

# Assignment 5 – Graphical User Interface Testing

## Assignment

Name: Sai Vamsi Krishna Yedlapati

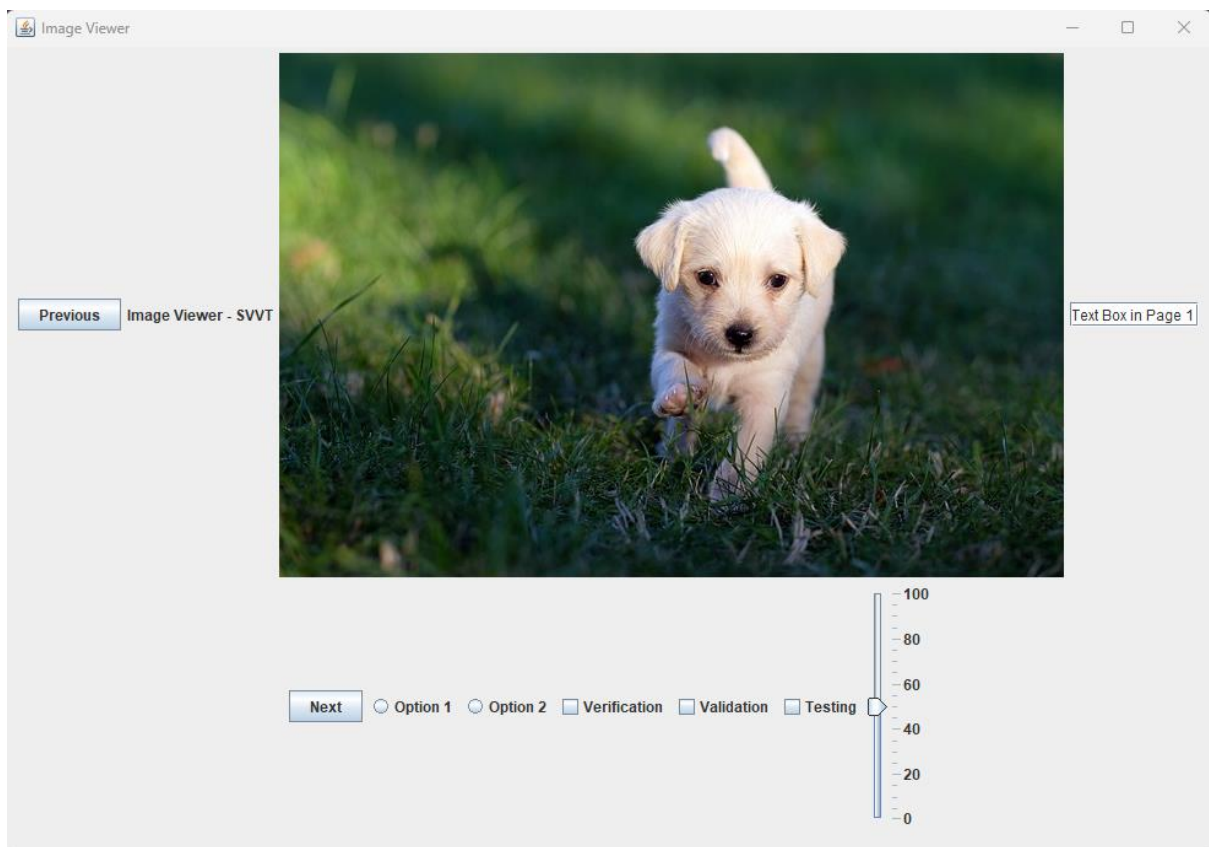
ASU ID: 1231515755

### Development of the two GUIs of an application as per the requirements:

Two versions of a basic GUI containing 3 pages are developed ensuring all the requirements mentioned are covered. Both the GUIs are developed in Java making use of Java Swing APIs.

#### Version 1:

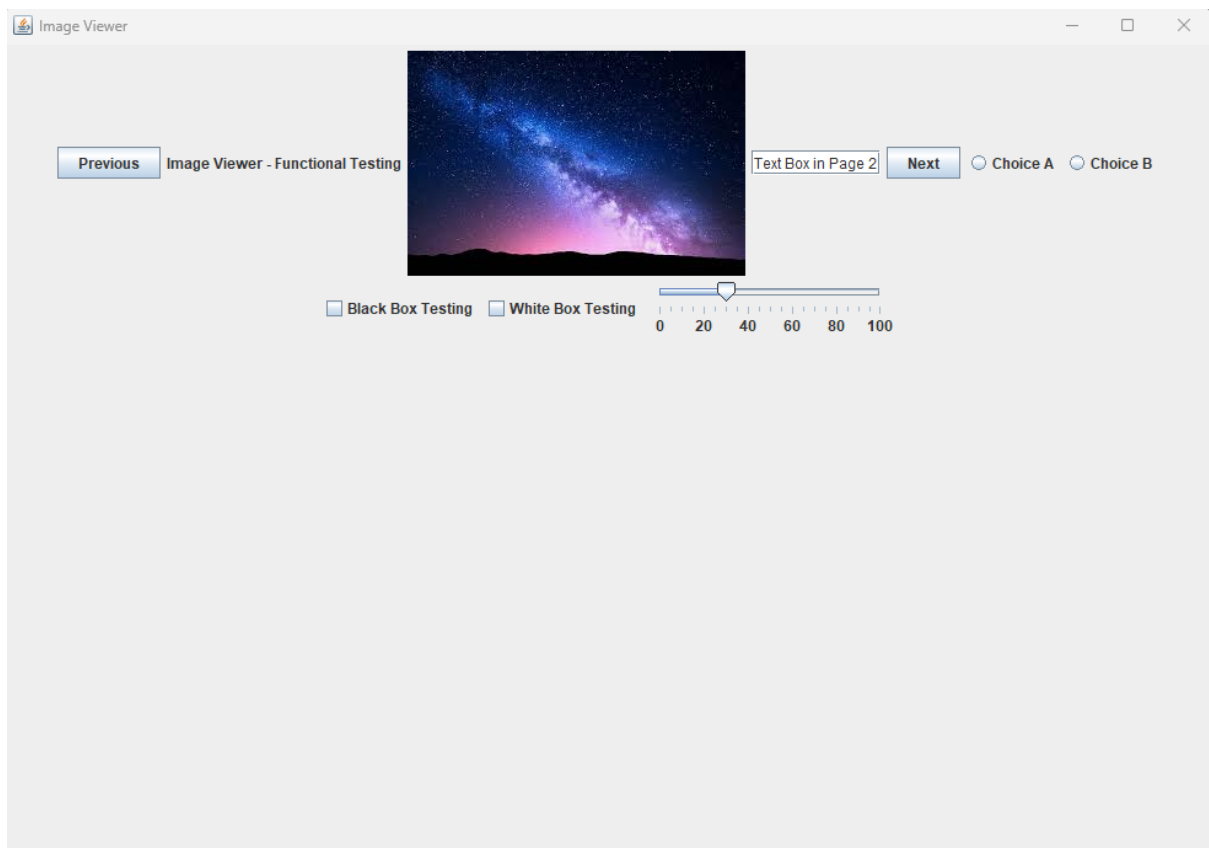
Page 1:



