REPORT NO. 7: Implementation of Architectural Design Pattern(MVVM:Model-View-ViewModel)

Course Code: CSE 404

Course Title: Software Engineering and ISD Laboratory

Submitted by

SHANJIDA ALAM(ID: 353)

Submitted to

Dr. Md. MUSHFIQUE ANWAR, Professor

Dr. Md. HUMAYUN KABIR, Professor

Computer Science and Engineering

Jahangirnagar University

Dhaka, Bangladesh

October 24, 2024

Contents

Chapter 1

Introduction

The **Model-View-ViewModel (MVVM)** architecture is a software design pattern that promotes separation of concerns by organizing code into three distinct layers: **Model**, **View**, and **ViewModel**. It is particularly popular in mobile and desktop applications due to its ability to improve code modularity, maintainability, and scalability.

- **Model:** This layer is responsible for handling the business logic and data-related tasks. It represents the data or information that the application works with, including communicating with databases, performing calculations, or managing network requests.
- **View:** The View layer is responsible for displaying data to the user and handling the user interface (UI).
- **ViewModel:** The ViewModel acts as a bridge between the View and the Model. It holds and prepares the data from the Model to be displayed by the View. The ViewModel is responsible for processing user actions (captured by the View) and providing the necessary data to the View in a format that can be easily displayed.

Chapter 2

Objectives

The primary objectives of this lab task are:

- To gain hands-on experience in applying an architectural design pattern (MVVM) in software development, ensuring a clean separation of concerns between data handling, business logic, and user interface.
- Improve the quality of Javadoc comments, including descriptive comments, tags, and formatting.
- Implement the MVVM Pattern in Mathematical Operations.
- To follow proper coding standards, including clear naming conventions, code documentation, and error handling, ensuring that the code is readable, scalable, and easy to debug.

Chapter 3

Source Code: MultiplicationModel Class (Model Class)

In this chapter, we will explore an example Java class that demonstrates architectural design pattern. The class provided is called `MultiplicationModel` class creates under the **model** packages.

```java
package com.example.sw_lab_07.model;

/**
 * This class calculates the multiplication of two numbers.
 *
 * @author Shanjida
 * @version 1.0
 * @since 03/10/2024
 */

public class MultiplicationModel {
    /**
     * This method handles the multiplication of two numbers
     * @param firstNumber is the first number
     * @param secondNumber is the second number
     * @return product of the two numbers
     */
    public int multiply(int firstNumber, int secondNumber) {
        return firstNumber * secondNumber;
    }

    /**
     * Overloaded method to multiply three numbers
     * @param firstNumber is the first number
```

```
25       * @param secondNumber is the second number
26       * @param thirdNumber is the third number
27       * @return product of the three numbers
28       */
29      public int multiply(int firstNumber, int secondNumber, int
        thirdNumber) {
30          return firstNumber * secondNumber * thirdNumber;
31      }
32  }
```

Listing 3.1: MultiplicationModel Class in Java

Chapter 4

Source Code: MainActivity Class (View Class)

