70094 17

Interactive Applications

Submitters

Mark: 19/20
Very nice implementation! You've implemented more calculator functions than expected. However, you may need to pay a little attention to code duplication in tests. Well done!

jh5723	James Hartley
sf23	Shihan Fu

Emarking

```
Final Tests: Summary for jh5723 of v5
-------------------------------------

  Public Tests:
    Compiles:      1 / 1
    Tests Pass:    1 / 1
    Style Checks:  1 / 1

Git Repo: git@gitlab.doc.ic.ac.uk:lab2324_spring/SE_Design_Ex7_jh5723.git
Commit ID: 61d0c
```

Great ✓

```java
package ic.doc;

import org.junit.Test;
import org.jmock.lib.concurrent.Synchroniser;
import org.jmock.Expectations;
import org.jmock.Mockery;
import static org.junit.Assert.fail;

public class RpnCalculatorModelTest {

  // Need synchroniser for mockery to ensure thread safety
  private Mockery context = new Mockery() {{
    setThreadingPolicy(new Synchroniser());
  }};
  private final Observer mockObserver = context.mock(Observer.class);

  @Test
  public void pushValueNotifiesObserver() {
    RpnCalculatorModel calculator = new RpnCalculatorModel();
    calculator.addObserver(mockObserver);

    double value = 5.0;

    context.checking(new Expectations() {{
      oneOf(mockObserver).update(value);
    }});

    calculator.pushValue(value);

    context.assertIsSatisfied();
  }

  @Test
  public void performOperationNotifiesObserverWithResult() {
    RpnCalculatorModel calculator = new RpnCalculatorModel();
    calculator.addObserver(mockObserver);

    final double value1 = 5.0;
    final double value2 = 10.0;
    final double expectedResult = 15.0;

    context.checking(new Expectations() {{
      exactly(1).of(mockObserver).update(5.0);
      exactly(1).of(mockObserver).update(10.0);
      exactly(1).of(mockObserver).update(expectedResult);
    }});

    calculator.pushValue(value1);
    calculator.pushValue(value2);
    calculator.performOperation("+");

    context.assertIsSatisfied();
  }

  @Test
  public void canClearStack() {
    RpnCalculatorModel calculator = new RpnCalculatorModel();
    calculator.addObserver(mockObserver);

    context.checking(new Expectations() {{
      exactly(1).of(mockObserver).update(5.0);
      exactly(1).of(mockObserver).update(10.0);
    }});

    calculator.pushValue(5.0);
    calculator.pushValue(10.0);
```

Good to use mock object to test in isolation

You could move the setup outside the methods to avoid code duplication


```java
    calculator.clearStack();
    try {
      calculator.performOperation("+");
      fail("Expected IllegalStateException to be thrown");
    } catch (IllegalStateException e) {
      System.out.println("Expected IllegalStateException was caught: "
          + e.getMessage());
    }

    context.assertIsSatisfied();
  }

  @Test
  public void addsTogetherTopTwoNumbersOnStack() {
    RpnCalculatorModel calculator = new RpnCalculatorModel();
    calculator.addObserver(mockObserver);

    context.checking(new Expectations() {{
      exactly(1).of(mockObserver).update(1.0);
      exactly(1).of(mockObserver).update(2.0);
      exactly(1).of(mockObserver).update(3.0);
    }});

    calculator.pushValue(1.0);
    calculator.pushValue(2.0);
    calculator.performOperation("+");

    context.assertIsSatisfied();
  }

  @Test
  public void multipliesTogetherTopTwoNumbersOnStack() {
    RpnCalculatorModel calculator = new RpnCalculatorModel();
    calculator.addObserver(mockObserver);

    context.checking(new Expectations() {{
      exactly(1).of(mockObserver).update(4.0);
      exactly(1).of(mockObserver).update(2.0);
      exactly(1).of(mockObserver).update(8.0);
    }});

    calculator.pushValue(4.0);
    calculator.pushValue(2.0);
    calculator.performOperation("*");

    context.assertIsSatisfied();
  }

  @Test
  public void subtractsTopNumberOnStackFromOneBeneath() {
    RpnCalculatorModel calculator = new RpnCalculatorModel();
    calculator.addObserver(mockObserver);

    context.checking(new Expectations() {{
      exactly(1).of(mockObserver).update(5.0);
      exactly(1).of(mockObserver).update(2.0);
      exactly(1).of(mockObserver).update(3.0);
    }});

    calculator.pushValue(5.0);
    calculator.pushValue(2.0);
    calculator.performOperation("-");

    context.assertIsSatisfied();
  }

