ECS505U Software Engineering

Laboratory Exercise

Design Patterns Exercise: MVC and Observer Patterns

Version 1.2, November 2017

1. Background

For this assignment you must create a simple Java application that uses the Model View Controller (MVC) and Observer design patterns.

In a brief overview of the assignment you will be required to program most of the elements of the code yourself. Some help will be provided for some of the non-trivial methods and the driver class.

The purpose of learning MVC and Observer is to create better code structure. This will help you towards designing the game structure for your project.

2. Prerequisites

- You should be able to run NetBeans on an ITL machine under Windows, or Linux. NetBeans version 8 and up is installed on the ITL machines for campus students.
- You should be able to run Visual Paradigm under Windows. In ITL machines version 14.1 is installed.
- You should have gone through the Lecture slides of week 8 and completed the lab exercises 1 and 2.

3. Aims

The aims and objectives of this lab assignment are for you to:

- Understand the importance of a design pattern.
- Understand how to code MVC and Observer structures for any project.
- Learn how to link a View to a Model by using a Controller.
- Learn how to implement and use Observers.

4. Designing an Application Based On MVC

Create the following UML class diagram using Visual Paradigm. The diagram only uses MVC to make the design easier to understand and avoid confusion.

The elements required for the Observer pattern will be introduced during implementation.

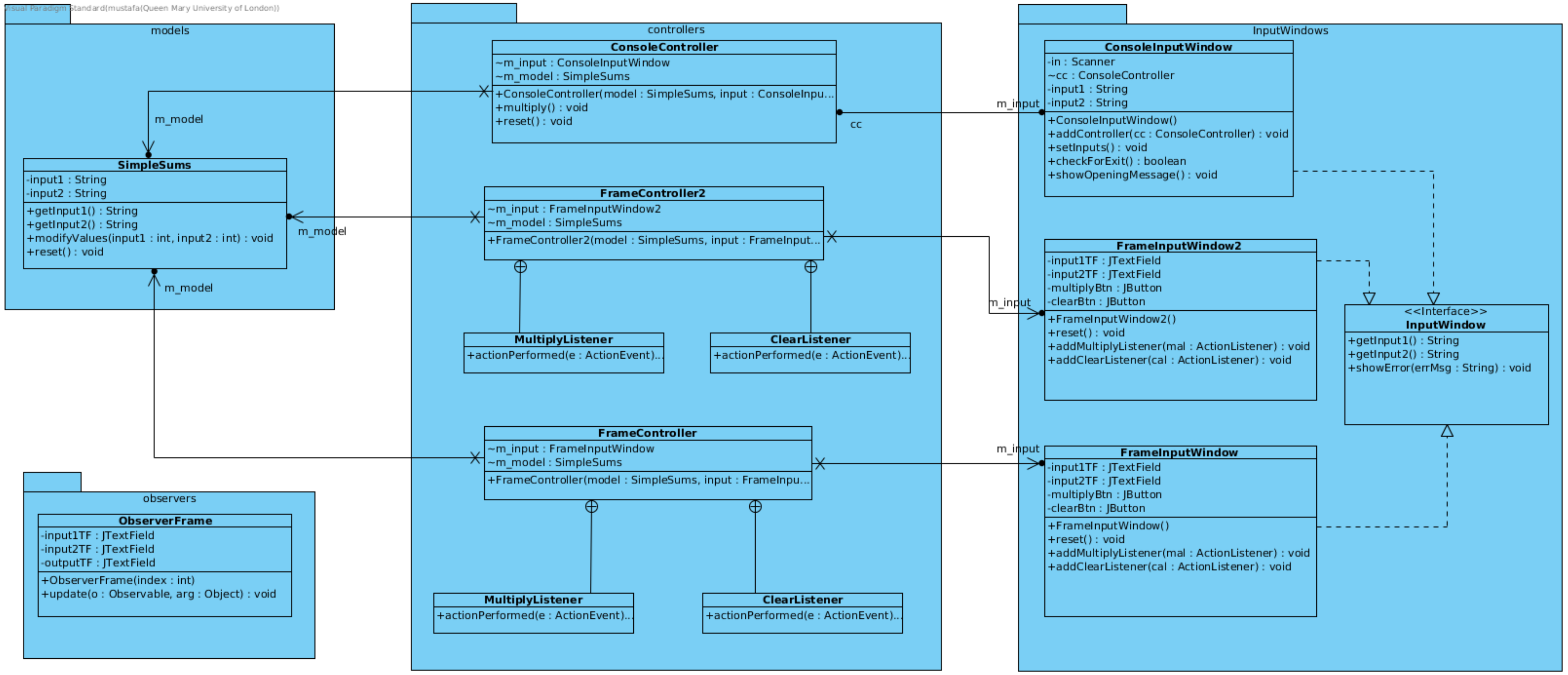
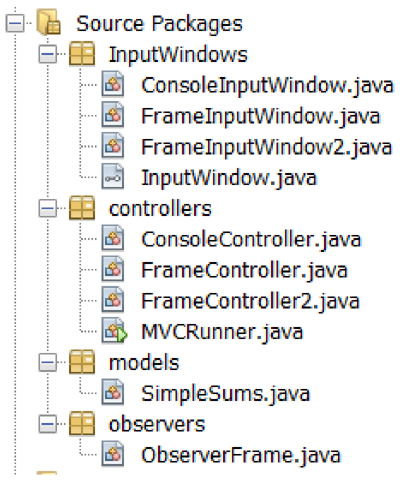


