FREIGHTOS GROUP
WebCargo | Freightos.com | Freightos Data

Software Development Engineer Qualification Assignment

Name: Saleh Al-Qassis
Date: 13/11/2022

Thank you for your interest in joining Freightos team and congratulations on being shortlisted for this employment opportunity at Freightos.

This 24-hour assignment will test your skills relating to the professional execution of the Software Development Engineer position major job responsibilities. This assignment will test your level of knowledge mainly in Computer Science Fundamentals (coding, algorithms and data structures), OO, and problem solving among other skills.

Introduction

Vending machines can be be of different types. Some vending machines are dedicated to coffee, some are dedicated to drinks, and others are dedicated to snacks. For this assignment, you will implement a Snack Vending machine.

Deliverables:

1. Provide a UML diagram/s to illustrate your understanding of Object Oriented design techniques and concepts.
2. Provide a code skeleton to the design.
3. Implement the SnackMachine class
4. Provide a test suite to the main class.

Specifications of the Snack Machine

The Snack Machine has the following characteristics:

- **Money Slots:** the machine accepts money of the following types:
  - CoinSlot: There are four denominations: • 10c • 20c • 50c • $1
  - CardSlot : all cards accepted
  - Notes Slot :20$ and 50$ only
  - Machine only accepts USD currency
- **Snack Slots**
  - The machine has five rows to display snack items.
  - Each row has 5 columns to pile the items.
  - Each column has a number.
- **Keypad**
  Users can select the items to be purchased using a keypad.

Purchase a Snack Use Case

Basic Flow

1. This use case begins when the customer wants to purchase snacks.
2. The customer selects a number by pressing on the keypad.
3. The VM displays a message that the snack is available for the selected number and displays its price.
4. The customer inserts the money.
5. The VM validates the money.
6. The VM accepts the money.
7. The VM displays the accumulated amount of money each time a new money is entered.
8. The VM monitors the amount of the accepted money, If the money is enough, the VM dispenses the selected snack to the customer.
9. The VM determines if any change should be sent back to customer.
10. The VM displays the change at panel.
11. Then, the VM dispenses change.

[P.S You are required to come up with alternative scenarios to this basic flow.]

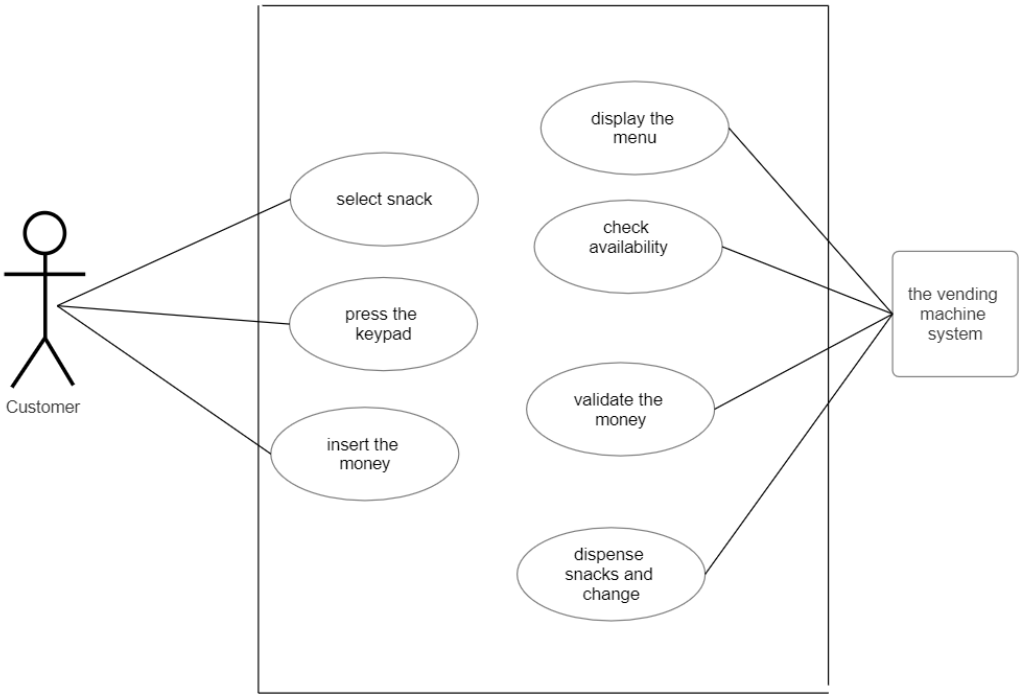
Solution:

