COMP2396 Object-oriented programming and Java

Assignment 2: Vending Machine

Due date: 25th March 2022 (Friday), 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.
- Program comment is not necessary for this assignment.
- Please contact us as soon as possible if you encounter any problem regarding this submission

Description:

Consider a vending machine, which sells soft drinks, that accept coins only. The vending machine allows customers to insert coins, reject inserted coins and purchase a product. In this assignment, you need to build a system to simulate the operations of 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. You should assume there is an **infinite supply of coins** in the coin changer.
- A component to hold the same products of soft drinks in a column. When a transaction is made, it drops a can of drink into the dispenser.

Standard workflow for selling 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 number of credits for the product, the vending machine firstly drops the product and return change (if necessary). Then, all coins in the Coin Slot are collected.

Task: Implement this system. Below shows the main program and a simple run-down.

Provided Main,java

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Main {

  public static void main(String[] args) {
    BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
    String inputLine = "";

    VendingMachine v = new VendingMachine();

    // Adds products to the vending machine
    v.addProduct(new Product("Cocacola", 4, 1)); // Price: $4, Quantity: 1
    v.addProduct(new Product("Pepsi", 5, 3)); // Price: $5, Quantity: 3

    System.out.println("Welcome to COMP2396 Assignment 2 - Vending Machine");

    // Reads user inputs continuously
    while (true) {
      try {
        inputLine = input.readLine();
      } catch (IOException e) {
        System.out.print("Input Error!");
        e.printStackTrace();
      }

      // Split the input line
      String[] cmdParts = inputLine.split(" ");

      if (cmdParts[0].equalsIgnoreCase("Exit")) {
        break;
      } else if (cmdParts[0].equalsIgnoreCase("Insert")) {
        System.out.println(((Command) new CmdInsertCoin()).execute(v, cmdParts[1]));
      } else if (cmdParts[0].equalsIgnoreCase("Reject")) {
        System.out.println(((Command) new CmdRejectCoins()).execute(v, ""));
      } else if (cmdParts[0].equalsIgnoreCase("Buy")) {
        System.out.println(((Command) new CmdPurchase()).execute(v, cmdParts[1]));
      } else {
        System.out.println("Unknown input.");
      }
      inputLine = "";
    }

    System.out.println("Bye");
  }
}
```

We hard code the inventory in the main program

Sample output (User inputs are in Bold style)

```
Welcome to COMP2396 Assignment 2 - Vending Machine
Insert 10
Inserted a $10 coin. $10 in total.
Insert 1
Inserted a $1 coin. $11 in total.
Insert 2
Inserted a $2 coin. $13 in total.
Reject
Rejected $1, $2, $10. $13 in total.
Reject
Rejected no coin!
Insert 5
Inserted a $5 coin. $5 in total.
Buy Pepsi
Dropped Pepsi. Paid $5. No change.
Insert 10
Inserted a $10 coin. $10 in total.
Insert 2
Inserted a $2 coin. $12 in total.
Insert 2
Inserted a $2 coin. $14 in total.
Insert 2
Inserted a $2 coin. $16 in total.
Buy Pepsi
Dropped Pepsi. Paid $16. Your change: $1, $10.
Insert 1
Inserted a $1 coin. $1 in total.
Insert 1
Inserted a $1 coin. $2 in total.
```

Refer to note 1 for the format of listing of coins

Refer to note 1 for the format of listing of coins

	Buy Pepsi Not enough credit to buy Pepsi! Inserted $2 but needs $5. **Insert 2** Inserted a $2 coin. $4 in total. **Buy Cocacola** Dropped Cocacola. Paid $4. No change. **Insert 2** Inserted a $2 coin. $2 in total. **Insert 2** Inserted a $2 coin. $4 in total. **Buy Cocacola** Cocacola is out of stock! **Reject** Rejected $2, $2. $4 in total. **Exit** Bye

A partially completed VendingMachine class is provided as follows.

```
import java.util.ArrayList;

public class VendingMachine {
	// ArrayList of Integers represents inserted coins in Coin Slot
	private ArrayList<Integer> insertedCoins;

	// ArrayList of Product represents inventories of products
	private ArrayList<Product> products;

	public VendingMachine() {
		insertedCoins = new ArrayList<Integer>();
		products = new ArrayList<Product>();
	}

	public void addProduct(Product p) {
		products.add(p);
	}

	public void insertCoin(Integer c) {
		insertedCoins.add(c);
	}
	/* You may add other properties and methods */
}
```

The Product class is provided as follow.