Figure 1: UML class diagram for the exercise

If you are having difficulty creating the given UML class diagram the VPP file is included in the QMplus. You can download the file and continue this exercise from the next page onwards.

5. Implementing the Observer Pattern

Generate the Java code for your class diagram. Create a new Java project in NetBeans and copy the generated code to the “src” directory under your project folder. You should have the project file structure in the picture below.



```
Source Packages
    InputWindows
        ConsoleInputWindow.java
        FrameInputWindow.java
        FrameInputWindow2.java
        InputWindow.java
    controllers
        ConsoleController.java
        FrameController.java
        FrameController2.java
        MVCRunner.java
    models
        SimpleSums.java
    observers
        ObserverFrame.java
```

Program the following attributes and operations of each of the classes listed below. A description of the instance variables, constructors and operations has been provided for each class in every package.

DO NOT FORGET TO ADD THE IMPORTS for JAVA CLASSES!

5.1. Models Package

SimpleSums class is the class with the data (observable and model) so in this case it needs to extend Observable class (included in JDK). Use the following code to complete the implementation of this class.

```
public void modifyValues(int input1, int input2) {
      this.input1 = input1;
      this.input2 = input2;
      result = input1 * input2;
      setChanged();
      notifyObservers();
   }

   public void reset() {
      input1 = 0;
      input2 = 0;
      result = 0;
      setChanged();
      notifyObservers();
   }
```

As you can see the multiply method includes two methods that we didn’t declare *setChanged* and *notifyObservers*. These methods are inherited from Observable class. NotifyObservers method updates all observers. You can read more about Observable class on Java docs website. (https://docs.oracle.com/javase/8/docs/api/java/util/Observable.html).

5.2. Observers Package

ObserverFrame class is the observer (also the view class in MVC). To make it an observer it needs to implement the Observer interface. Use the following code to complete the implementation of this class.

```
public ObserverFrame(int index) {
    input1TF = new JTextField(10);
    input1TF.setEditable(false);
    input2TF = new JTextField(10);
    input2TF.setEditable(false);
    outputTF = new JTextField(10);
    outputTF.setEditable(false);
    JPanel content = new JPanel();
    content.setLayout(new FlowLayout());
    content.add(new JLabel("Input1 value"));
    content.add(input1TF);
    content.add(new JLabel("Input2 value"));
    content.add(input2TF);
    content.add(new JLabel("Result"));
    content.add(outputTF);
    this.setContentPane(content);
    this.pack();
    this.setTitle("Observer Frame " + (index + 1));
    this.setLocation(0, index * 200);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.setVisible(true);
  }

public void update(Observable o, Object arg) {
    if (o instanceof SimpleSums) {
      SimpleSums s = (SimpleSums) o;
      int input1 = s.getInput1();
      int input2 = s.getInput2();
      if (input1 == 0 && input2 == 0) {
        input1TF.setText(String.valueOf(""));
        input2TF.setText(String.valueOf(""));
        outputTF.setText(String.valueOf(""));
      } else {
        input1TF.setText(String.valueOf(input1));
        input2TF.setText(String.valueOf(input2));
        outputTF.setText(String.valueOf(s.getResult()));
      }
    }
  }
```

The update method is declared in Observer interface and we need to overwrite it to update the view with the modified observable values.