Page 1 contains the following:

1. Buttons to navigate to Previous Page and Next Page
2. Label mentioning "Image Viewer – SVVT"
3. Image of a Dog
4. A Text Box with some default data
5. Two radio button options
6. 3 Check Box options saying Verification, Validation and Testing
7. A Slider from 0 to 100 on the vertical axis.

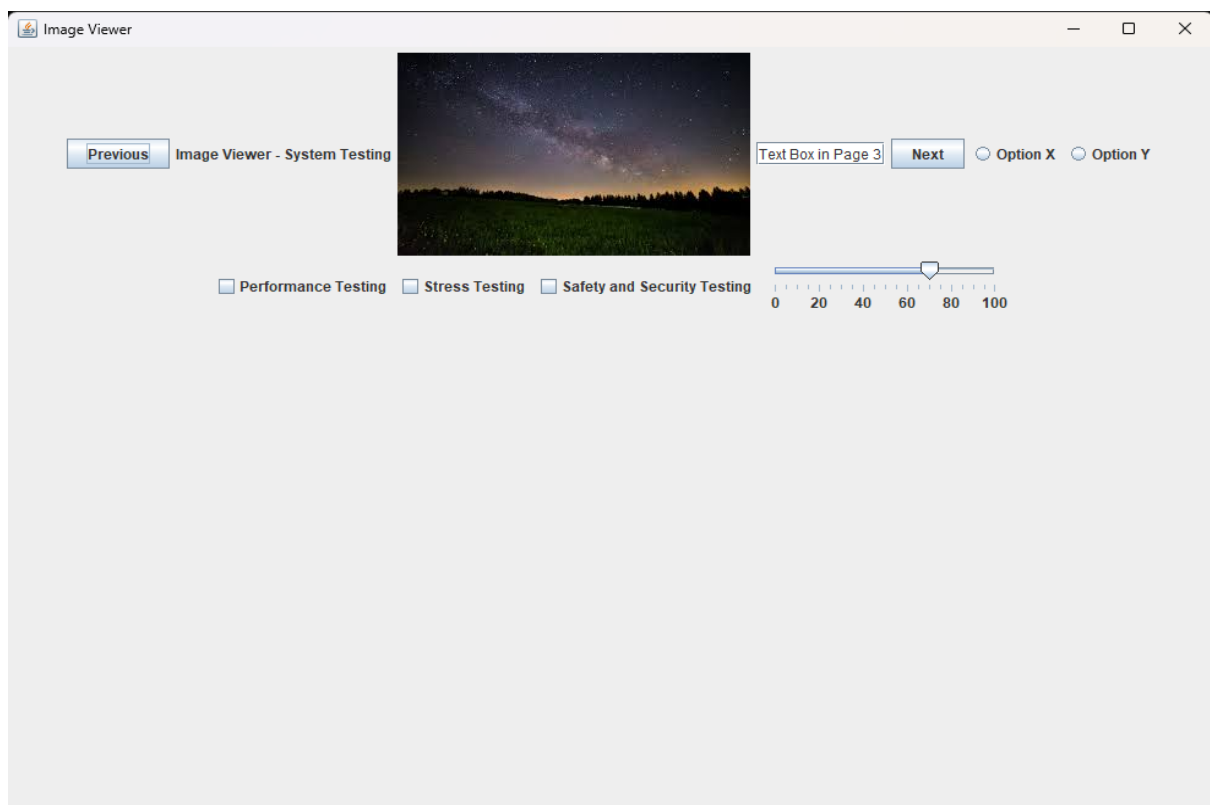
Page 2:



Page 2 contains the following:

1. Buttons to navigate to Previous Page and Next Page
2. Label mentioning "Image Viewer – Functional Testing"
3. Image of Milky way Galaxy
4. A Text Box with some default data
5. Two radio button options
6. 2 Check Box options Black-Box Testing and White Box Testing
7. A Slider from 0 to 100 on the horizontal axis.

