Code of Elevator System

Write object-oriented code to implement the design of the elevator system problem.

We'll cover the following

We've discussed different aspects of the elevator system and observed the attributes attached to the problem using various UML diagrams. Let's explore the more practical side of things, where we will work on implementing the elevator system 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 elevator control system:

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

Elevator system classes

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

> **Note:** For simplicity, we aren't 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 methods function.

Enumerations

First of all, we will define all the enumerations required in the elevator system. According to the class diagram, there are three enumerations used in the system i.e., `ElevatorState`, `Direction` and `DoorState`. The code to implement these enumerations is as follows:

> **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.

```
// definition of enumerations used in elevator system
enum ElevatorState {
    IDLE,
    UP,
    DOWN
}

enum Direction {
    UP,
    DOWN
}

enum DoorState {
    OPEN,
    CLOSE
}
```

Enum definitions

Button

This section contains the implementation of a `Button` class and its subclasses which are `HallButton` and the `ElevatorButton`. The `Button` class has a pure virtual function `isPressed()` in it. The code to implement this relationship is given below:

```
public abstract class Button {
    private boolean status;

    public pressDown();
    public abstract boolean isPressed();
}

public class DoorButton extends Button {

    public boolean isPressed() {
        // Definition
    }
}

public class HallButton extends Button  {
    private Direction buttonSign;

    public boolean isPressed() {
        // definition
    }
}

public class ElevatorButton extends Button  {
    private int destinationFloorNumber;

    public boolean isPressed() {
        // definition
    }
}
```

Button and its subclasses

Elevator panel and hall panel

`ElevatorPanel` and the `HallPanel` are classes which use the instance of `ElevatorButton` and `HallButton` respectively. The code to implement these classes is provided below:

```
public class ElevatorPanel {
    private List<ElevatorButton> floorButtons;
    private DoorButton openButton;
    private DoorButton closeButton;
}

public class HallPanel {
    private HallButton up;
    private HallButton down;
}
```

The ElevatorPanel and HallPanel classes

Display

This component shows the implementation of the `Display` class. This class is responsible for showing the display inside and outside of the elevator cars. The code to implement this class is shown below:

```
public class Display {
    private int floor;
    private int capacity;
    private Direction direction;

    public void showElevatorDisplay();
    public void showHallDisplay();
}
```

The Display class

Elevator car

This section contains the definition of the `ElevatorCar` class. An elevator car contains the instance of `Door`, `Display`, and `ElevatorPanel`. The implementation of this class is represented below:

```
public class ElevatorCar {
    private int id;
    private Door door;
    private ElevatorState state;
    private Display display;
    private ElevatorPanel panel;

    public void move();
    public void stop();
    public void openDoor();
    public void closeDoor();
}
```

The ElevatorCar class

Door and floor

This section contains the code for the `Door` and `Floor` classes. In the `Door` class, the enumeration `DoorState` is used and the `Floor` class contains the instances of `Display` and `HallPanel`. The implementation of this class is given below:

```
public class Door {
    private DoorState state;
    public boolean isOpen();
}

public class Floor {
    private List<Display> display;
    private List<HallPanel> panel;

    public boolean isBottomMost();
    public boolean isTopMost();
}
```

The Door and Floor classes

Elevator system and building

The final class of an elevator system is the `ElevatorSystem` class which will be a Singleton class, which means that the entire system will have only one instance of this class. Moreover, there is a `Building` class that contains the instances of `Floor` and `ElevatorCar`. The implementation of these Singleton classes are provided below:

```
public class ElevatorSystem {
    private Building building;
    public void monitoring();
    public void dispatcher();

    // Private constructor to prevent direct instantiation
    private ElevatorSystem() {
        // Initialize the ElevatorSystem
    }

    // The ElevarSystem is a singleton class that ensures it will have only one active instance at a time
    private static ElevatorSystem system = null;

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

public class Building {
    private List<Floor> floor;
    private List<ElevatorCar> elevator;

    private static Building building = null;

    public static Building getInstance() {
        if (building == null) {
```

The ElevatorSystem and Building classes

Wrapping up

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

← Back

Activity Diagram for the Elevator System

Complete

Next →

Getting Ready: Library Management System