Model-View-Controller (MVC) using Observer Pattern

Affan Rauf

September 20, 2024

What is MVC?

- **MVC (Model-View-Controller)** is an architectural pattern used for building user interfaces and applications.
- It divides an application into three interconnected components:
  - **Model**: Represents the data and business logic.
  - **View**: Responsible for displaying data to the user.
  - **Controller**: Handles user input and updates the Model.
- MVC improves separation of concerns and maintainability.

Breakdown of MVC Components

- **Model**
  - Manages the application's data.
  - Notifies the View when data changes.
- **View**
  - Displays data from the Model to the user.
  - Updates when the Model changes.
- **Controller**
  - Handles user input (e.g., button clicks).
  - Updates the Model based on user interactions.

How Does the Observer Pattern Fit into MVC?

- **Observer pattern** helps connect the **Model** and **View** components.
- The **View** acts as an Observer to the **Model** (the Subject).
- Whenever the Model's data changes, it notifies the View (via the Observer pattern) to update the display.
- This decouples the Model from the View, promoting reusability and flexibility.

MVC Flow Using Observer Pattern

1. **User interacts with the View** (e.g., clicks a button).
2. **View informs the Controller** about the action.
3. **Controller updates the Model** (e.g., changes data).
4. **Model notifies the View** via the Observer pattern.
5. **View updates the UI** to reflect changes in the Model.

Example: Simple MVC Application

Model Class:

```
class Model {
    private int counter;
    private List<Observer> observers = new ArrayList<>();

    public void addObserver(Observer o) { observers.add(o); }
    public void increment() {
        counter++;
        notifyObservers();
    }
    public int getCounter() { return counter; }

    private void notifyObservers() {
        for (Observer o : observers) {
            o.update();
        }
    }
}
```

Example: View and Controller

View Class (Observer)[1]:

```
class View implements Observer {
    public void update(int counter) {
        System.out.println("Counter: " + counter);
    }
}
```

[1] The code specific to AWT components has been omitted from the slides.

Example: View and Controller

View Class (Observer)[1]:

```
class View implements Observer {
    public void update(int counter) {
        System.out.println("Counter: " + counter);
    }
}
```

Controller Class:

```
class Controller {
    private Model model;

    public Controller(Model model) {
        this.model = model;
    }

    public void onButtonClick() {
        model.increment();
    }
}
```

[1] The code specific to AWT components has been omitted from the slides.

Main Method to Glue MVC Components

```
public class Main {
    public static void main(String[] args) {
        Model model = new Model();
        View view = new View();
        Controller controller = new Controller(model);

        // Register the View as an Observer of the Model
        model.addObserver(view);

        // Simulate a button click in the Controller
        controller.onButtonClick();  // Output: Counter: 1
        controller.onButtonClick();  // Output: Counter: 2
    }
}
```

Code Explanation

- **Model**: Keeps track of the counter and notifies observers (Views) when the data changes.
- **View**: Implements the ‘Observer‘ interface to update the UI when the Model changes.
- **Controller**: Handles user actions (e.g., button clicks) and modifies the Model.

Benefits of MVC with Observer Pattern

- **Separation of Concerns**: Clear division between data (Model), presentation (View), and logic (Controller).
- **Reusability**: Views can be reused for different Models.
- **Loose Coupling**: The Model and View are decoupled through the Observer pattern, making the system more flexible.
- **Scalability**: Easier to extend or modify parts of the application without impacting others.