5.3. Input Windows Package

This package contains three-different type of input windows. Each window implements InputWindow interface.

Add the following code to each input window to complete their implementation.

5.3.1. Console Input Window

```
public ConsoleInputWindow() {
      in = new Scanner(System.in);
      input1 = "";
      input2 = "";
   }

   public void addController(ConsoleController cc) {
      this.cc = cc;
   }

   public void setInputs() {
      System.out.print("Enter input 1:");
      input1 = in.nextLine();
      System.out.print("Enter input 2:");
      input2 = in.nextLine();
      cc.multiply();
   }

   public String getInput1() {
      return this.input1;
   }

   public String getInput2() {
      return this.input2;
   }

   public boolean checkForExit() {
      System.out.print("Multiply more numbers?(y/n/r(eset)/e(xit)) ");
      String userIN = in.nextLine();
      if (userIN.equalsIgnoreCase("y")) {
         return true;
      } else if (userIN.equalsIgnoreCase("e")) {
         System.exit(0);
      } else if (userIN.equalsIgnoreCase("r")) {
         cc.reset();
         return true;
      } else;
      return false;
   }

   public void showOpeningMessage() {
      System.out.println("A Simple MVC Example(Multiplication)");
      System.out.println("Awaiting input");
   }

   public void showError(String errMsg) {
      System.err.println(errMsg);
   }
```

5.3.2. Frame Input Window

```
public FrameInputWindow() {
      input1TF = new JTextField(10);
      input2TF = new JTextField(10);
      multiplyBtn = new JButton("Multiply");
      clearBtn = new JButton("Clear");
      JPanel content = new JPanel();
      content.setLayout(new FlowLayout());
      content.add(new JLabel("Input 1"));
      content.add(input1TF);
      content.add(new JLabel("Input 2"));
```

```
        content.add(input2TF);
        content.add(multiplyBtn);
        content.add(clearBtn);
        this.setContentPane(content);
        this.pack();
        this.setTitle("Simple MVC Example(Multiplication)");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public String getInput1() {
        return input1TF.getText();
    }

    public void reset(){
        input1TF.setText("");
        input2TF.setText("");
    }

    public String getInput2() {
        return input2TF.getText();
    }

    public void addMultiplyListener(ActionListener mal) {
        multiplyBtn.addActionListener(mal);
    }

    public void addClearListener(ActionListener cal) {
        clearBtn.addActionListener(cal);
    }

    public void showError(String errMsg) {
        JOptionPane.showMessageDialog(this, errMsg);
    }
```

5.3.3. Frame Input Window 2

```
public FrameInputWindow2() {
        Font f = new Font("Courier", Font.BOLD,14);
        input1TF = new JTextField(5);  input2TF = new JTextField(5);
        multiplyBtn = new JButton("Multiply");
        multiplyBtn.setFont(f);
        clearBtn = new JButton("Clear");
        clearBtn.setFont(f);
        JPanel content = new JPanel();
        content.setLayout(new FlowLayout());
        JLabel in1 = new JLabel("Input 1");
        in1.setFont(f);
        content.add(in1);
        content.add(input1TF);
        JLabel in2 = new JLabel("Input 2");
        in2.setFont(f);
        content.add(in2);
        content.add(input2TF);
        content.add(multiplyBtn);
        content.add(clearBtn);
        this.setContentPane(content);
        this.pack();
        this.setTitle("Alternative Frame for MVC Example(Multiplication)");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

        public String getInput1() {
        return input1TF.getText();
    }
    public String getInput2() {
      return input2TF.getText();
    }

    public void reset(){
      input1TF.setText("");
      input2TF.setText("");
    }

    public void addMultiplyListener(ActionListener mal) {
      multiplyBtn.addActionListener(mal);
    }

    public void addClearListener(ActionListener cal) {
      clearBtn.addActionListener(cal);
    }

    public void showError(String errMsg) {
     JOptionPane.showMessageDialog(this, errMsg);
    }
```

5.4. Controllers Package

This package contains three controllers for three-different type of input windows. Add the following code to each input window to complete their implementation.