```

```
  @Test
  public void dividesSecondNumberOnStackByFirst() {
    RpnCalculatorModel calculator = new RpnCalculatorModel();
    calculator.addObserver(mockObserver);

    context.checking(new Expectations() {{
      exactly(1).of(mockObserver).update(6.0);
      exactly(1).of(mockObserver).update(2.0);
      exactly(1).of(mockObserver).update(3.0);
    }});

    calculator.pushValue(6.0);
    calculator.pushValue(2.0);
    calculator.performOperation("/");

    context.assertIsSatisfied();
  }
}
```



```
// NOTE TO EXAMINER -- We have commented out this test as, even with
// java.awt.headless set to true, LabTS throws an exception

//package ic.doc;
//
//import org.junit.Test;
//import static org.junit.Assert.assertEquals;
//import org.junit.BeforeClass;
//
//public class RpnCalculatorGuiTest {
//
//  @BeforeClass
//  public static void setUp() {
//    System.setProperty("java.awt.headless", "true");
//  }
//
//  @Test
//  public void updateUpdatesDisplay() {
//    RpnCalculatorModel model = new RpnCalculatorModel();
//    RpnCalculatorGui gui = new RpnCalculatorGui(model) {
//
//      // Directly verify update
//      @Override
//      public void update(double result) {
//        super.update(result);
//        assertEquals("Expected display text", String.valueOf(result), this.display.getText());
//      }
//    };
//
//    gui.update(5.0);
//  }
//}

```

```java
/**
 * Represents the model component of a Reverse Polish Notation (RPN) calculator.
 * This class manages the arithmetic operations using a stack to store numerical
 * values. It implements the Observable interface to notify observers about
 * changes in the calculation result. Observers are notified whenever a new
 * value is pushed onto the stack or after the top of the stack is changed.
 */

package ic.doc;

import java.util.ArrayList;
import java.util.List;
import java.util.Stack;

public class RpnCalculatorModel implements Observable {
  private Stack<Double> stack = new Stack<>();
  private List<Observer> observers = new ArrayList<>();

  // Function to push displayed number to the stack
  public void pushValue(double value) {
    stack.push(value);
    notifyObservers();
  }

  // Function defining behaviour of operation buttons
  public void performOperation(String operation) {

    if (stack.size() == 0) {
      throw new IllegalStateException("Error: Insufficient operands");
    }
    if (stack.size() < 2) {
      return; // Allows operation on
    }
    double b = stack.pop();
    double a = stack.pop();
    double result = 0;
    switch (operation) {
      case "+":
        result = a + b;
        break;
      case "-":
        result = a - b;
        break;
      case "*":
        result = a * b;
        break;
      case "/":
        result = a / b;
        break;
      default:
        throw new IllegalArgumentException("Unsupported operation: "
            + operation);
    };
    stack.push(result);
    notifyObservers();
  }

  public void clearStack() {
    stack.clear();
  }

  @Override
  public void addObserver(Observer observer) {
    observers.add(observer);
  }

```

Good to set a list of observers instead of one single observer

Great!


```java
  @Override
  public void removeObserver(Observer observer) {
    observers.remove(observer);
  }

  @Override
  public void notifyObservers() {
    double result = stack.isEmpty() ? 0 : stack.peek();
    for (Observer observer : observers) {
      observer.update(result);
    }
  }
}
```

```java
/**
 * Represents the Graphical User Interface (GUI) for a Reverse Polish Notation
 * calculator, with a visual interface for users to input numbers and
 * operations. It subscribes to the RpnCalculatorModel as an observer,
 * sending values and operations to the model and updating the display in
 * response to changes in the model's state.
 */

package ic.doc;

// Swing classes implemented
import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.JTextArea;
import javax.swing.JPanel;

// awt classes implemented
import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;


public class RpnCalculatorGui implements Observer {
  JTextArea display;   It'd be better to write the access modifier explicitly
  private RpnCalculatorModel rpnCalculator;
  private JFrame frame;

  public RpnCalculatorGui(RpnCalculatorModel model) {
    this.rpnCalculator = model;
    this.rpnCalculator.addObserver(this); // Observer to the RpnCalculatorModel
    frame = new JFrame("Reverse Polish Notation Calculator");
    frame.setSize(300, 200);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    initializeUi();
  }

  // Function to create the buttons and display and add ActionListeners
  private void initializeUi() {
    frame.setLayout(new BorderLayout());
    display = new JTextArea();
    frame.add(display, BorderLayout.NORTH);

    JPanel buttonPanel = new JPanel();
    buttonPanel.setLayout(new GridLayout(4, 4));
    frame.add(buttonPanel, BorderLayout.CENTER);

    String[] buttons = {"1", "2", "3", "+",
        "4", "5", "6", "-",
        "7", "8", "9", "*",
        "0", "Enter", "C", "/"};
    for (String buttonText : buttons) {
      JButton button = new JButton(buttonText);
      button.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
          buttonPressed(e);
        }
      });
      buttonPanel.add(button);
    }
  }

  // Function to communicate keystrokes to the model
  private void buttonPressed(ActionEvent e) {
    String command = e.getActionCommand();
```



```java
    if ("Enter".equals(command)) {
      try {
        double value = Double.parseDouble(display.getText());
        rpnCalculator.pushValue(value);
        display.setText("");                                    Great!
      } catch (NumberFormatException ex) {
        display.setText("Error");
      }
    } else if ("C".equals(command)) {
      // "C" clears the stack and display
      rpnCalculator.clearStack();
      display.setText("");
    } else if ("+".equals(command) || "-".equals(command)
        || "*".equals(command) || "/".equals(command)) {
      // Attempts to apply operation to most recent pair of values in the stack
      rpnCalculator.performOperation(command);
    } else {
      // Append digits to the display
      display.append(command);
    }
  }

  @Override
  public void update(double result) {
    display.setText(String.valueOf(result));
  }

  public void show() {
    frame.setVisible(true);
  }
}
```