In this chapter, we will explore an example Java class that demonstrates architectural design pattern. The class provided is called `MainActivity` class creates under the **view** packages.

```
1  package com.example.sw_lab_07.view;
2  import com.example.sw_lab_07.R;
3
4  import android.os.Bundle;
5  import android.text.TextUtils;
6  import android.view.View;
7  import android.widget.Button;
8  import android.widget.EditText;
9  import android.widget.TextView;
10
11 import androidx.appcompat.app.AppCompatActivity;
12 import androidx.lifecycle.ViewModelProvider;
13
14 import com.example.sw_lab_07.viewModel.MultiplicationViewModel;
15
16 public class MainActivity extends AppCompatActivity {
17
18     private EditText input1, input2, input3;
19     private TextView resultView;
20     private MultiplicationViewModel multiplicationViewModel;
21
22     @Override
23     protected void onCreate(Bundle savedInstanceState) {
24         super.onCreate(savedInstanceState);
```

```
25          setContentView(R.layout.activity_main);
26
27          // Initialize UI components
28          input1 = findViewById(R.id.input1);
29          input2 = findViewById(R.id.input2);
30          input3 = findViewById(R.id.input3);
31          resultView = findViewById(R.id.result_view);
32          Button buttonMultiply = findViewById(R.id.button_multiply);
33
34          // Initialize the ViewModel
35          multiplicationViewModel = new ViewModelProvider(this).get(
    MultiplicationViewModel.class);
36
37          // Set button click listener
38          buttonMultiply.setOnClickListener(new View.OnClickListener()
    {
39              @Override
40              public void onClick(View v) {
41                  // Retrieve input values
42                  String num1Str = input1.getText().toString();
43                  String num2Str = input2.getText().toString();
44                  String num3Str = input3.getText().toString();
45
46                  // Check if input is valid
47                  if (!TextUtils.isEmpty(num1Str) && !TextUtils.isEmpty
    (num2Str)) {
48                      int num1 = Integer.parseInt(num1Str);
49                      int num2 = Integer.parseInt(num2Str);
50
51                      // Check if third input is provided
52                      if (!TextUtils.isEmpty(num3Str)) {
53                          int num3 = Integer.parseInt(num3Str);
54                          // Multiply three numbers
55                          int result = multiplicationViewModel.multiply
    (num1, num2, num3);
56                          resultView.setText("Result: " + result);
57                      } else {
58                          // Multiply two numbers
59                          int result = multiplicationViewModel.multiply
    (num1, num2);
60                          resultView.setText("Result: " + result);
61                      }
62                  } else {
63                      resultView.setText("Please enter at least two
    numbers");
64                  }
```

```
65                  }
66              });
67      }
68  }
```

Listing 4.1: MainActivity Class in Java

Chapter 5

Source Code: MultiplicationViewModel Class (ViewModel Class)

In this chapter, we will explore an example Java class that demonstrates architectural design pattern. The class provided is called `MultiplicationViewModel` class creates under the **ViewModel** packages.

```
1  package com.example.sw_lab_07.viewModel;
2
3  import androidx.lifecycle.LiveData;
4  import androidx.lifecycle.MutableLiveData;
5  import androidx.lifecycle.ViewModel;
6
7  import com.example.sw_lab_07.model.MultiplicationModel;
8
9  public class MultiplicationViewModel extends ViewModel {
10     private final MultiplicationModel multiplication;
11
12     /**
13      * This is the Constructor class
14      */
15     public MultiplicationViewModel() {
16         multiplication = new MultiplicationModel();
17     }
18
19     /**
20      * This method calls the multiplication of two numbers
21      * @param a is the first number
22      * @param b is the second number
```

```
23      * @return the product of the two numbers
24      */
25     public int multiply(int a, int b) {
26         return multiplication.multiply(a, b);
27     }
28 
29     /**
30      * This method calls the multiplication of three numbers
31      * @param a is the first number
32      * @param b is the second number
33      * @param c is the third number
34      * @return the product of the three numbers
35      */
36     public int multiply(int a, int b, int c) {
37         return multiplication.multiply(a, b, c);
38     }
39 }
```

Listing 5.1: MultiplicationViewModel Class in Java

Chapter 6

Code Output

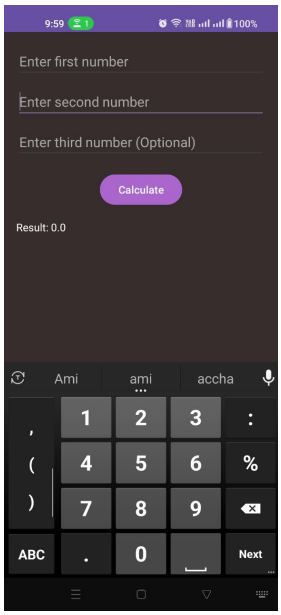

Figure 6.1: User Input Interface

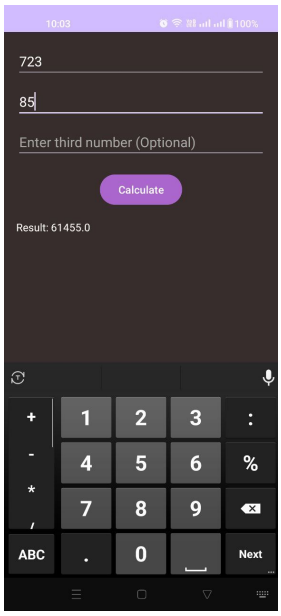

Figure 6.2: Input the two numbers

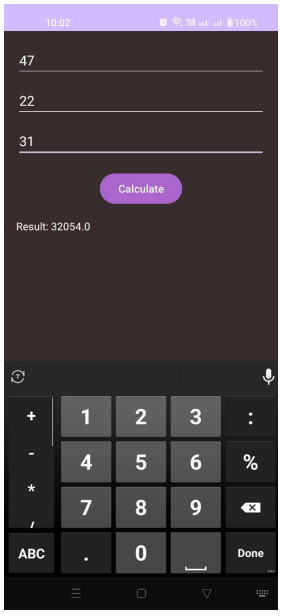

Figure 6.3: Input the three numbers

Chapter 7

Coding Standard

In this part, I would have described which parts of the given code implement the coding standard.

