Code for the Vending Machine

Write the object-oriented code to implement the design of the vending machine problem.

We'll cover the following

We've reviewed different aspects of the vending machine problem and observed the attributes attached to the problem using various UML diagrams. Let us now explore the more practical side of things, where we will work on implementing the vending machine using multiple languages. This is usually the last step in an object-oriented design interview process.

We have chosen the following languages to write the skeleton code of the different classes present in the vending machine:

- Java
- C#
- Python
- C++
- JavaScript

Vending machine classes

In this section, we will provide the skeleton code of the classes designed in the class diagram lesson.

> **Note:** For simplicity, we are not defining getter and setter functions. The reader can assume that all class attributes are private and accessed through their respective public getter methods and modified only through their public method functions.

Enumerations

The following code provides the definition of the enumeration used in the vending machine system:

> **Note:** JavaScript does not support enumerations, so we will be using the `Object.freeze()` method as an alternative that freezes an object and prevents further modifications.

```
// Enumerations
enum ProductType {
  CHOCOLATE,
  SNACK,
  BEVERAGE,
  OTHER
}
```

Enum definitions

State

`State` is an interface and the `NoMoneyInsertedState`, `MoneyInsertedState`, and `DispenseState` classes implement the `State` interface. The definition of these classes is provided below:

```
// State is an interface
public interface State {
    // Interface method (does not have a body)
    public void insertMoney(VendingMachine machine, double amount);
    public void pressButton(VendingMachine machine, int rackNumber);
    public void returnChange(double amount);
    public void updateInventory(VendingMachine machine, int rackNumber);
    public void dispenseProduct(VendingMachine machine, int rackNumber);
}

public class NoMoneyInsertedState implements State {
    @override
    public void insertMoney(VendingMachine machine, double amount) {
      // changes state to MonenInsertedState
    }
    public void pressButton(VendingMachine machine, int rackNumber) {}
    public void returnChange(double amount) {}
    public void updateInventory(VendingMachine machine, int rackNumber) {}
    public void dispenseProduct(VendingMachine machine, int rackNumber) {}
}

public class MoneyInsertedState implements State {
    @override
    public void insertMoney(VendingMachine machine, double amount) {}
    public void pressButton(VendingMachine machine, int rackNumber) {
        // check if product item is available
        // validate money
        // change state to DispenseState
    }
    public void returnChange(double amount) {}
```

The State interface and its derived classes

Product, rack and inventory

We will explore the `Product`, `Rack`, and `Inventory` classes that provide the details of the items available as well as their positions inside the vending machine. The definition of these classes is provided below:

```
public class Product {
  private String name;
  private int id;
  private double price;
  private ProductType type;
}

public class Rack {
  private List<int> productIds;
  private int rackNumber;

  public boolean isEmpty();
}

public class Inventory {
  private int noOfProducts;
  private List<Product> products;

  public void addProduct(int productId, int rackId);
  public void removeProduct(int productId, int rackId);
}
```

The Product, Rack, and Inventory classes

Vending machine

The `VendingMachine` class is the final class of the system, and it will also be a Singleton class so that there will only be a single instance of this class in the whole system. The definition of this class is given below:

```
public class VendingMachine {
    private State currentState;
    private double amount;
    private int noOfRacks;
    private List<Rack> racks;
    private List<int> availableRacks;

    // The VendingMachine is a Singleton class that ensures it will have only one active instance at a time
    private static VendingMachine vendingMachine = null;

    // Created a private constructor to add a restriction (due to Singleton)
    private VendingMachine();

    // Created a static method to access the singleton instance of VendingMachine
    public static VendingMachine getInstance() {
        if (vendingMachine == null) {
            vendingMachine = new VendingMachine();
        }
        return vendingMachine;
    }

    public void insertMoney(double amount) {}
    public void pressButton(int rackNumber) {}
    public void returnChange(double amount) {}
    public void updateInventory(int rackNumber) {}
    public void dispenseProduct(int rackNumber) {}
    public int getProductIdAtRack(int rackNumber) {}
}
```

The VendingMachine class

Wrapping up

We've explored the complete design of a vending machine in this chapter. We've looked at how a basic vending machine can be visualized using various UML diagrams and designed using object-oriented principles and design patterns.

Back

Activity Diagram for the Vending Machine

Complete

Next

Getting Ready: Online Blackjack Game