Purchase a Snack Use Case

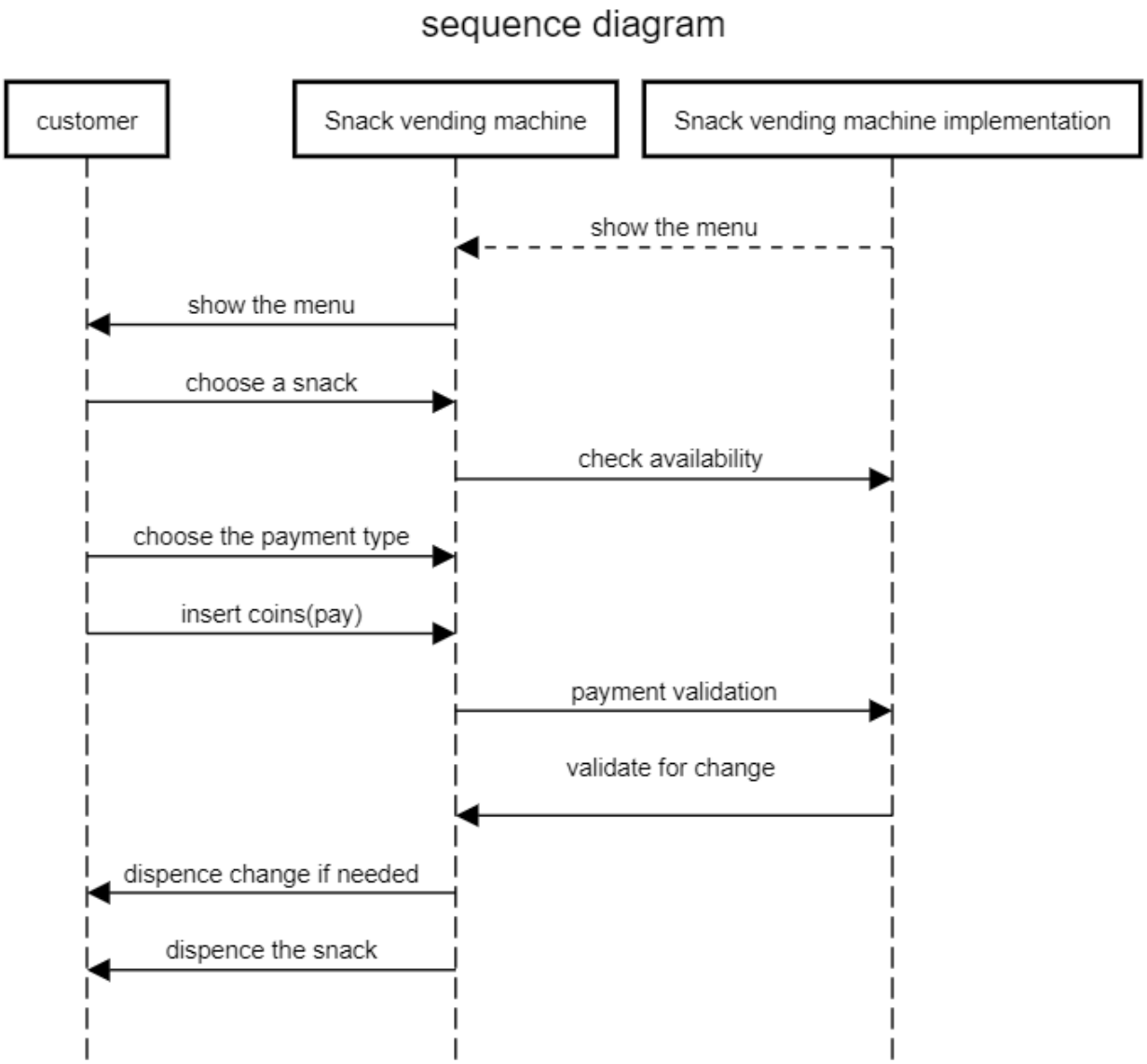
1. This use case begins when the customer wants to purchase snacks.
2. The VM displays the snacks and displays its price and location
3. The customer selects the location by pressing on the keypad.
4. The VM validates that the snack is available.
5. The customer chooses the payment type.
6. The customer inserts the money according to the instructions.
7. The VM validates the money.
8. The VM accepts the money.
9. The VM check if the money was the enough.
10. The VM determines if any change should be sent back to customer.
11. Then, the VM dispenses change if there was any for the customer.
12. The VM the VM dispenses the selected snack to the customer.
13. The VM displays the change at panel.