### Page 3:



### Page 3 contents:

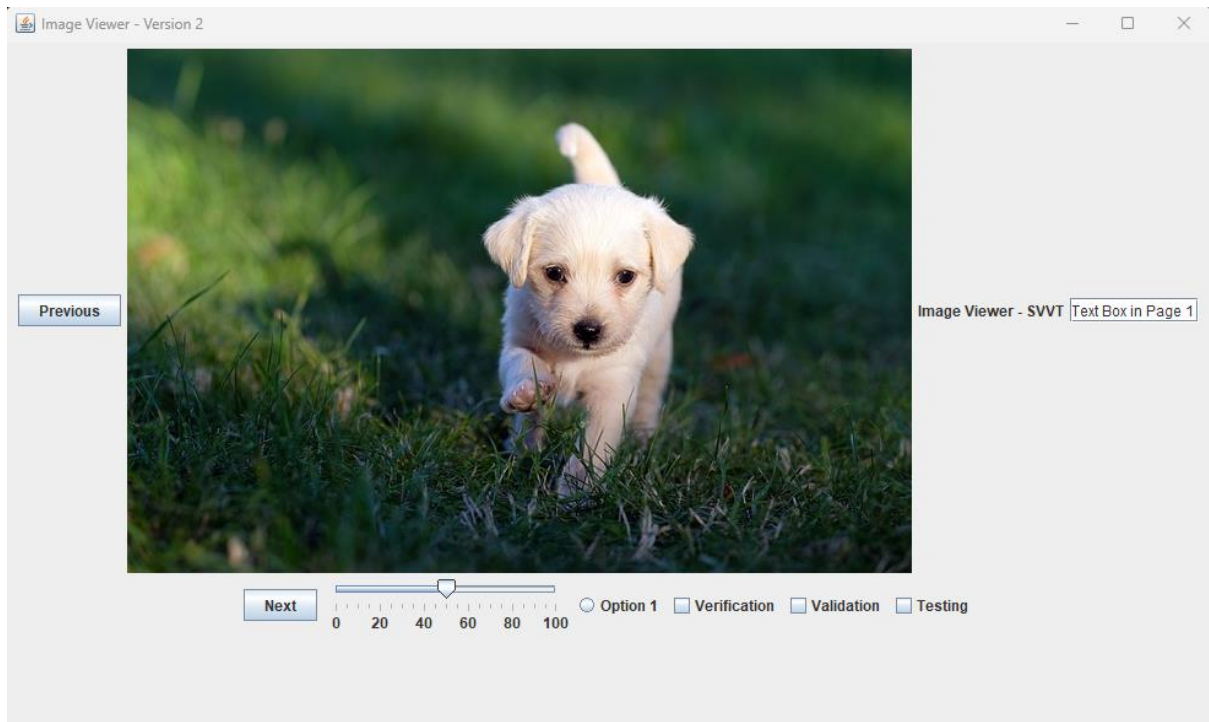
1. Buttons to navigate to Previous Page and Next Page
2. Label mentioning "Image Viewer – System Testing"
3. Image of a Night Sky
4. A Text Box with some default data
5. Two radio button options
6. 3 Check Box options Performance, Stress, Safety and Security Testing
7. A Slider from 0 to 100 on the horizontal axis.

The flow of pages is Page 1 → Page 2 → Page 3 → Page 1.

All the 3 pages have a similar pattern in terms of the placement/linearity of each of the options i.e., Previous button is followed by the Label which is followed by the Image etc. This is a very basic GUI model developed using Java Swing and tasking advantage of the various options provided by it. The GUI elements were added as per the requirements. Each page contains at least 1 button, 1 image, 1 label and 1 text box. Also, from the other GUI elements suggested, all the pages include radio buttons, checkboxes and sliders.

## Version 2:

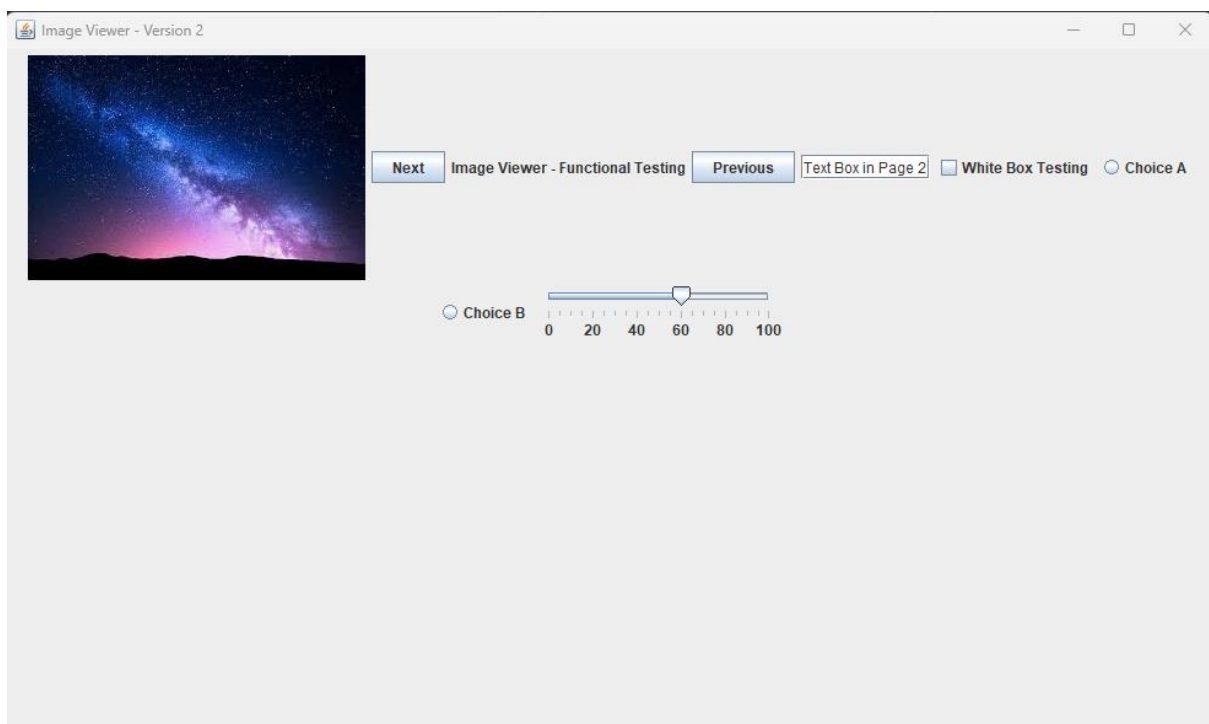
Page 1:



In Version 2, there are a total of 3 modifications in the Page 1:

1. The image and the label positions are switched
2. The Slider is made horizontal.
3. 1 Radio Button option is removed.

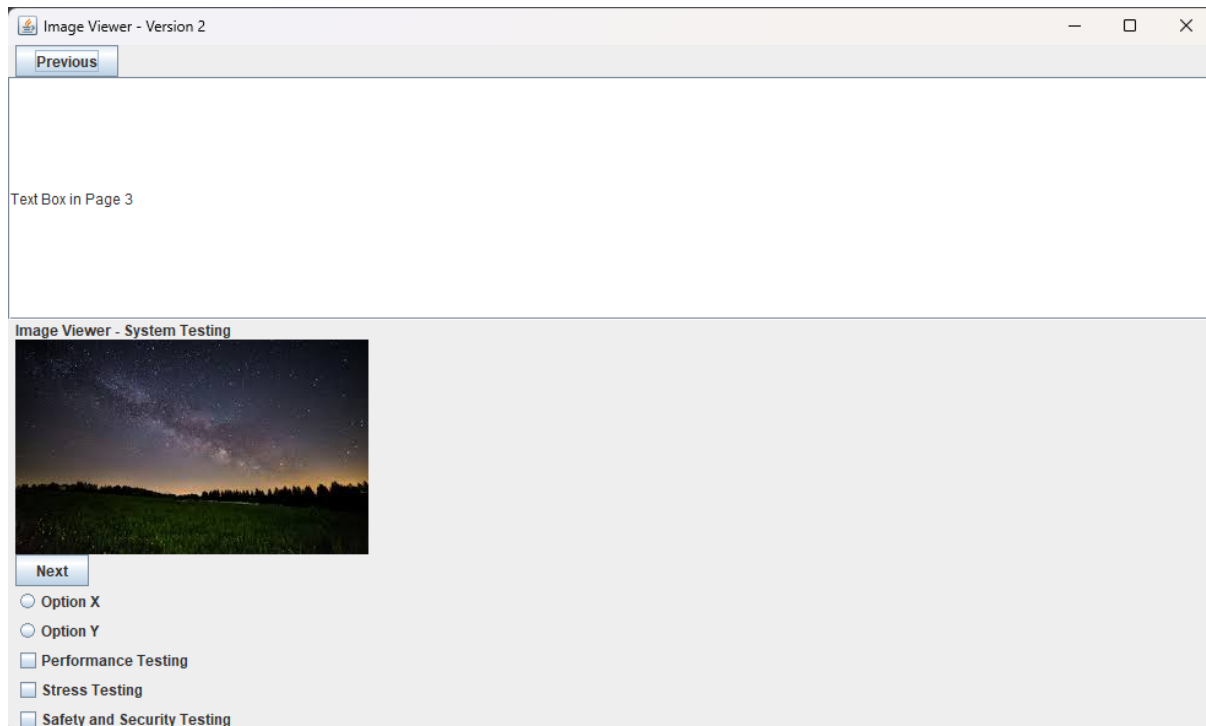
Page 2:



In Page2, following changes are made:

1. Image, Previous and Next Buttons and few other GUI elements positions are changed.
2. The default value of the slider is changed from 30 to 60.
3. One of the check-boxes is removed.

Page 3:



In Page 3, following changes are made:

1. GUI elements positions are modified in this page.
2. The size of the test box is modified.
3. The Slider is removed in this Page.

Apart from these changes, the flow of pages is modified in this Version 2. The flow is:

Page 1 → Page 3 → Page 2 → Page 1.

## Description of the tool selected for GUI testing:

The tool selected for GUI testing is IntelliJ IDE. It is popular IDE developed by JetBrains. It contains various features, provides great performance and supports many types of frameworks and plugins. It offers great features for Java development including code suggestions and completions based on the user's need, error catching and debugging mechanisms. It allows users to develop GUIs in Java using Swing and JavaFX (in this assignment, Java Swing is utilised). IntelliJ provides multiple support techniques for testing GUI applications and elements in UI testing and unit testing using JUnit and JUnit Swing. It provides excellent code coverage due to its integration with multiple languages/platforms and technologies. Overall, IntelliJ includes robust and latest features in code generation and enhancement, error catching and handling, and debugging tools which provide a great user experience and seamless integration.

## GUI test case design:

In total, 17 testcases were developed each designed to detect various properties of the GUI elements:

### 1. Testcases 1 to 8:

Testcases 1 to 8 are created to check the existence of various GUI elements part of the GUI.

```
void testImageLabelExists() {
    assertNotNull(new ImageViewerApp().imageLabel1);
    assertNotNull(new ImageViewerApp().imageLabel2);
    assertNotNull(new ImageViewerApp().imageLabel3);
}

@Test
void testTextBoxExists() {
    assertNotNull(new ImageViewerApp().textBox1);
    assertNotNull(new ImageViewerApp().textBox2);
    assertNotNull(new ImageViewerApp().textBox3);
}

@Test
void testHeadingLabelExists() {
    assertNotNull(new ImageViewerApp().headingLabel1);
    assertNotNull(new ImageViewerApp().headingLabel2);
    assertNotNull(new ImageViewerApp().headingLabel3);
}

@Test
void testNextButtonExists() {
    assertNotNull(new ImageViewerApp().nextButton1);
    assertNotNull(new ImageViewerApp().nextButton2);
    assertNotNull(new ImageViewerApp().nextButton3);
}

@Test
void testPreviousButtonExists() {
    assertNotNull(new ImageViewerApp().prevButton1);
    assertNotNull(new ImageViewerApp().prevButton2);
    assertNotNull(new ImageViewerApp().prevButton3);
}
```

- a. Testcase1 checks for the existence of Images.
- b. Testcase 2 checks for the existence of Text Boxes
- c. Testcase 3 checks for the existence of the Labels
- d. Testcase 4 and 5 check for the Previous and Next button existences.
- e. Testcases 6,7 and 8 check for Radio Buttons, CheckBoxes and Slider respectively.

## 2. Testcases 9-12:

Testcases 9 to 12 are created to check the size and locations of the following GUI components. Since Page 1 is the default page, showPage1 is not called. The values in the expectedSize and expectedLocation are calculated and incorporated into the code:

- a. Test case 9 checks Image locations and sizes.

```

31      @Test
32      void testImageLabelsLocationAndSize() {
33          JLabel imageLabel1 = app.imageLabel1;
34          Dimension expectedSize1 = new Dimension( width: 640, height: 427);
35          Point expectedLocation1 = new Point( x: 223, y: 5);
36          assertEquals(expectedSize1, imageLabel1.getSize());
37          assertEquals(expectedLocation1, imageLabel1.getLocation());
38
39          app.showPage2();
40          JLabel imageLabel2 = app.imageLabel2;
41          Dimension expectedSize2 = new Dimension( width: 275, height: 183);
42          Point expectedLocation2 = new Point( x: 329, y: 5);
43          assertEquals(expectedSize2, imageLabel2.getSize());
44          assertEquals(expectedLocation2, imageLabel2.getLocation());
45
46          app.showPage3();
47          JLabel imageLabel3 = app.imageLabel3;
48          Dimension expectedSize3 = new Dimension( width: 288, height: 175);
49          Point expectedLocation3 = new Point( x: 318, y: 5);
50          assertEquals(expectedSize3, imageLabel3.getSize());
51          assertEquals(expectedLocation3, imageLabel3.getLocation());
52      }
53