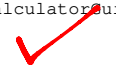
```java
/**
 * Observer interface for receiving updates about changes in a subject's state.
 * Implementations of this interface should define how to handle the updated
 * result. The update method is called with the latest result value whenever
 * the subject's state changes.
 */

package ic.doc;

public interface Observer {
  void update(double result);
}
```

Great!


```java
/**
 * Observable interface for objects that can be observed by Observers.
 * This interface declares methods for adding and removing Observers,
 * as well as notifying them of any state changes. Implementing classes
 * should maintain a list of Observers and call their update method
 * when there are relevant changes to notify them about.
 */

package ic.doc;

public interface Observable {
  void addObserver(Observer observer);

  void removeObserver(Observer observer);

  void notifyObservers();
}
```

Good design

```
 1: package ic.doc;
 2:
 3: import javax.swing.SwingUtilities;
 4:
 5: public class Main {
 6:   public static void main(String[] args) {
 7:     // Place GUI initialization code inside a Runnable to ensure thread safety
 8:     SwingUtilities.invokeLater(() -> {
 9:       RpnCalculatorModel model = new RpnCalculatorModel();
10:       RpnCalculatorGui calculatorGui = new RpnCalculatorGui(model);
11:       calculatorGui.show();
12:     });
13:   }
14: }
```



```
-------- Test Output --------
Running LabTS build... (Thu 29 Feb 14:51:08 UTC 2024)

Submission summary...
You made 6 commits
  - 274b316 Feat: adds combinations of 1 and 2 in Reverse Polish Notation [8 files changed, 277 insertions, 4 deletions]
  - 0bcbc12 Feat: can clear stack [3 files changed, 50 insertions, 15 deletions]
  - 50e9111 Feat: multiplies, divides, subtracts and adds integers inputted through integer keys 0-9 [3 files changed, 100 insertions, 27 deletions]
  - efbf0fc Fix: sets aws property to headless in Gui tests to pass on LabTS [1 file changed, 6 insertions, 1 deletion]
  - 7e51217 Fix: Deleted GUI updateDisplay test as cannot be passed on LabTS, even with headless set to true [1 file changed, 29 deletions]
  - 61d0c53 Refactor: created display helper functions and extracted GUI main function to Main.java [4 files changed, 50 insertions, 9 deletions]

Preparing...

BUILD SUCCESSFUL in 649ms

Compiling...
BUILD SUCCESSFUL in 4s

Running tests...

ic.doc.RpnCalculatorModelTest > multipliesTogetherTopTwoNumbersOnStack PASSED

ic.doc.RpnCalculatorModelTest > dividesSecondNumberOnStackByFirst PASSED

ic.doc.RpnCalculatorModelTest > subtractsTopNumberOnStackFromOneBeneath PASSED

ic.doc.RpnCalculatorModelTest > addsTogetherTopTwoNumbersOnStack PASSED

ic.doc.RpnCalculatorModelTest > performOperationNotifiesObserverWithResult PASSED

ic.doc.RpnCalculatorModelTest > canClearStack PASSED

ic.doc.RpnCalculatorModelTest > pushValueNotifiesObserver PASSED

BUILD SUCCESSFUL in 1s

Checking code style...
BUILD SUCCESSFUL in 5s
Finished auto test. (Thu 29 Feb 14:51:30 UTC 2024)

-------- Test Errors --------
```

Clear commit messages