```
public class Product {

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

	public Product(String name, int price, int quantity) {
		this.name = name;
		this.price = price;
		this.quantity = quantity;
	}

	/* You may add other properties and methods */
}
```

You should interact with the vending machine instance via command classes. In other words, for every user command, you should create an object of Command and perform actions on the Vending Machine in the Command object. Different types of commands are handled by different specific classes that inherit the Command Interface.

The Command Interface is defined as follows

```
public interface Command {
	public String execute(VendingMachine v, String cmdPart);
}
```

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

1. CmdInsertCoin – Accept a coin. This command should report the face value of the inserted coin and the total amount of credit currently stored in the coin slot.
2. CmdRejectCoins – Reject all coins from Coin Slot. This command should report the total amount of credit rejected.
3. CmdPurchase – Sell a can of soft drink to the customer. A transaction can be made only if sufficient credit is inserted into the Coin Slot. This command should report the statue of the transaction and how much change is returned to the customer. Refer to Note 3 below for more details.

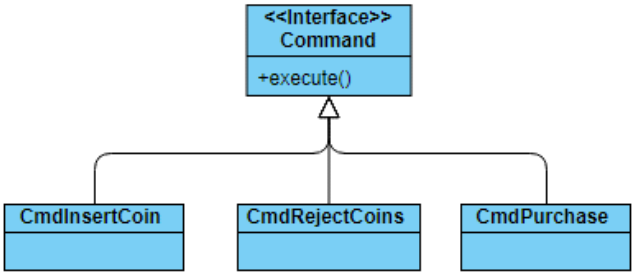


Figure 1 illustration

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

```
public class CmdInsertCoin implements Command {

	@Override
	public String execute(VendingMachine v, String cmdPart) {
		Integer c = Integer.valueOf(cmdPart);
		// Add the coin to Coin Slot
		v.insertCoin(c);
		// Do something
		// return something. Format: "Inserted a $x coin. $y in total."
	}
}
```

You also need to handle the following special cases, but you can assume no more than one special case would happen simultaneously in our test cases. Please refer to sample output for reference.

- Inserted credit is not enough to buy the drink.
- The drink is out of stock.

Notes:

1. The listing of coins should be sorted by the face value ascendingly.
2. All coins are in dollar unit (i.e., $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, in which there is an **infinite supply of coins**. The system should look for the largest coin, which is not larger than needed. Then, repeat the same step to look for the largest possible coin to make up the denomination. For example, if we want a $7 change, we should select [$5, $2] in order instead of selecting [$2, $2, $2, $1] or [$5, $1, $1]. Please also refer to sample output for reference.
4. All program output should be done via the return of the execute() in command objects and the main program. In other words, you cannot use System.out.println() in your own code.
5. You are welcome to add other classes that help you to implement the system. You can upload up to 20 files to Moodle.

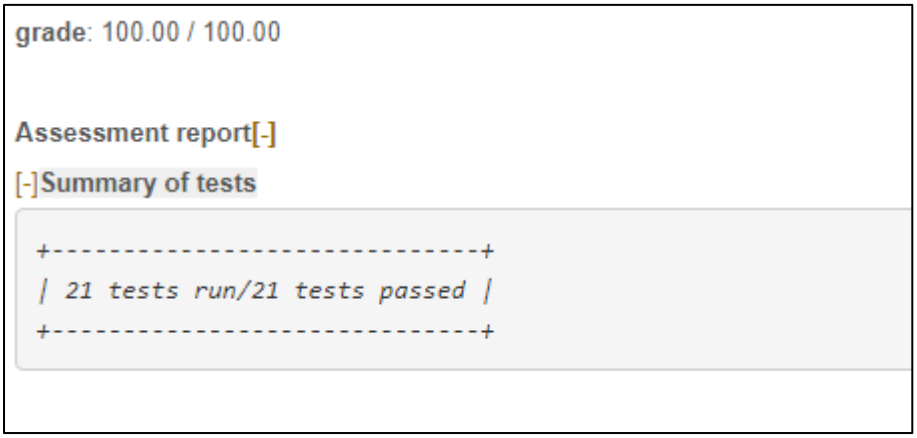
List of required files for submission:

File	Provided in this assignment sheet?	Can I edit it?
Main.java	Yes	No
Command.java	Yes	No
VendingMachine.java	Yes, partially completed	Yes
Product.java	Yes, partially completed	Yes
CmdInsertCoin.java	Yes, partially completed	Yes
CmdPurchase.java	No, you need to create it	Yes, of course
CmdRejectCoins.java	No, you need to create it	Yes, of course

Submission and marking:

Please submit all .java files (including the Main.java) to Moodle and evaluate.

You will see the following result if you got 100% marks.



Distribution of marks:

1. Implementation of basic features: 2 test case (~9.5%)
2. Implementation of CmdInsertCoin: 5 test cases (~23.8%)
3. Implementation of CmdRejectCoins: 7 test cases (~33.3%)
4. Implementation of CmdPurchase: 7 test cases (~33.3%)

Main.java and Commnad.java ensure that you interact with the vending machine via command objects. You must not change the program code in the two files. We will check your submission manually after the deadline. If we found that you have changed it, we will impose a 30% marks penalty.

-END-