```

- b. Testcase 10 checks text box locations and sizes.



```

104      @Test
105      void testTextBoxLocationAndSize() {
106          JTextField textBox1 = app.textBox1;
107          Dimension expectedSize = new Dimension( width: 105, height: 20);
108          Point expectedLocation = new Point( x: 868, y: 208);
109          assertEquals(expectedSize, textBox1.getSize());
110          assertEquals(expectedLocation, textBox1.getLocation());
111
112          app.showPage2();
113          JTextField textBox2 = app.textBox2;
114          Dimension expectedSize2 = new Dimension( width: 105, height: 20);
115          Point expectedLocation2 = new Point( x: 609, y: 86);
116          assertEquals(expectedSize2, textBox2.getSize());
117          assertEquals(expectedLocation2, textBox2.getLocation());
118
119          app.showPage3();
120          JTextField textBox3 = app.textBox3;
121          Dimension expectedSize3 = new Dimension( width: 105, height: 20);
122          Point expectedLocation3 = new Point( x: 611, y: 82);
123          assertEquals(expectedSize3, textBox3.getSize());
124          assertEquals(expectedLocation3, textBox3.getLocation());
125      }

```

c. Testcase 11 checks heading locations and sizes.

```

127      @Test
128      void testHeadingLabelLocationAndSize() {
129          JLabel headingLabel1 = app.headingLabel1;
130          Dimension expectedSize = new Dimension( width: 119, height: 16);
131          Point expectedLocation = new Point( x: 99, y: 210);
132          assertEquals(expectedSize, headingLabel1.getSize());
133          assertEquals(expectedLocation, headingLabel1.getLocation());
134
135          app.showPage2();
136          JLabel headingLabel2 = app.headingLabel2;
137          Dimension expectedSize2 = new Dimension( width: 191, height: 16);
138          Point expectedLocation2 = new Point( x: 133, y: 88);
139          assertEquals(expectedSize2, headingLabel2.getSize());
140          assertEquals(expectedLocation2, headingLabel2.getLocation());
141
142          app.showPage3();
143          JLabel headingLabel3 = app.headingLabel3;
144          Dimension expectedSize3 = new Dimension( width: 176, height: 16);
145          Point expectedLocation3 = new Point( x: 137, y: 84);
146          assertEquals(expectedSize3, headingLabel3.getSize());
147          assertEquals(expectedLocation3, headingLabel3.getLocation());
148      }

```

d. Testcase 12 checks Next Button locations and sizes.



```

150      @Test
151      void testNextButtonLocationAndSize() {
152          JButton nextButton1 = app.nextButton1;
153          Dimension expectedSize = new Dimension( width: 60, height: 26);
154          Point expectedLocation = new Point( x: 231, y: 524);
155          assertEquals(expectedSize, nextButton1.getSize());
156          assertEquals(expectedLocation, nextButton1.getLocation());
157
158          app.showPage2();
159          JButton nextButton2 = app.nextButton2;
160          Dimension expectedSize2 = new Dimension( width: 60, height: 26);
161          Point expectedLocation2 = new Point( x: 719, y: 83);
162          assertEquals(expectedSize2, nextButton2.getSize());
163          assertEquals(expectedLocation2, nextButton2.getLocation());
164
165          app.showPage3();
166          JButton nextButton3 = app.nextButton3;
167          Dimension expectedSize3 = new Dimension( width: 60, height: 26);
168          Point expectedLocation3 = new Point( x: 721, y: 79);
169          assertEquals(expectedSize3, nextButton3.getSize());
170          assertEquals(expectedLocation3, nextButton3.getLocation());
171      }
172

```

3. Testcases 13 and 14: These testcases check the contents of the Label and the default content of the Text Box.

```

173      @Test
174      void testTextBoxContent() {
175          assertEquals( expected: "Text Box in Page 1", new ImageViewerApp().textBox1.getText());
176          assertEquals( expected: "Text Box in Page 2", new ImageViewerApp().textBox2.getText());
177          assertEquals( expected: "Text Box in Page 3", new ImageViewerApp().textBox3.getText());
178      }
179
180      @Test
181      void testHeadingLabelContent() {
182          assertEquals( expected: "Image Viewer - SVVT", new ImageViewerApp().headingLabel1.getText());
183          assertEquals( expected: "Image Viewer - Functional Testing", new ImageViewerApp().headingLabel2.getText());
184          assertEquals( expected: "Image Viewer - System Testing", new ImageViewerApp().headingLabel3.getText());
185      }
186

```

4. Testcases 15 and 16: These testcases check for the functionality of the Previous and Next Buttons i.e., whether clicking on these buttons change the pages according to the flow.

```

188      @Test
189      void testNextButtonsFunctionality() {
190          app.showPage1();
191          app.nextButton1.doClick();
192          assertEquals(app.panel2, app.frame.getContentPane().getComponent(n: 0));
193
194          app.showPage2();
195          app.nextButton2.doClick();
196          assertEquals(app.panel3, app.frame.getContentPane().getComponent(n: 0));
197
198          app.showPage3();
199          app.nextButton3.doClick();
200          assertEquals(app.panel1, app.frame.getContentPane().getComponent(n: 0));
201      }
202
203      @Test
204      void testPrevButtonsFunctionality() {
205          app.showPage3();
206          app.prevButton3.doClick();
207          assertEquals(app.panel2, app.frame.getContentPane().getComponent(n: 0));
208
209          app.showPage2();
210          app.prevButton2.doClick();
211          assertEquals(app.panel1, app.frame.getContentPane().getComponent(n: 0));
212
213          app.showPage1();
214          app.prevButton1.doClick();
215          assertEquals(app.panel3, app.frame.getContentPane().getComponent(n: 0));
216      }