7.1 Naming Conventions

The class follows the usual name standards used by Java:

- **Class Name:** The class name `MultiplicationModel, MultiplicationViewModel` follows **PascalCase**, where the first letter of each word is capitalized. This is the standard naming convention for Java classes.
- **Variable Names:** The instance variable names `firstNumber`, `secondNumber`, `number1`, and `number2` are in **camelCase**, where the first word is lowercase, and each subsequent word begins with a capital letter. These names are also descriptive and clarify the purpose of each variable (i.e., storing integer and floating-point numbers).
- **Function Names:** The function names `multiply(int, int)` and `multiply(int, int)` follow **camelCase** and are descriptive of their functionality. The method names clearly indicate that they multiply two and three numbers.

7.2 Layout Conventions

The class follows the usual layout conventions used by Java:

- **Indentation:** Use 4 spaces per indentation level. Do not use tabs.
- **Braces:** Always put the opening brace on the same line as the statement.

7.3 Code Comments

- **Function and Class Comments:** Each function/class should have a Javadoc comment to describe its purpose, parameters, and return value (for functions).Here is the segment of code where I have implemented the Javadoc comments:

```
1 /**
2  * Multiplies two {@code int} values and returns the result.
3  *
4  * @param firstNumber  The first {@code int} to multiply.
5  * @param secondNumber The second {@code int} to multiply.
6  * @return The result of multiplying {@code firstNumber} and {
     @code secondNumber}, as an {@code int}.
7  */
8 public int multiplyTwoNumbers(int firstNumber, int secondNumber)
      {
9     return firstNumber * secondNumber;
10 }
```

Listing 7.1: Function: multiplyTwoNumbers

Chapter 8

Workflow of the MVVM architectural pattern

In this part, I would have explained the how the architectural pattern is applied in my code segment.

- **Step 1: Define the Model**
  Create the core business logic for the application.Multiplication mathematical operation is implemented as a separate class. This is the **Model** layer in MVVM.The Model handles data manipulation and business logic. For example, the `MultiplicationMode` class will include methods for multiplying two or three numbers.

```
1  package com.example.sw_lab_07.model;
2
3  /**
4   * This class calculates the multiplication of two numbers.
5   *
6   * @author Shanjida
7   * @version 1.0
8   * @since 03/10/2024
9   */
10
11 public class MultiplicationModel {
12     /**
13      * This method handles the multiplication of two numbers
14      * @param firstNumber is the first number
15      * @param secondNumber is the second number
16      * @return product of the two numbers
17      */
18     public int multiply(int firstNumber, int secondNumber) {
19         return firstNumber * secondNumber;
```

```
20      }
21
22      /**
23       * Overloaded method to multiply three numbers
24       * @param firstNumber is the first number
25       * @param secondNumber is the second number
26       * @param thirdNumber is the third number
27       * @return product of the three numbers
28       */
29      public int multiply(int firstNumber, int secondNumber, int
        thirdNumber) {
30          return firstNumber * secondNumber * thirdNumber;
31      }
32  }
```

