JUnit-Testing GUI Components

Agenda

- Test GUI Components
  - Simple GUI Application
  - Test Cases Design.
  - Test Cases Implementation with JUnit.
  - Test Cases Execution.
- Project Part 3

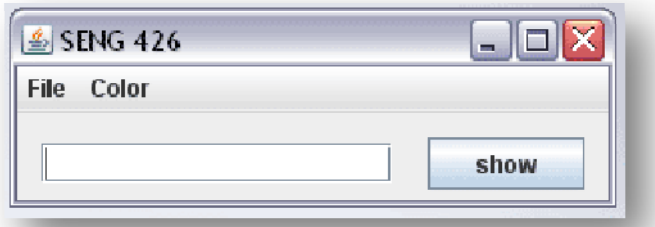
Simple GUI Application

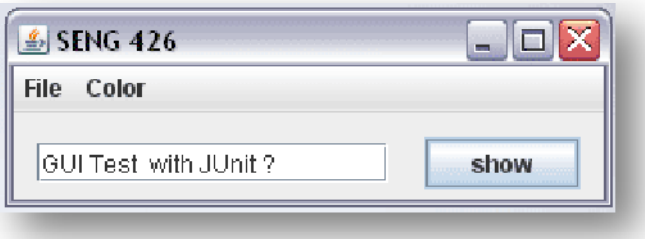
- In our application we will cover testing different GUI components, such as
  - JTextField
  - JButton
  - JDialog.
  - JMenu.

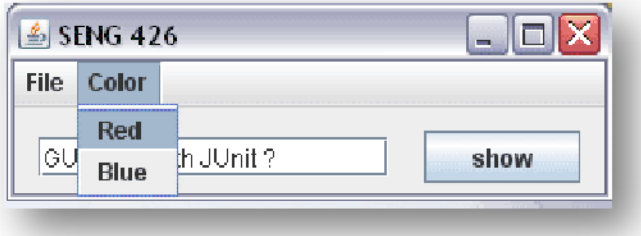
Application Overview

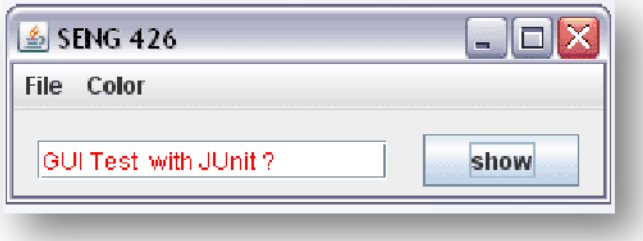
- The application consist of one java class, named ***MainFrame.java***
- The application has a text field. When a string is typed, it adds **?** to the end.
- When the show button is clicked, a dialog box displays the text + (...It works!).
- The application also has a menu for changing the text color.

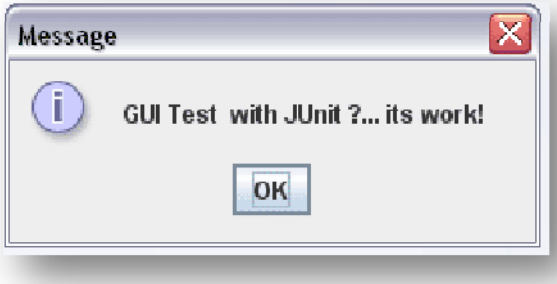
Application GUI











MainFrame.java

```
1   package Test;
2   import java.awt.Color;
3   import javax.swing.JOptionPane;
4   public class MainFrame extends javax.swing.JFrame {
5       /**...*/
6       public MainFrame() {...}
11      /**...*/
16      @SuppressWarnings("unchecked")
17      Generated Code
98
99      private void showButtonActionPerformed(java.awt.event.ActionEvent evt) {...}
104
105     private void inputTextFieldActionPerformed(java.awt.event.ActionEvent evt) {...}
109
110     private void redColorMenuActionPerformed(java.awt.event.ActionEvent evt) {...}
114
115     private void blueColorMenuActionPerformed(java.awt.event.ActionEvent evt) {...}
119
120     /**...*/
123     public static void main(String args[]) {...}
130
131     // Variables declaration - do not modify
132     private javax.swing.JMenuItem blueColorMenu;
133     private javax.swing.JMenu colorMenu;
134     private javax.swing.JMenuItem exitFileMenu;
135     private javax.swing.JMenu fileMenu;
136     private javax.swing.JTextField inputTextField;
137     private javax.swing.JMenuBar jMenuBar1;
138     private javax.swing.JMenuItem redColorMenu;
139     private javax.swing.JButton showButton;
140     // End of variables declaration
141 }
142
```