```

5. Testcases 17: This testcase checks for the default value set on the Slider.

```

218      @Test
219      void testSliderValue() {
220
221          int expectedValue1 = 50;
222          int expectedValue2 = 30;
223          int expectedValue3 = 70;
224
225          int actualValue1 = app.slider1.getValue();
226          int actualValue2 = app.slider2.getValue();
227          int actualValue3;
228          if(app.slider3 == null){
229              actualValue3 = 0;
230          }
231          else {
232              actualValue3 = app.slider3.getValue();
233          }
234
235          assertEquals(expectedValue1, actualValue1);
236          assertEquals(expectedValue2, actualValue2);
237          assertEquals(expectedValue3, actualValue3);
238      }

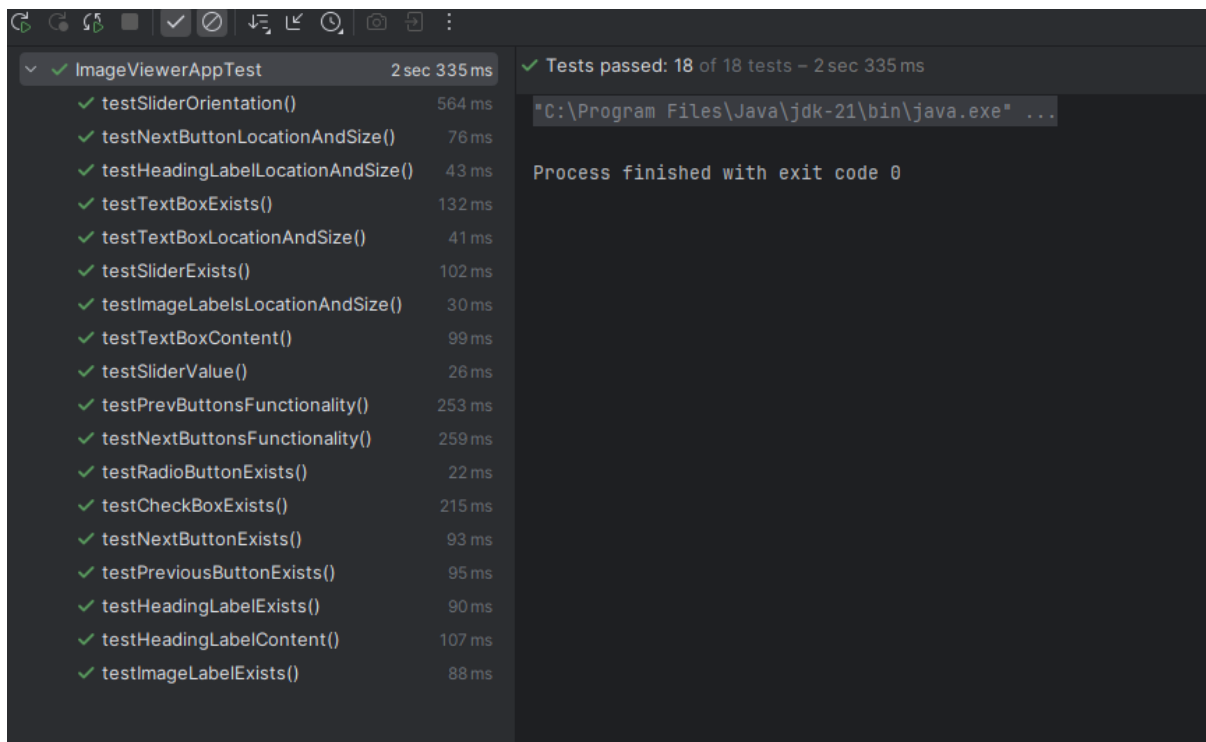
```

6. Testcase 18: This testcase checks for the orientation of the slider i.e., whether it is horizontal or vertical.

```
240      @Test
241      void testSliderOrientation() {
242          int expectedOrientation1 = JSlider.VERTICAL;
243          int expectedOrientation2 = JSlider.HORIZONTAL;
244          int expectedOrientation3 = JSlider.HORIZONTAL;
245
246          int actualOrientation1 = app.slider1.getOrientation();
247          int actualOrientation2 = app.slider2.getOrientation();
248          int actualOrientation3;
249          if(app.slider3 == null){
250              actualOrientation3 = 2;
251          }
252          else {
253              actualOrientation3 = app.slider3.getOrientation();
254          }
255
256          assertEquals(expectedOrientation1, actualOrientation1);
257          assertEquals(expectedOrientation2, actualOrientation2);
258          assertEquals(expectedOrientation3, actualOrientation3);
259      }
```

### Explanation of the Test Results:

#### Test Result for Version1:



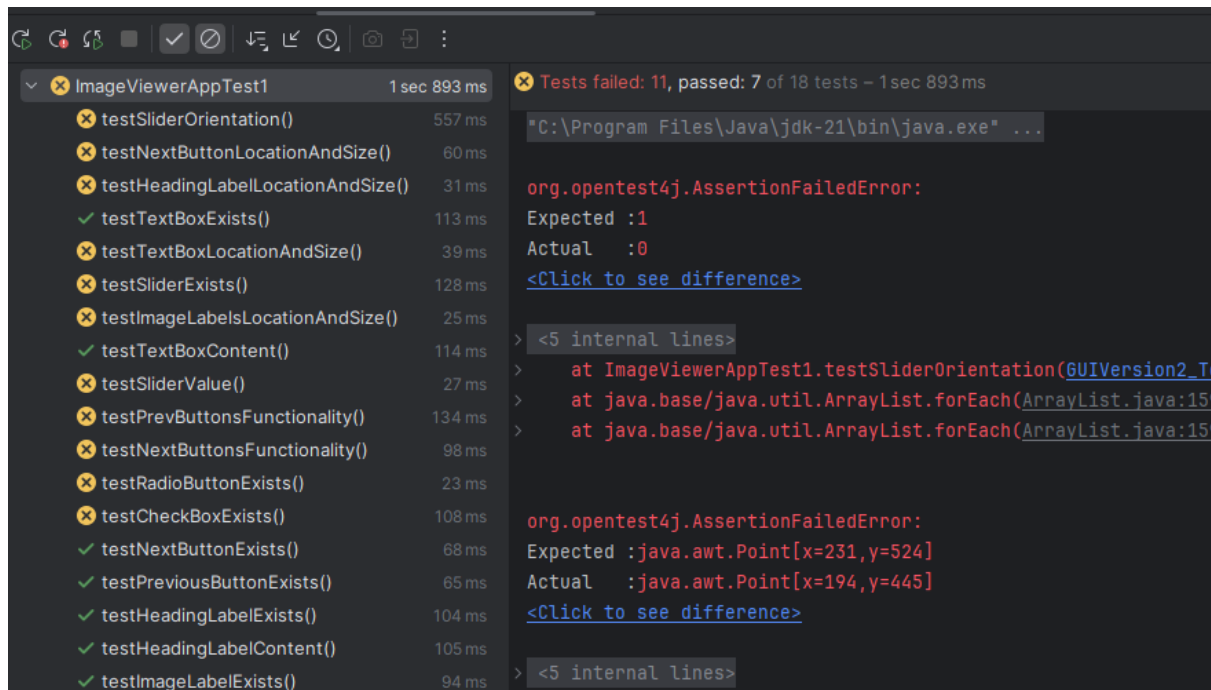
Test Name	Duration
testSliderOrientation()	564 ms
testNextButtonLocationAndSize()	76 ms
testHeadingLabelLocationAndSize()	43 ms
testTextBoxExists()	132 ms
testTextBoxLocationAndSize()	41 ms
testSliderExists()	102 ms
testImageLabelsLocationAndSize()	30 ms
testTextBoxContent()	99 ms
testSliderValue()	26 ms
testPrevButtonsFunctionality()	253 ms
testNextButtonsFunctionality()	259 ms
testRadioButtonExists()	22 ms
testCheckBoxExists()	215 ms
testNextButtonExists()	93 ms
testPreviousButtonExists()	95 ms
testHeadingLabelExists()	90 ms
testHeadingLabelContent()	107 ms
testImageLabelExists()	88 ms