5.4.1. Console Controller

```
public ConsoleController(SimpleSums model, ConsoleInputWindow input) {
        m_model = model;
        m_input = input;
         }

      public void multiply(){
        try {
          m_model.modifyValues(Integer.parseInt(m_input.getInput1()),
Integer.parseInt(m_input.getInput2()));
        } catch (NumberFormatException nfex) {
          m_input.showError("Bad input: '" + m_input.getInput1() + "'" + ", '" +
m_input.getInput2() + "'");
        }
      }

      public void reset(){
        m_model.reset();
      }
```

5.4.2. Frame Controller

```
public FrameController(SimpleSums model, FrameInputWindow input) {
      m_model = model;
      m_input = input;
//... Add listeners to the input.
      input.addMultiplyListener(new MultiplyListener());
      input.addClearListener(new ClearListener());
    }

    public class MultiplyListener implements ActionListener {

      @Override
      public void actionPerformed(ActionEvent e) {
          try {
            m_model.modifyValues(Integer.parseInt(m_input.getInput1()),
Integer.parseInt(m_input.getInput2()));
          } catch (NumberFormatException nfex) {
            m_input.showError("Bad input: '" + m_input.getInput1() + "'" + ", '" +
m_input.getInput2() + "'");
          }
      }
  }//end inner class MultiplyListener

  public class ClearListener implements ActionListener {
      @Override
      public void actionPerformed(ActionEvent e) {
          m_model.reset();
          m_input.reset();
      }
  }// end inner class ClearListener
```

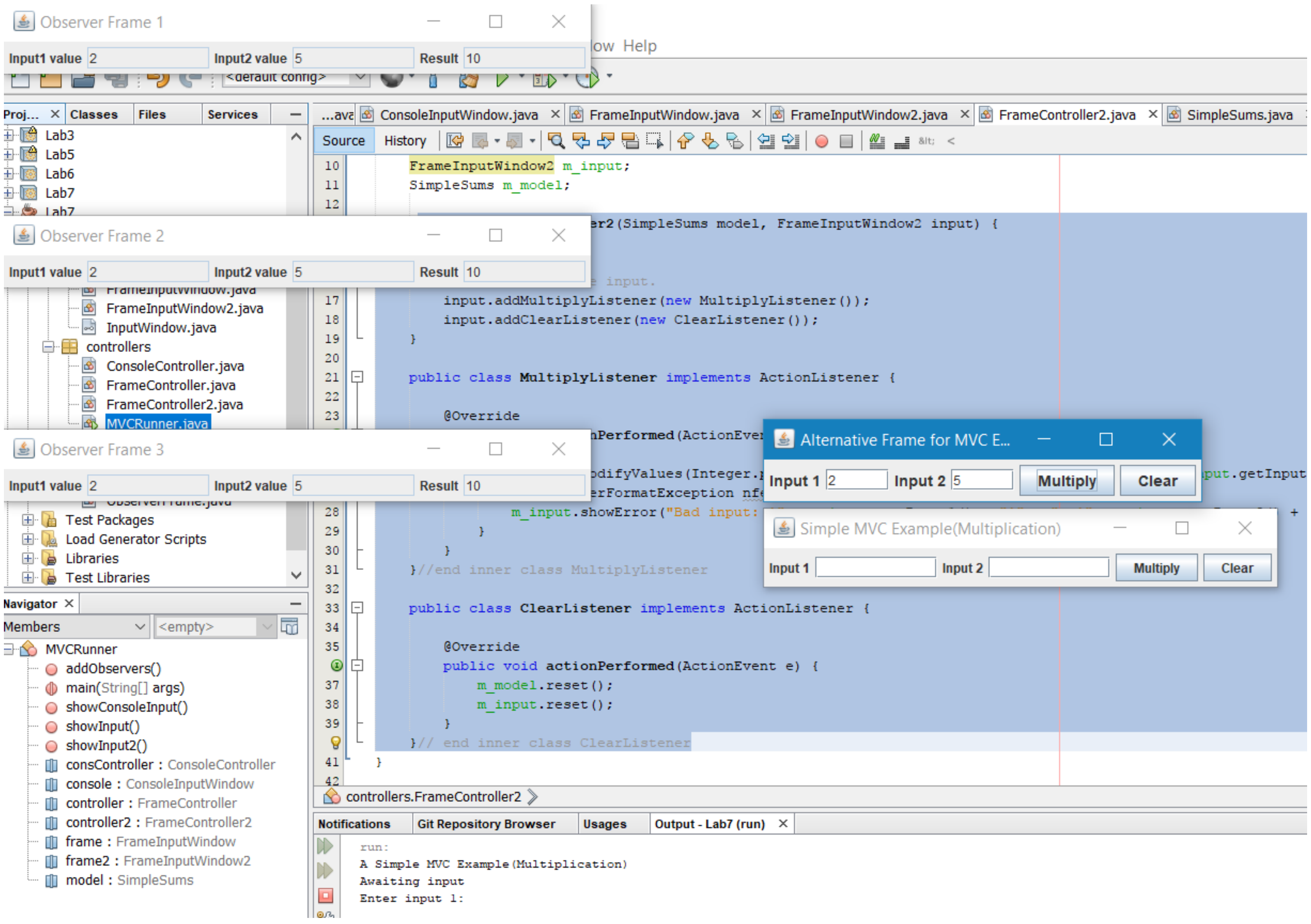
5.4.3. Frame2 Controller

```
public FrameController2(SimpleSums model, FrameInputWindow2 input) {
      m_model = model;
      m_input = input;
//... Add listeners to the input.
      input.addMultiplyListener(new MultiplyListener());
      input.addClearListener(new ClearListener());
  }

  public class MultiplyListener implements ActionListener {

      @Override
      public void actionPerformed(ActionEvent e) {
          try {
            m_model.modifyValues(Integer.parseInt(m_input.getInput1()),
Integer.parseInt(m_input.getInput2()));
          } catch (NumberFormatException nfex) {
            m_input.showError("Bad input: '" + m_input.getInput1() + "'" + ", '" +
m_input.getInput2() + "'");
          }
      }
  }//end inner class MultiplyListener

  public class ClearListener implements ActionListener {

      @Override
      public void actionPerformed(ActionEvent e) {
          m_model.reset();
          m_input.reset();
      }
  }// end inner class ClearListener
```