GUI Functions

```
    private void showButtonActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        JTextArea text = new JTextArea(inputTextField.getText()+"... its work!");
        text.setColumns(20);
        text.setLineWrap(true);
        text.setBackground(null);
        text.setEditable(false);
        JOptionPane.showMessageDialog(this,text );
    }

    private void inputTextFieldActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        inputTextField.setText(inputTextField.getText()+"?");
    }

    private void redColorMenuActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        inputTextField.setForeground(Color.RED);
    }

    private void blueColorMenuActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        inputTextField.setForeground(Color.BLUE);
    }
```

Prepare for Testing

- Our Test Cases should cover:
  - JTextField.
  - JDialog Box
  - JButton.
  - JMenu.
- Create test class for MainFrame using JUnit plugin in Netbeans.

MainFrameTest.java

```
1   package Test;
2   import org.junit.After;
3   import org.junit.AfterClass;
4   import org.junit.Before;
5   import org.junit.BeforeClass;
6   import org.junit.Test;
7   import static org.junit.Assert.*;
8
9   public class MainFrameTest {
10
11      public MainFrameTest() {
12      }
13      @BeforeClass
14      public static void setUpClass() throws Exception {...}
16      @AfterClass
17      public static void tearDownClass() throws Exception {...}
19      @Before
20      public void setUp() {...}
22      @After
23      public void tearDown() {...}
25
26      /**
27       * Test of main method, of class MainFrame.
28       */
29      @Test
30      public void testMain() {
31          System.out.println("main");
32          String[] args = null;
33          MainFrame.main(args);
34          // TODO review the generated test code and remove the default call to fail.
35          fail("The test case is a prototype.");
36      }
37  }
```

Problems

- JUnit can not generate test functions to GUI components.
- GUI functions are private so there is no direct path to access them.

Solution?

- There are many ways to access Swing components:
  1. Application code has getXxx() methods to return each component of interest.
  2. Test code invokes events on a screen, mimicking a human operator. Events are typically mouse moves/clicks and key typing.
  3. Test code traverses the component tree and finds a component of a specific signature (class, location, order, text contents, etc.).

Traverses GUI

- To allows the test code to traverses the GUI component tree.
  - Name each component that your test code will request access to it using setName() method.
  - Write the appropriated code to traverse the GUI components and provide an access to these components

Naming the GUI Components

- We need to add the following function to the MainFrame.java class
- We call this function from the class construction.

```
private void setComponentsNames(){
    blueColorMenu.setName("blue");
    redColorMenu.setName("red");
    inputTextField.setName("input");
    showButton.setName("show");
}
```

Create Traverse Class

- Component traversal code is encapsulated into a utility class, TestUtils.
- The ***TestUtils*** class contains the following static methods:
  - getChildNamed()
  - getChildIndexed()
  - getChildIndexedInternal()

TestUtils.java

```
1   package Test;
2
3   /*
4    * TestUtils
5    */
6   import java.awt.*;
7   import javax.swing.*;
8
9   public class TestUtils {
10
11      static int counter;
12
13      public static Component getChildNamed(Component parent, String name) {...}
30
31      public static Component getChildIndexed(Component parent, String name, int index) {...}
51
52      private static Component getChildIndexedInternal(Component parent, String name, int index) {...}
73  }
```

getChildNamed ( )

```
public static Component getChildNamed(Component parent, String name) {

        if (name.equals(parent.getName())) { return parent; }

        if (parent instanceof Container) {
                Component[] children = (parent instanceof JMenu) ?
                                ((JMenu)parent).getMenuComponents() :
                                ((Container)parent).getComponents();

                for (int i = 0; i < children.length; ++i) {
                        Component child = getChildNamed(children[i], name);
                        if (child != null) { return child; }
                }
        }

        return null;
}
```