✓ Tests passed: 18 of 18 tests – 2 sec 335 ms

Process finished with exit code 0

All the testcases passed on the Version1 code of the GUI. This shows that all the buttons, labels, checkboxes, radio buttons, sliders, images and other GUI elements are present in the Version 1 code and the locations, sizes of each of these elements are accurate in the Version 1 code. The testcases were developed and the expected values were calculated, interpreted and fed into the testcases such that they pass for the Version 1.

## Test Results for Version2:



Based on the above images, it is observed that only 7 of the total 18 testcases are passing.

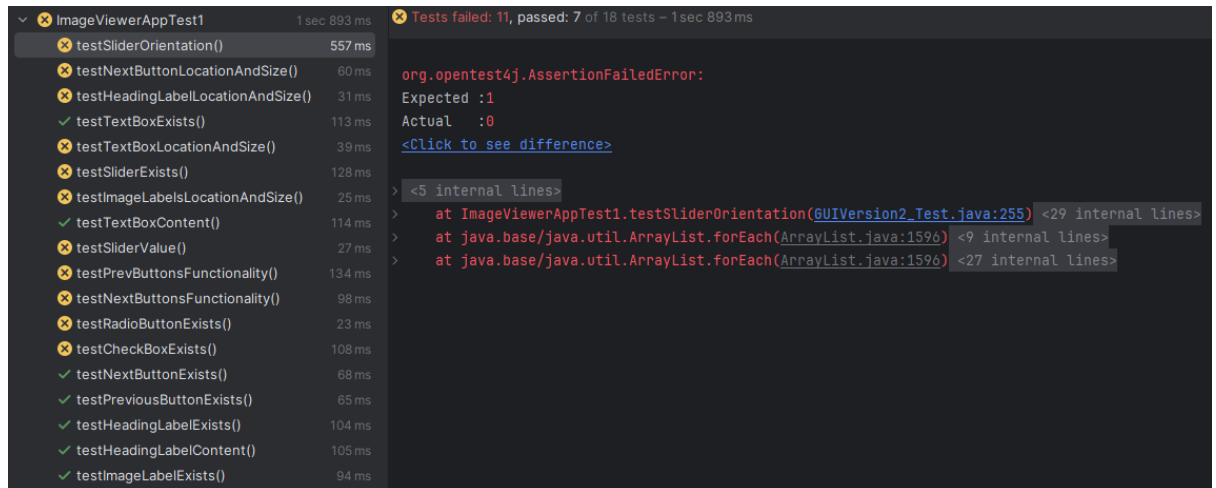
### Pass Testcases:

1. testTextBoxExists: Since this testcase passed, it can be confirmed that there are 3 text boxes, 1 in each page of the Version 2 similar to Version 1.
2. testTextBoxContent(): The success/pass of this testcase means that the content of the text boxes are same between Version 1 and Version2.
3. testNextButtonExists(): Since this testcase passed, it can be confirmed that there are 3 Next Buttons, 1 in each page of the Version 2 similar to Version 1.
4. testPreviousButtonExists(): Since this testcase passed, it can be confirmed that there are 3 Previous Buttons, 1 in each page of the Version 2 similar to Version 1.
5. testNextButtonExists(): Since this testcase passed, it can be confirmed that there are 3 Next Buttons, 1 in each page of the Version 2 similar to Version 1.
6. testHeadingLabelExists(): Since this testcase passed, it can be confirmed that there are 3 Labels, 1 in each page of the Version 2 similar to Version 1.
7. testHeadingLabelContent(): Since this testcase passed, it can be confirmed that the content of the Labels between Version 1 and Version 2 is same.

8. `testImageLabelExists()`: Since this testcase passed, it can be confirmed that there are 3 `ImageLabel`, 1 in each page of the Version 2 similar to Version 1.

Failure Testcases:

1. `testSliderOrientation()`: This testcase failed because in Version 2 Page 1, the Slider is oriented Horizontally as opposed to it being oriented Vertically in Version 1 Page 1. Apart from that, in Version 2 Page 3, Slider is removed, but it is present in Version 1 Page 3. In the test result, it can be seen that the expected value = 1(1 is for Vertical and 0 is for Horizontal Orientations), but the actualValue is 0.



