needs $5.

Inserted a $2 coin. $4 in Total.
Purchasing Cocacola... Success! Paid $4. No change.

Inserted a $2 coin. $2 in Total.
Inserted a $2 coin. $4 in Total.
Purchasing Cocacola... Out of stock!
Rejected $2, $2. $4 in Total.
```

Only one $5 and $10 coins in Coin Changer, cannot make $2 change.

A partial completed VendingMachine class is provided as follows

```java
import java.util.ArrayList;

public class VendingMachine {
	private ArrayList<Integer> coinChanger;
	private ArrayList<Integer> coinSlot;
	private ArrayList<SoftDrinkSlot> softDrinkSlots;

	public VendingMachine() {
		coinChanger = new ArrayList<Integer>();
		coinSlot = new ArrayList<Integer>();
		softDrinkSlots = new ArrayList<SoftDrinkSlot>();
	}

	public void addCoinToCoinChanger(Integer c) {
		coinChanger.add(c);
	}

	public void addSoftDrinkSlot(SoftDrinkSlot s) {
		softDrinkSlots.add(s);
	}
	/* You may add other non-static properties and methods */
}
```

You must not modify the parameter lists of the constructor, addCoinToCoinChanger() and addSoftDrinkSlot() as we will use these methods to initiate the vending machine in test cases evaluation.

The SoftDrinkSlot class is provided as follow

```java
public class SoftDrinkSlot {

	private String name;
	private int price;
	private int quantity;

	public SoftDrinkSlot(String name, int price, int quantity) {
		this.name = name;
		this.price = price;
		this.quantity = quantity;
	}
	/* You may add other non-static properties and methods */
}
```

You must not modify the parameter lists of the constructor as we will use it in test cases evaluation.

The system should be operated via commands (except initiating the vending machine), which is, every time we want to operate the vending machine, an instance of Command object should be created. The Command object will be the interface for the main program to perform actions on the Vending Machine. Different types of commands are handled by different specific classes that inherit the Command class.

The Command class is defined as follows

```java
public class Command {
	public String execute(VendingMachine v, String cmdPart){
		String result = null;
		return result;
	}
}
```

You need to implement 3 commands for the system, namely CmdInsertCoin, CmdRejectCoins and CmdPurchase. These commands should inherit the Command class. Each Command class perform specific actions to the vending machine in the execute() method. The method takes the VendingMachine object and a command content in String as the input parameters. After performing actions, the method should return the result of the command in String.

1. CmdInsertCoin – Insert a coin
2. CmdRejectCoins – Reject all coins from Coin Slot
3. CmdPurchase – Vend a can of soft drink to the customer. When giving a change to the customer, the machine should find the **minimum number of coins** in the Coin Changer to make the change.

For example, CmdInsertCoin inherit Command class and perform actions in the execute() method:

```java
public class CmdInsertCoin extends Command {

	@Override
	public String execute(VendingMachine v, String cmdPart) {
		Integer coin = Integer.valueOf(cmdPart);
		// Add the coin to Coin Slot
		return /*...*/;
	}
}
```

You also need to add handling for the following special cases, but you can assume no more than one special case would happen simultaneously in our test cases.

- Insufficient amount of money to buy the drink.
- The drink is out of stock
- Not enough coins in Coin Changer for giving a particular amount of change to the customer.

Notes:

1. The listing of coins should be sorted by the face value ascendingly.
2. All coins are in dollar unit ($1, $2, $5, $10), no need to deal with cents.
3. When preparing coins for a change, the vending machine looks for coins in the **Coin Changer** only.
4. The system should find the **minimum number of coins** for preparing the change. For example, if one $5 coin is available, then the $5 coin should be selected instead of selecting $1, $2 and $2 coins.
5. You are free to add other classes that help you to implement the system. You can upload up to 50 files to Moodle.
6. You must **NOT** declare any **static variable** in your program as a static variable will disrupt our evaluation system.
7. If you failed to pass a test case, Moodle only shows you the last line of expected output and program output.

Grading:

- **80% marks** are given to the **correctness** of your program by the Moodle **evaluation** system.

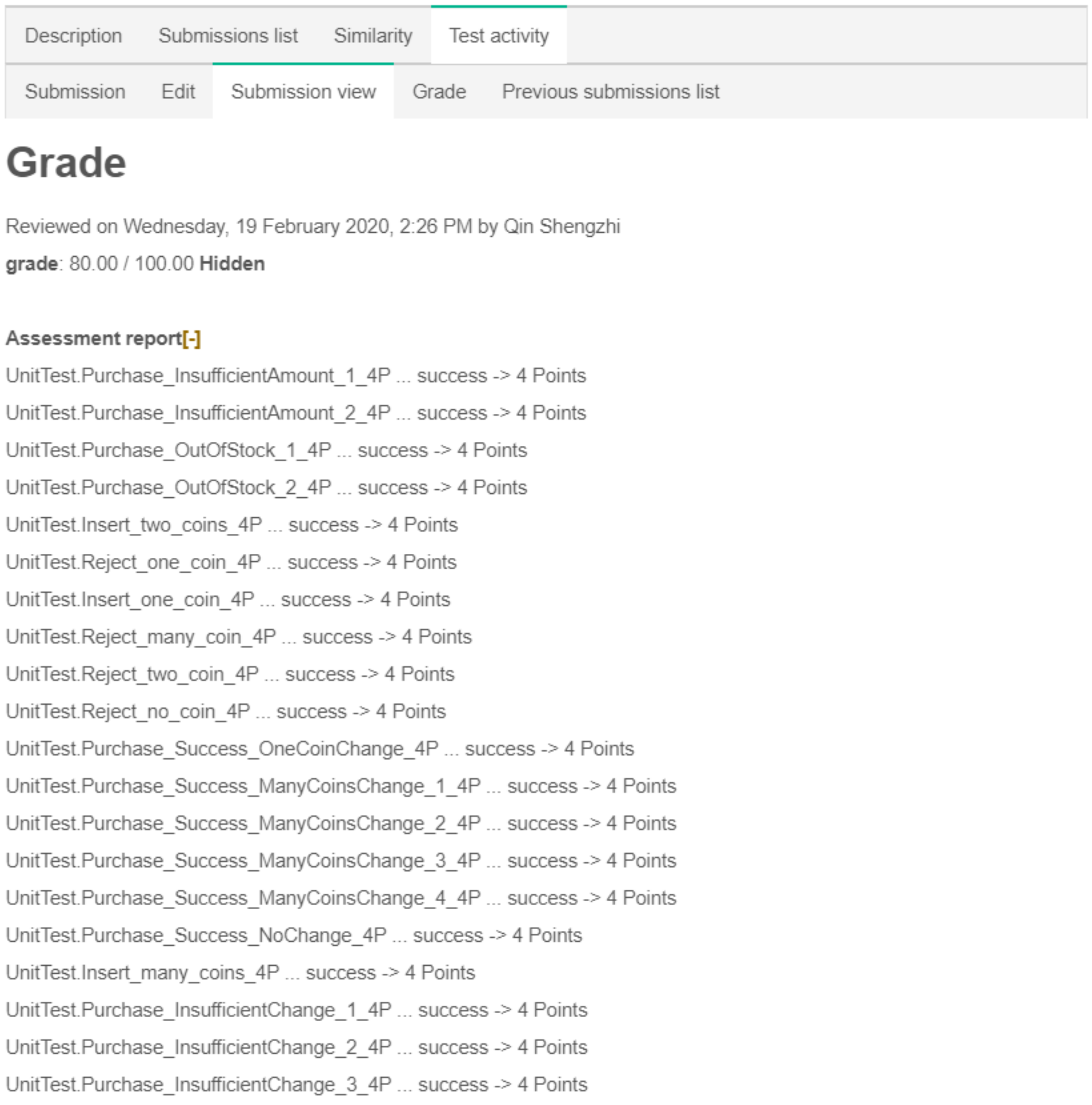
- ➢ You will see your marks immediately right after the evaluation.
- ➢ You can re-submit the assignment before the deadline. We only consider the latest submission for final grading.
- ➢ Output format is critical to the evaluation. Make sure there are space characters and newlines in proper locations in the program output.
- ➢ You are not recommended to declare any static variable in your program as a static variable will keep data across test cases and make you difficult to debug.
- ➢ Normally, we will not release any test case. However, if you failed to pass a certain test case, the expected output shown on the screen will give you some hints to debug your program.

- **20% marks** are given to your **JavaDoc** and will be given **manually** after the due date. A complete JavaDoc includes documentation of every classes, member fields and methods that are not private.

Submission:

Please submit all .java files (except the Main.java) to Moodle. **Late submission is not allowed.**

If you got all the marks from the evaluation system (80% of the assignment), the grading report would look like this:



Grade

Reviewed on Wednesday, 19 February 2020, 2:26 PM by Qin Shengzhi

grade: 80.00 / 100.00 **Hidden**

Assessment report[-]

```
UnitTest.Purchase_InsufficientAmount_1_4P ... success -> 4 Points
UnitTest.Purchase_InsufficientAmount_2_4P ... success -> 4 Points
UnitTest.Purchase_OutOfStock_1_4P ... success -> 4 Points
UnitTest.Purchase_OutOfStock_2_4P ... success -> 4 Points
UnitTest.Insert_two_coins_4P ... success -> 4 Points
UnitTest.Reject_one_coin_4P ... success -> 4 Points
UnitTest.Insert_one_coin_4P ... success -> 4 Points
UnitTest.Reject_many_coin_4P ... success -> 4 Points
UnitTest.Reject_two_coin_4P ... success -> 4 Points
UnitTest.Reject_no_coin_4P ... success -> 4 Points
UnitTest.Purchase_Success_OneCoinChange_4P ... success -> 4 Points
UnitTest.Purchase_Success_ManyCoinsChange_1_4P ... success -> 4 Points
UnitTest.Purchase_Success_ManyCoinsChange_2_4P ... success -> 4 Points
UnitTest.Purchase_Success_ManyCoinsChange_3_4P ... success -> 4 Points
UnitTest.Purchase_Success_ManyCoinsChange_4_4P ... success -> 4 Points
UnitTest.Purchase_Success_NoChange_4P ... success -> 4 Points
UnitTest.Insert_many_coins_4P ... success -> 4 Points
UnitTest.Purchase_InsufficientChange_1_4P ... success -> 4 Points
UnitTest.Purchase_InsufficientChange_2_4P ... success -> 4 Points
UnitTest.Purchase_InsufficientChange_3_4P ... success -> 4 Points
```

-END-