getChildIndexed()

```
31
32      public static Component getChildIndexed(Component parent, String name, int index) {
33          counter = 0;
34          // Step in only owned windows and ignore its components in JFrame
35          if (parent instanceof Window) {
36              Component[] children = ((Window)parent).getOwnedWindows();
37
38                  for (int i = 0; i < children.length; ++i) {
39                      // take only active windows
40                      if (children[i] instanceof Window &&
41                              !((Window)children[i]).isActive()) { continue; }
42                      Component child = getChildIndexedInternal(
43                              children[i], name, index);
44                      if (child != null) { return child; }
45                  }
46          }
47          return null;
48      }
```

getChildIndexedInternal()

```
49
50      private static Component getChildIndexedInternal(Component parent, String name, int index) {
51
52          if (parent.getClass().toString().endsWith(name)) {
53              if (counter == index) { return parent; }
54              ++counter;
55          }
56
57          if (parent instanceof Container) {
58              Component[] children = (parent instanceof JMenu) ?
59                  ((JMenu)parent).getMenuComponents() :
60                  ((Container)parent).getComponents();
61
62                          for (int i = 0; i < children.length; ++i) {
63                              Component child = getChildIndexedInternal(
64                                      children[i], name, index);
65                              if (child != null) { return child; }
66                          }
67          }
68          return null;
69
70      }
71  }
72
```

Design Test Cases with JUint

- For each GUI component define a new test method in MainFrameTest.java class with the following signature:
  - public void ***testYourGUICompnentName()***
- Define appropriate variables to implement your test scenario.
- Use the ***TestUtils*** class to obtains access to the GUI components
- Use reference to control your GUI components.
- Use you GUI component reference and swings/awt APIs to change your GUI component behaviors.
- Perform action to trigger the action listener of your component using ***postActionEvent()*** method.

Testing the JTextFiled

- Define your test method

```
//This method test the JTextField Component in MainFrame.java class
@Test
public void testInputJTextField(){...}
```

- Define variables for your test Cases

```
//This method test the JTextField Component in MainFrame.java class
@Test
public void testInputJTextField(){
    //Define appropriate variables to implement your test scenario
    MainFrame frame;
    JTextField inputTest;
    String expResult;
```

Testing the JTextField *continue1*

- Begin your testing scenario

```
// start simulating your testing scenario
frame = new MainFrame();
frame.setVisible(true);
```

- Use the traversal code to access the GUI component

```
// use the inputTest as a reference to your JTextField component
inputTest = (JTextField)TestUtils.getChildNamed(frame, "input");

//check that you actually obtain an access to your target component
assertNotNull("Cann't access the JTextField component", inputTest);
```

Testing the JTextField *continue2*

- Use the reference of the GUI component to modify it.

```
//add some text to the JTextField.
inputTest.setText("Testing");
sleep(2000);
```

This method for automation purpose only you don't have to use it

- Post an action to the GUI component

```
//perform action to trigger the action listener of your component.
inputTest.postActionEvent();
sleep(2000);
```

This method for automation purpose only you don't have to use it

Testing the JTextField *continue3*

- Verify your test case

```
//define and test your expected result.
expResult = "Testing?";
assertEquals(expResult,inputTest.getText());
```

testInputJTextField()

```
48
49      //This method test the JTextField Component in MainFrame.java class
50      @Test
51      public void testInputJTextField(){
52          //Define appropriate variables to implement your test scenario
53          MainFrame frame;
54          JTextField inputTest;
55          String expResult;
56
57          // start simulating your testing scenario
58          frame = new MainFrame();
59          frame.setVisible(true);
60
61          // use the inputTest as a reference to your JTextField component
62          inputTest = (JTextField)TestUtils.getChildNamed(frame, "input");
63
64          //check that you actually obtain an access to your target component
65          assertNotNull("Cann't access the JTextField component", inputTest);
66
67          //add some text to the JTextField.
68          inputTest.setText("Testing");
69          sleep(2000);
70
71          //perform action to trigger the action listener of your component.
72          inputTest.postActionEvent();
73          sleep(2000);
74
75          //define and test your expected result.
76          expResult = "Testing?";
77          assertEquals(expResult,inputTest.getText());
78
79      }
80
```

Execute the Test Cases

- You can execute your test cases by:
  1. Right click on test Suite class and select Run File.
  2. Right click on the test class and select Run File.
  3. Right click on the Project Name and select Test.
  4. Press Alt+F6 or Shift+F6

Project Part 3:Test Preparation

- Requirements
  - Test Plan.
  - Test Design.
  - Test Implementation.