1-diagrams

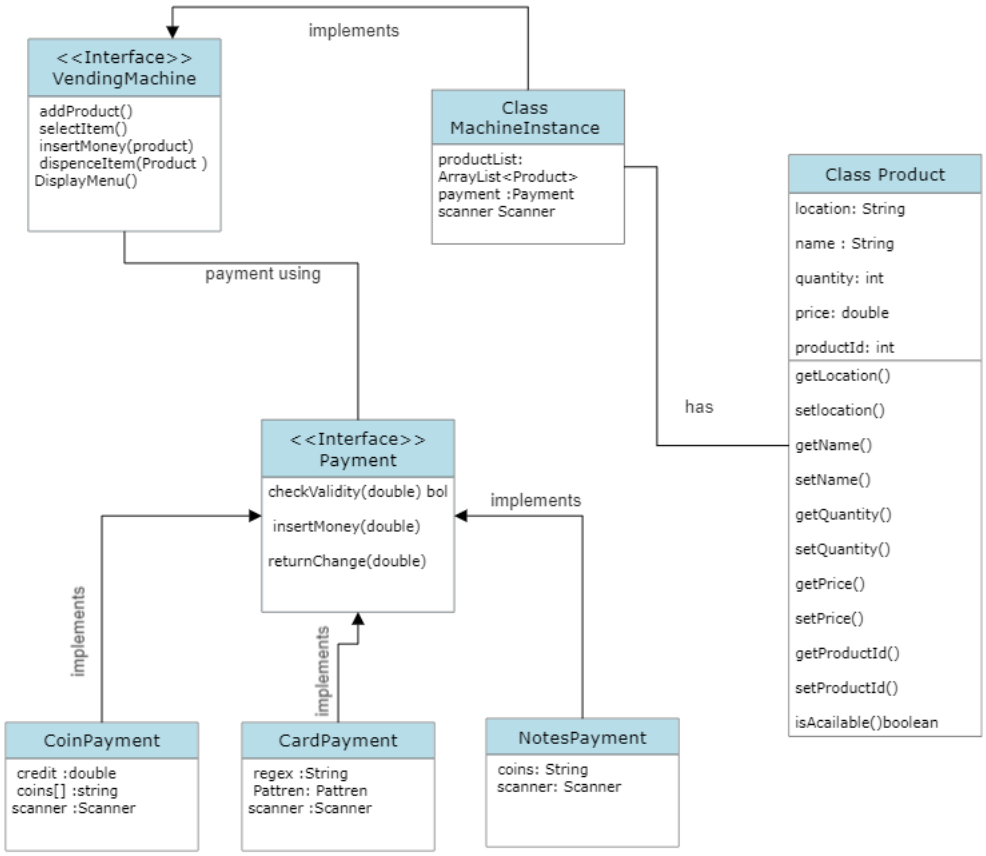
1.1 -Use case diagram



1.2 -Sequence diagram



1.3- class uml diagram



2- code skeleton

1-Interface:

Vending machine

main > java > com > freightos > SnackVendingMachine > J VendingMachine.java

```java
package com.freightos.SnackVendingMachine;
//controle the whole opperations from here

public interface VendingMachine {
    void addProduct(); add products


    void selectItem(); choose the product

    void insertMoney(Product item);  pay for the product

    void returnChange(); dispense the change

    void dispenceItem(Product item); drop the product if all the opperations went ok
     void DisplayMenu(); menu for the products
}

```

Payment

```java
package com.freightos.SnackVendingMachine;

public interface Payment {

    void returnChange(double change);

    boolean checkValidity(double money);

    void insertMoney(double price);
}

