



OOAD with Java Lab Assignment-1: MVC Spring Framework

2022

Submitted by

PES1UG19CS090, Arvind Krishna

Sem: 6, Sec:B

Step1: Create the Model

'Model' is the part of your application that contains the data and provides the same to the app. It also updates the data on the disk upon receiving a call from the Controller.

Code:

```
public class Model {  
  
    private int x;  
  
    public Model() {  
        this.x = 0;  
    }  
    public void incX() {  
        this.x++;  
    }  
    public void decX() {  
        this.x--;  
    }  
    public int getX() {  
        return this.x;  
    }  
    public void resetX() {  
        this.x = 0;  
    }  
    public void setX(int copy) {  
        this.x = copy;  
    }  
}
```

Step2: Create the View

For this part, you can either use Eclipse IDE or NetBeans. The View segment of your application shows what the Controller allows. In Eclipse, you would have to write the code, whereas NetBeans comes with drag-and-drop functionality to implement a simple GUI. View displays the data from model.

```
import javax.swing.*;
import java.awt.*;

public class View {

    private JFrame frame;
    private JLabel label;
    private JButton incrementButton;
    private JButton decrementButton;
    private JButton resetButton;
    private JButton setButton;
    Controller controller;

    JTextField input1 = new JTextField(10);

    public View() {
        frame = new JFrame("View");
        frame.getContentPane().setLayout(new BorderLayout());
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.setLocationRelativeTo(null);
        frame.setSize(300, 400);
        frame.setLayout(null);
        frame.setVisible(true);

        incrementButton = new JButton("Increase");
        incrementButton.setBounds(50, 30, 200, 40);
        frame.add(incrementButton);
```

```
decrementButton = new JButton("Decrease");
decrementButton.setBounds(50,80,200,40);
frame.add(decrementButton);

label = new JLabel("0");
label.setBounds(150, 130,2000, 10);
frame.getContentPane().add(label);

resetButton = new JButton("Reset");
resetButton.setBounds(50,170,200,40);
frame.add(resetButton);

input1.setBounds(50,220,200,40);

setButton = new JButton("Set value");
setButton.setBounds(50,270,200,40);

frame.add(setButton);
frame.add(input1);
}

public JButton getIncButton(){
    return incrementButton;
}

public JButton getDecButton(){
    return decrementButton;
}

public JButton getResetButton(){
    return resetButton;
}
```

```
public JButton getSetButton() {  
    return setButton;  
}  
  
public void setText(String text) {  
    label.setText(text);  
}  
}
```

Step3. Create the Controller

The Controller is perhaps the most vital segment of the MVC application as it contains the logic for the Model-View interaction. The Controller includes three main functions. These methods are described below.

- **startApplication:** It starts the application by calling the View (In Java's Swing API, you set the setVisible property as true to make the JFrame visible)
- **Extract data from Model**
- **Return data to View**

```
import javax.swing.*;  
import java.awt.*;  
  
public class View {  
  
    private JFrame frame;  
    private JLabel label;  
    private JLabel srn;  
    private JButton incrementButton;  
    private JButton decrementButton;  
    private JButton resetButton;  
    private JButton setButton;  
    Controller controller;  
  
    JTextField input1 = new JTextField(10);  
}
```

```
public View() {  
    frame = new JFrame("PES1UG19CS090");  
    frame.getContentPane().setLayout(new BorderLayout());  
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
    frame.setLocationRelativeTo(null);  
    frame.setSize(300, 400);  
    frame.setLayout(null);  
    frame.setVisible(true);  
  
    incrementButton = new JButton("Increase");  
    incrementButton.setBounds(50, 30, 200, 40);  
    frame.add(incrementButton);  
  
    decrementButton = new JButton("Decrease");  
    decrementButton.setBounds(50, 80, 200, 40);  
    frame.add(decrementButton);  
  
    label = new JLabel("0");  
    label.setBounds(150, 130, 200, 10);  
    frame.getContentPane().add(label);  
  
    resetButton = new JButton("Reset");  
    resetButton.setBounds(50, 170, 200, 40);  
    frame.add(resetButton);  
  
    input1.setBounds(50, 220, 200, 40);  
  
    setButton = new JButton("Set value");  
    setButton.setBounds(50, 270, 200, 40);  
  
    frame.add(setButton);  
}
```

```
frame.add(input1);

srn = new JLabel("PES1UG19CS090");
srn.setBounds(90, 320, 200, 10);
frame.getContentPane().add(srn);
}

public JButton getIncButton() {
    return incrementButton;
}

public JButton getDecButton() {
    return decrementButton;
}

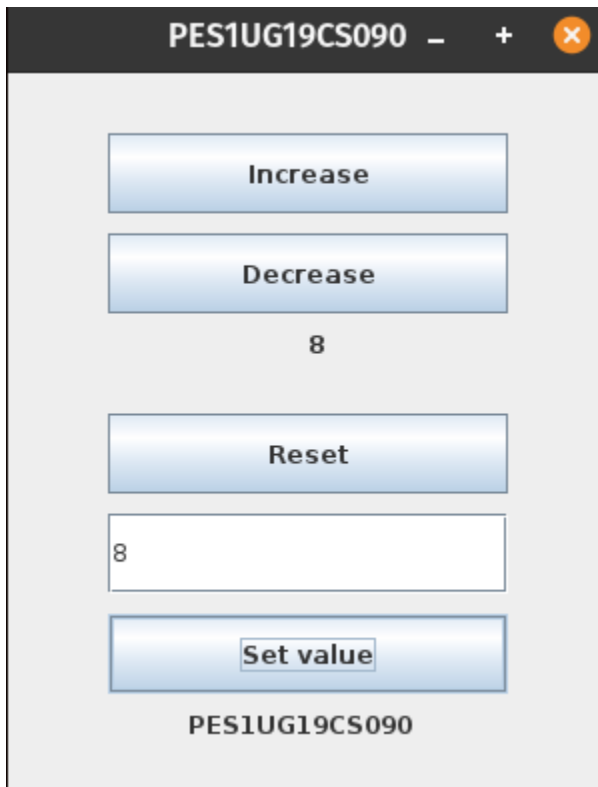
public JButton getResetButton() {
    return resetButton;
}

public JButton getSetButton() {
    return setButton;
}

public void setText(String text) {
    label.setText(text);
}
}
```

Step4. Run the application

When the user wants to transmit or receive data, the Controller responds by asking or sending the data from the Model. After that, the Controller sends the result (success or error) back to the View. View also operated via the Controller, inquiring about the data or adding it to the Model. Finally, the Controller validates the data for updating by the Model.



PES1UG19CS090

Increase

Decrease

8

Reset

8

Set value

PES1UG19CS090