COMP2396 Object-oriented programming and Java

Assignment 3: Vending Machine

Due date: 4th November, 2025 23:59

Notes about submitting this assignment

- Please submit your solution to Moodle and evaluate it.
- Javadoc comments are not needed for this assignment.
- Please contact us as soon as possible if you encounter any problems regarding this submission.

Description:

Consider a vending machine that sells soft drinks that accept coins only. The vending machine allows add and check product information, 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.

The vending machines consist of these components:

- A Coin Slot allows customers to insert coins into the machine. It also serves as temporary storage for inserted coins and accumulates the amount of face value.
- Add product quantity and check product information.
- A button to reject all inserted coins.
- A Coin Changer stores coins to give 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 holds the same products of soft drinks in a column. When a transaction is made, it drops a can of drink into the dispenser.

The procedure for selling a soft drink in a vending machine is as follows:

1. The customer inserts coins into the Coin Slot.
2. The customer selects a product.
3. If the customer has inserted enough credits for the product, the vending machine first drops the product and returns the change (if necessary). Then, all coins in the Coin Slot are collected.

Additionally, the customer could reject all coins and check the product information. More requirements for this assignment are below.

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

Provided Main.java

```
import java.io.*;

public class Main {

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

		VendingMachine v = new VendingMachine();

/*implement some code in initializeProducts()in VendingMachine class so that all the products
are initialized with quantity 0 with their corresponding price when v.initializeProducts() is
called in Main. Please refer to Notes 6 in page 5 for the products and their corresponding
price.*/

		v.initializeProducts();

		System.out.println("Welcome to COMP2396 Assignment 3 - Vending Machine");

		// Reads user inputs continuously
		while (true) {
			inputLine = input.readLine();

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

			Command cmdObj = null;

			if (cmdParts[0].equalsIgnoreCase("Exit")) {
				break;
			} else if (cmdParts[0].equalsIgnoreCase("Check")) {
				cmdObj = new CmdCheckProductInfo();
			 } else if (cmdParts[0].equalsIgnoreCase("Insert")) {
				cmdObj = new CmdInsertCoin();
			} else if (cmdParts[0].equalsIgnoreCase("Reject")) {
				cmdObj = new CmdRejectCoins();
			} else if (cmdParts[0].equalsIgnoreCase("Buy")) {
				cmdObj = new CmdPurchase();
			} else if (cmdParts[0].equalsIgnoreCase("Add")) {
				cmdObj = new CmdAddProduct();
			} else {
				System.out.println("Unknown user command.");
			}

			if (cmdObj != null) {
				System.out.println(cmdObj.execute(v, cmdParts));
			}

			inputLine = "";
		}

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

Sample output (User inputs are in green text)

```
Welcome to COMP2396 Assignment 3 - Vending Machine
Add Cocacola 1
Added Cocacola for 1 can(s).
Add Pepsi 3
Added Pepsi for 3 can(s).
Add Sprite 2
Added Sprite for 2 can(s).
Add Mirinda 2
Added Mirinda for 2 can(s).
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 $10, $2, $1. $13 in total.
Reject
```

Refer to note 1 for the format of listing of coins

```
Rejected no coin!
Check Pepsi
Pepsi: Price = 5, Quantity = 3.
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: $10, $1.
Check Pepsi
Pepsi: Price = 5, Quantity = 1.
Insert 1
Inserted a $1 coin. $1 in total.
Insert 1
Inserted a $1 coin. $2 in total.
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!
Insert 2
Inserted a $2 coin. $6 in total.
Buy Sprite
Dropped Sprite. Paid $6. No change.
Insert 10
Inserted a $10 coin. $10 in total.
Buy Mirinda
Dropped Mirinda. Paid $10. Your change: $2, $1.
Exit
Bye
```

Refer to note 1 for the format of listing of coins

Refer to note 1 for the format of listing of coins

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);
	}

	 public void initializeProducts() {

	 /* Write some code here so that all the products are initialized with quantity
	 0 with their corresponding price. Please refer to Notes 6 in page 5 for the
	 products and their corresponding price. */
	 }

	/* 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[] cmdParts);
}
```

You need to implement 5 commands for the system, namely CmdInsertCoin, CmdCheckProductInfo, CmdRejectCoins CmdPurchase and CmdAddProduct. 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 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. CmdCheckProductInfo – Check the information of one product. The command should report the price and the quantity of the product.

