Understanding MVC Architecture with Java

Dr. Ekincan Ufuktepe

CMP_SC 3330 – Object-oriented Programming

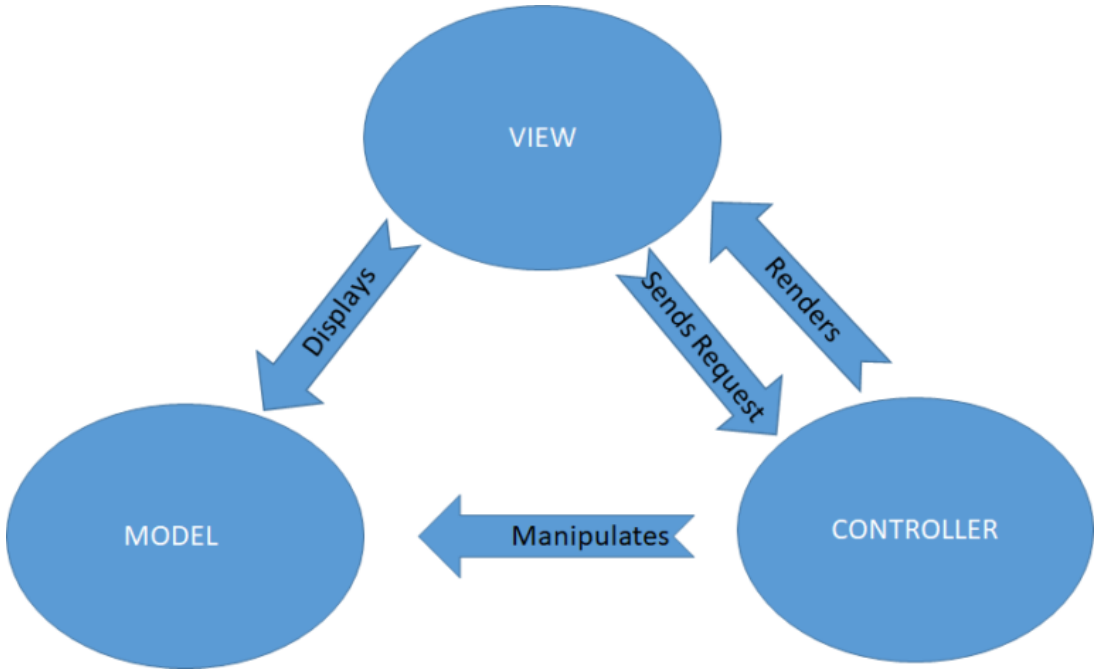
University of Missouri – Columbia
Electrical Engineering & Computer Science

Introduction to MVC Architecture

- Definition of MVC (Model-View-Controller) Architecture
- Explanation of the three components: Model, View, Controller
- Purpose of MVC: Separation of concerns, enhancing maintainability, and scalability

MVC Components

- Model:
  - Represents the data and business logic of the application
  - Independent of the user interface
- View:
  - Represents the presentation layer
  - Displays information to the user
- Controller:
  - Mediates between the Model and View
  - Handles user input and updates the Model accordingly



Detailed Explanation of Components

- Model:
  - Contains application data and business logic
  - Independent of user interface
  - Examples: POJOs (Plain Old Java Objects), Data Access Objects (DAOs), Service Classes

Detailed Explanation of Components (cont.)

- View:
  - Represents the presentation layer
  - Displays the data from the model
  - Examples: JSP (JavaServer Pages), HTML, Swing components

Detailed Explanation of Components (cont.)

- Controller:
  - Receives user input
  - Interacts with both model and view
  - Orchestrates the flow of data and operations
  - Examples: Servlets, Spring MVC Controllers

Benefits of MVC

- Separation of concerns: Enhances maintainability and scalability
- Reusability: Components can be reused across different views
- Testability: Easy to unit test each component independently

Implementing MVC in Java

- Use case scenario: Building a simple web application
- Demonstration of how each component interacts in a Java application

Sample Code - Model

```
public class UserModel {
    private String username;
    private String password;

    // Getters and setters
}
```

Sample Code - View

```
public class UserView extends JFrame {
    private JTextField usernameField;
    private JPasswordField passwordField;
    private JButton registerButton;

    public UserView() {
        initialize();
    }
```

Sample Code – View (contd.)

```
private void initialize() {
        setTitle("User Registration");
        setSize(300, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        getContentPane().setLayout(null);

        usernameField = new JTextField();
        usernameField.setBounds(30, 30, 200, 25);
        getContentPane().add(usernameField);

        passwordField = new JPasswordField();
        passwordField.setBounds(30, 70, 200, 25);
        getContentPane().add(passwordField);

        registerButton = new JButton("Register");
        registerButton.setBounds(80, 120, 100, 30);
        getContentPane().add(registerButton);
        setVisible(true);
    }
```

Sample Code – View (contd.)

```
// Getter methods for username, password, and register button
    public String getUsername() {
        return usernameField.getText();
    }

    public String getPassword() {
        return new String(passwordField.getPassword());
    }

    public void addRegisterListener(ActionListener listener) {
        registerButton.addActionListener(listener);
    }
}
```

Sample Code - View (a JSP example)

```
<html>
<head><title>User Registration</title></head>
<body>
    <form action="register" method="post">
        Username: <input type="text"
name="username"><br>
        Password: <input type="password"
name="password"><br>
        <input type="submit" value="Register">
    </form>
</body>
</html>
```

Sample Code - Controller

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;


public class RegistrationController {
    private UserModel userModel;
    private UserView userView;


    public RegistrationController(UserModel
userModel, UserView userView) {
        this.userModel = userModel;
        this.userView = userView;
        this.userView.addRegisterListener(new
RegisterListener());
    }


}
```

```
 class RegisterListener implements ActionListener
{
        @Override
        public void actionPerformed(ActionEvent e)
{
            String username =
userView.getUsername();
            String password =
userView.getPassword();


            userModel.setUsername(username);
            userModel.setPassword(password);


            // Process registration logic,
interact with DAOs, etc.
        }
    }
```

Best Practices and Tips

- Keep models lightweight and focused on business logic
- Views should be dumb and avoid containing business logic
- Controllers should be lean, avoiding complex logic
- Use GUI builders like WindowBuilder to streamline Swing UI development