A Reminder: In an ideal MVC it’s expected that the controller updates both the model and view however, in this lab exercise we implemented our model to update view. This was done to accommodate standard observer design pattern. It’s possible to move update to controllers to separate model and view completely however this might be confusing to some students. To keep the exercise simple, I didn’t use that solution.

Add the MVC runner class (from Qmplus) and run your code with it to see if your implementation is correct. You should get 3 observer frames and 2 FrameControllers and a ConsoleController when you execute the code.

Take a screenshot of the running code and submit it.



Observer Frame 1 — Input1 value 2 | Input2 value 5 | Result 10

Observer Frame 2 — Input1 value 2 | Input2 value 5 | Result 10

Observer Frame 3 — Input1 value 2 | Input2 value 5 | Result 10

Alternative Frame for MVC E... — Input 1 2 | Input 2 5 | Multiply | Clear

Simple MVC Example(Multiplication) — Input 1 | Input 2 | Multiply | Clear

```
    FrameInputWindow2 m_input;
    SimpleSums m_model;

                 er2(SimpleSums model, FrameInputWindow2 input) {

                 e input.
        input.addMultiplyListener(new MultiplyListener());
        input.addClearListener(new ClearListener());
    }

    public class MultiplyListener implements ActionListener {

        @Override
                 nPerformed(ActionEve

                 odifyValues(Integer.                          put.getInput
                 erFormatException nf
                m_input.showError("Bad input:                              +
            }
        }
    }//end inner class MultiplyListener

    public class ClearListener implements ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) {
            m_model.reset();
            m_input.reset();
        }
    }// end inner class ClearListener
}
```

controllers.FrameController2

Output - Lab7 (run)

```
run:
A Simple MVC Example(Multiplication)
Awaiting input
Enter input 1:
```

Figure II: Submission screen shot