Listing 8.1: MultiplicationModel Class in Java

- **Step 2: Create the ViewModel**
  Bridge the gap between the data (Model) and the UI (View).The ViewModel layer is responsible for preparing data for the UI, responding to user actions, and calling methods from the **Model**.For multiplication math class, a corresponding ViewModel `MultiplicationViewModel` is created, which interacts with the Model. The ViewModel exposes data as `LiveData` to observe changes, allowing the View to automatically update without needing to directly call the Model.

```
1  package com.example.sw_lab_07.viewModel;
2
3  import androidx.lifecycle.LiveData;
4  import androidx.lifecycle.MutableLiveData;
5  import androidx.lifecycle.ViewModel;
6
7  import com.example.sw_lab_07.model.MultiplicationModel;
8
9  public class MultiplicationViewModel extends ViewModel {
10     private final MultiplicationModel multiplication;
11
12     /**
13      * This is the Constructor class
14      */
15     public MultiplicationViewModel() {
16         multiplication = new MultiplicationModel();
17     }
18
19     /**
```

```
20      * This method calls the multiplication of two numbers
21      * @param a is the first number
22      * @param b is the second number
23      * @return the product of the two numbers
24      */
25     public int multiply(int a, int b) {
26         return multiplication.multiply(a, b);
27     }
28
29     /**
30      * This method calls the multiplication of three numbers
31      * @param a is the first number
32      * @param b is the second number
33      * @param c is the third number
34      * @return the product of the three numbers
35      */
36     public int multiply(int a, int b, int c) {
37         return multiplication.multiply(a, b, c);
38     }
39 }
```

Listing 8.2: MultiplicationViewModel Class in Java

- **Step 3: Implement the View**
  Build the user interface that interacts with the user.The View component is typically an Activity or Fragment in Android. It consists of UI elements like input fields, buttons, and text views.The View is bound to the ViewModel, often using data binding to establish connections between UI elements and the data provided by the ViewModel.The View observes changes in the ViewModel and updates the UI accordingly without having to directly manage business logic.

```
1 package com.example.sw_lab_07.view;
2 import com.example.sw_lab_07.R;
3
4 import android.os.Bundle;
5 import android.text.TextUtils;
6 import android.view.View;
7 import android.widget.Button;
8 import android.widget.EditText;
9 import android.widget.TextView;
10
11 import androidx.appcompat.app.AppCompatActivity;
12 import androidx.lifecycle.ViewModelProvider;
13
14 import com.example.sw_lab_07.viewModel.MultiplicationViewModel;
15
```

```
16 public class MainActivity extends AppCompatActivity {
17
18     private EditText input1, input2, input3;
19     private TextView resultView;
20     private MultiplicationViewModel multiplicationViewModel;
21
22     @Override
23     protected void onCreate(Bundle savedInstanceState) {
24         super.onCreate(savedInstanceState);
25         setContentView(R.layout.activity_main);
26
27         // Initialize UI components
28         input1 = findViewById(R.id.input1);
29         input2 = findViewById(R.id.input2);
30         input3 = findViewById(R.id.input3);
31         resultView = findViewById(R.id.result_view);
32         Button buttonMultiply = findViewById(R.id.
    button_multiply);
33
34         // Initialize the ViewModel
35         multiplicationViewModel = new ViewModelProvider(this).
    get(MultiplicationViewModel.class);
36
37         // Set button click listener
38         buttonMultiply.setOnClickListener(new View.
    OnClickListener() {
39             @Override
40             public void onClick(View v) {
41                 // Retrieve input values
42                 String num1Str = input1.getText().toString();
43                 String num2Str = input2.getText().toString();
44                 String num3Str = input3.getText().toString();
45
46                 // Check if input is valid
47                 if (!TextUtils.isEmpty(num1Str) && !TextUtils.
    isEmpty(num2Str)) {
48                     int num1 = Integer.parseInt(num1Str);
49                     int num2 = Integer.parseInt(num2Str);
50
51                     // Check if third input is provided
52                     if (!TextUtils.isEmpty(num3Str)) {
53                         int num3 = Integer.parseInt(num3Str);
54                         // Multiply three numbers
55                         int result = multiplicationViewModel.
    multiply(num1, num2, num3);
56                         resultView.setText("Result: " + result);
```

```
57                      } else {
58                          // Multiply two numbers
59                          int result = multiplicationViewModel.
      multiply(num1, num2);
60                          resultView.setText("Result: " + result);
61                      }
62                  } else {
63                      resultView.setText("Please enter at least
      two numbers");
64                  }
65              }
66          });
67      }
68  }
```

Listing 8.3: MainActivity Class in Java

Chapter 9

Conclusion

In this lab exercise, we successfully implemented six mathematical classes using the**Model-View-ViewModel (MVVM)** architectural design pattern. Each class represented a unique mathematical operation, including addition, subtraction, multiplication, division, prime number checking, and finding the maximum value among three numbers. By utilizing **MVVM**, we were able to achieve a clear separation of concerns between data management (Model), user interaction (View), and business logic handling (ViewModel). This separation made the code modular, maintainable, and easily testable.