3. CmdRejectCoins – Reject all coins from Coin Slot. This command should report the total amount of credit rejected.
4. 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 status of the transaction and how much change is returned to the customer. Refer to Note 3 below for more details.
5. CmdAddProduct- Add the quantity of one product (including its name and quantity). The command should report how many of the product has been added.

For example, CmdInsertCoin inherits Command Interface and performs inserting coin action in the execute() method:

```
public class CmdInsertCoin implements Command {

	@Override
	public String execute(VendingMachine v, String[] cmdParts) {
		Integer c = Integer.valueOf(cmdParts[1]);
		// Add the coin to Coin Slot
		v.insertCoin(c);
		// Do something
		// return a string "Inserted a $x coin. $y in total."
		// x and y must be replaced by an appropriate value

	}
}
```

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 the 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 descendingly.
2. Only $1, $2, $5 and $10 coins exist. No need to deal with other values of coins or cents.
3. When preparing coins for a change, the vending machine only looks for coins in the **Coin Changer**, where 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 a $7 change is needed, you should select [$5, $2] in order instead of selecting [$2, $2, $2, $1] or [$5, $1, $1]. Please also refer to the sample output for reference.
4. All program output should be produced via the return of the execute() in command objects and the main program. In other words, you cannot use System.out.println() or other print functions in your own code.
5. You are welcome to add other classes that help you to implement the system. You can upload up to 10 files to Moodle.
6. In test cases, the products are only Cocacola ($4) ,Pepsi ($5), Sprite ($6), Mirinda ($7), Gatorade ($8), Bonaqua ($11), RedBull ($12), Tropicana ($15), MinuteMaid ($10), with no other products. You are **not** required to consider the situation that “Check” or “Buy ” or “Add” an unknown product.

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

CmdCheckProductInfo.java	No, you need to create it	Yes
CmdPurchase.java	No, you need to create it	Yes
CmdRejectCoins.java	No, you need to create it	Yes
CmdAddProduct.java	No, you need to create it	Yes

Submission and marking:

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

You must evaluate your program before the assignment due date in order to get marks. You should make sure the program output is an exact match with the expected output. Any mismatch, such as an additional or missing space character, will lead to 0 mark.

The full mark of the assignment is 100%. There would be 1 test case before the deadline for you to test your program. Another 10 hidden test cases would be used after the deadline to evaluate this assignment.

You will get 0 mark if:

- You submit .class files instead of .java source files, or
- You submit java source files that cannot be compiled, or
- You submit java source files that are downloaded from the Internet, or
- You submit java source files from your classmates, or
- You submit java source files from friends who took this course last year, or
- You submit java source files that is not compilable or executable with the given **Main class and Command Class**.

Main.java and Command.java ensure you interact with the vending machine via command objects. You cannot modify the program code in the two files. TA will check your submission manually after the deadline. If we discover any modification, 50% marks penalty shall be imposed.

Note the following:

- Double check your submission. Please check the assignment page again after submission to see all files you have submitted.
- You must evaluate your program before the assignment due date to get marks. **Late submission / evaluation is not allowed!**
- You are encouraged to keep an optional GitHub private repo for the assignment as a record. Please refer to the guidelines on Moodle for more details.
- If you encounter any technical problems, please contact us as soon as possible
- We will use more test cases to test your program outputs against standard outputs after the submission deadline.

Usage of LLM

- Usage of LLMs is strictly prohibited for ALL questions in this assignment.
- All answers must be your own work. When LLM is forbidden for a question, any form of consultation with LLM with regards to that question is treated as plagiarism.

-END-