```

2-implementation

```
public class MachineInstance implements VendingMachine {
    ArrayList<Product> productList = ""
    Scanner scanner = ""
    Payment payment;

    public void addProduct() {
        // row 1 A
        add products
        productList.add(new Product("A1", "lays original", 7, 1, 1));

    }

    public void selectItem() {
        // TODO Auto-generated method stub
        System.out.println(" choose your snack !.");
        DisplayMenu();



        switch (chosenItem) {
        // ROW A
        case "A1":
            if (productList.get(0).isAvailable()) {

                insertMoney(productList.get(0));
            } else {
                System.out.println("NOT AVAILABLE.");
                System.out.println("-----------------------------------");
                break;
            }
```

```
    // QUIT
    case "B":
        DisplayMenu();
        break;
    // others
    default:
        System.out.println("INVALID LOCATION : (TRY AGAIN)");
        System.out.println("-------------------------------------");
        break;
    }

}

public void insertMoney(Product item) {
    // TODO Auto-generated method stub

    choose payment method

    int inputPayMethod = scanner.nextInt();

    switch (inputPayMethod) {
    case 1:
        payment = new coinPayment();
        payment.insertMoney(item.getPrice());
        dispenceItem(item);

        System.out.println("\nTHANK YOU! HAVE A NICE DAY! :)\n\n");
        System.out.println("-------------------------------------");
        DisplayMenu();

        break;

    case 2:
        payment = new CardPayment();
        payment.insertMoney(item.getPrice());
        dispenceItem(item);

        System.out.println("SUCCESSFULLY PURCHASED " + item.getName());
```

```
            break;


        case 4:
            DisplayMenu();
        default:
            System.out.println("INVALID INPUT");
        }

    }

    public void returnChange() {
        // TODO Auto-generated method stub

    }

    public void dispenceItem(Product item) {
        // TODO Auto-generated method stub
    ArrayListItemCount--
    }

    public void DisplayMenu() {

    display the menu
    }
```

Card payment

```java
package com.freightos.snackvendingmachine;

import java.util.Scanner;
import java.util.concurrent.TimeUnit;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class CardPayment implements Payment {
    Scanner scanner = new Scanner(System.in);
    String regex ="";
    Pattern pattern = Pattern.compile(regex);

    @Override
    public void returnChange(double change) {
        // TODO Auto-generated method stub

    }

    @Override
    public boolean checkValidity(double price) {
        // TODO Auto-generated method stub
    //  should check credit card and if it contains the money here
        return true;
    }

    @Override
    public void insertMoney(double price) {
        // TODO Auto-generated method stub
        System.out.println("ENTER VALID CARD NUMBER");
        String CardNumber = scanner.nextLine();
        CardNumber = CardNumber.replaceAll("-", "");

        System.out.println("VALIDATING CARD...");

          checkValidity(price);

    }

}
```

Coin payment

```java
package com.freightos.SnackVendingMachine;

import java.util.Scanner;

public class coinPayment implements Payment {
    Scanner scanner = new Scanner(System.in);
    double credit = 0;
    @Override
    public boolean checkValidity(double money) {
        // TODO Auto-generated method stub
        boolean bool = false;
        if (not included in the currancy) {
            bool = true;}
        return bool;
    }

    @Override
    public void insertMoney(double price) {
        // TODO Auto-generated method stub
        double money;
        System.out.println("Enter the number of coins you want to pay with.");
        int numberOfCoins = scanner.nextInt();
        System.out.println("Enter the coins :");
        for (number of coins) {
            enter coins;
            if (valid money) {
                credit = credit + money;
            }else {
                enter money again;}
        }
        if (affordable) {
            give back change
        }else {
            quit the opperation not enough money        }}
    @Override
    public void returnChange(double change) {
        card pays the exaxt amount no change needed }
```

Notes Payment

```java
package com.freightos.SnackVendingMachine;

import java.util.Scanner;

public class NotesPayment implements Payment{
Scanner scanner=new Scanner(System.in);

    @Override
    public void returnChange(double change) {
        get change
    }

    @Override
    public boolean checkValidity(double money)
        is money AVAILABLE
    }

    @Override
    public void insertMoney(double price) {
        take the coins from the customer
        }



}

}
```

The main method:

main > java > com > freightos > SnackVendingMachine > J App.java

```java
package com.freightos.SnackVendingMachine;


public class App
{
    public static void main( String[] args )
    { MachineInstance m= new MachineInstance();
run the machine instance

    }
}

```