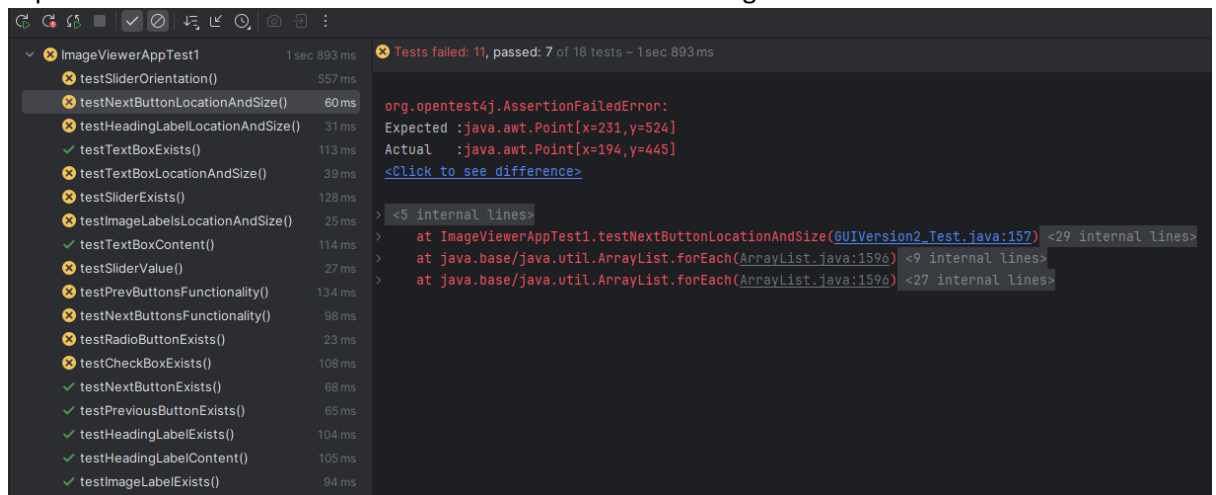
```
▼ ImageViewerAppTest1 1 sec 893 ms
  ✗ testSliderOrientation() 557 ms
  ✗ testNextButtonLocationAndSize() 60 ms
  ✗ testHeadingLabelLocationAndSize() 31 ms
  ✓ testTextBoxExists() 113 ms
  ✗ testTextBoxLocationAndSize() 39 ms
  ✗ testSliderExists() 128 ms
  ✗ testImageLabelsLocationAndSize() 25 ms
  ✓ testTextBoxContent() 114 ms
  ✗ testSliderValue() 27 ms
  ✗ testPrevButtonsFunctionality() 134 ms
  ✗ testNextButtonsFunctionality() 98 ms
  ✗ testRadioButtonExists() 23 ms
  ✗ testCheckBoxExists() 108 ms
  ✓ testNextButtonExists() 68 ms
  ✓ testPreviousButtonExists() 65 ms
  ✓ testHeadingLabelExists() 104 ms
  ✓ testHeadingLabelContent() 105 ms
  ✓ testImageLabelExists() 94 ms

Tests failed: 11, passed: 7 of 18 tests - 1 sec 893 ms

org.opentest4j.AssertionFailedError:
Expected :1
Actual   :0
<Click to see difference>

> <5 internal lines>
> at ImageViewerAppTest1.testSliderOrientation(GUIVersion2_Test.java:255) <29 internal lines>
> at java.base/java.util.ArrayList.forEach(ArrayList.java:1596) <9 internal lines>
> at java.base/java.util.ArrayList.forEach(ArrayList.java:1596) <27 internal lines>
```

2. `testNextButtonLocationAndSize()`: This testcase failed since the GUI elements locations are modified in the Version 2. In the below image, it can be seen that the test result shows the expected location and actual location of the Next Button in Page 1 is different.



```
▼ ImageViewerAppTest1 1 sec 893 ms
  ✗ testSliderOrientation() 557 ms
  ✗ testNextButtonLocationAndSize() 60 ms
  ✗ testHeadingLabelLocationAndSize() 31 ms
  ✓ testTextBoxExists() 113 ms
  ✗ testTextBoxLocationAndSize() 39 ms
  ✗ testSliderExists() 128 ms
  ✗ testImageLabelsLocationAndSize() 25 ms
  ✓ testTextBoxContent() 114 ms
  ✗ testSliderValue() 27 ms
  ✗ testPrevButtonsFunctionality() 134 ms
  ✗ testNextButtonsFunctionality() 98 ms
  ✗ testRadioButtonExists() 23 ms
  ✗ testCheckBoxExists() 108 ms
  ✓ testNextButtonExists() 68 ms
  ✓ testPreviousButtonExists() 65 ms
  ✓ testHeadingLabelExists() 104 ms
  ✓ testHeadingLabelContent() 105 ms
  ✓ testImageLabelExists() 94 ms

Tests failed: 11, passed: 7 of 18 tests - 1 sec 893 ms

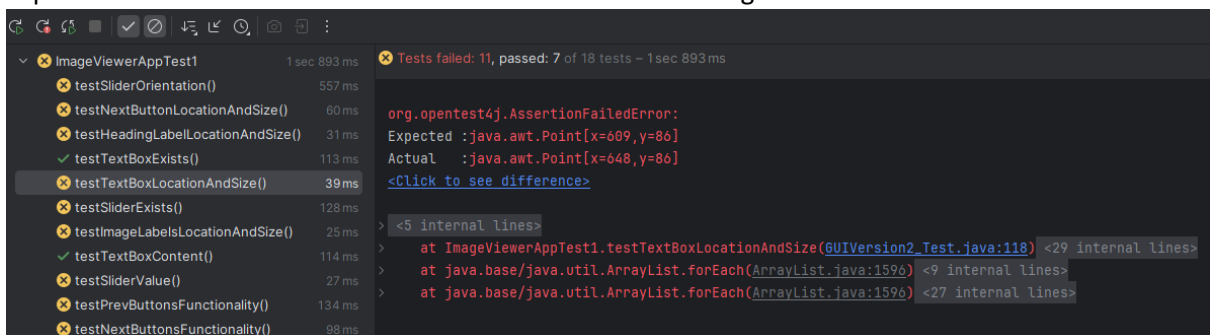
org.opentest4j.AssertionFailedError:
Expected :java.awt.Point[x=231,y=524]
Actual   :java.awt.Point[x=194,y=445]
<Click to see difference>

> <5 internal lines>
> at ImageViewerAppTest1.testNextButtonLocationAndSize(GUIVersion2_Test.java:157) <29 internal lines>
> at java.base/java.util.ArrayList.forEach(ArrayList.java:1596) <9 internal lines>
> at java.base/java.util.ArrayList.forEach(ArrayList.java:1596) <27 internal lines>
```

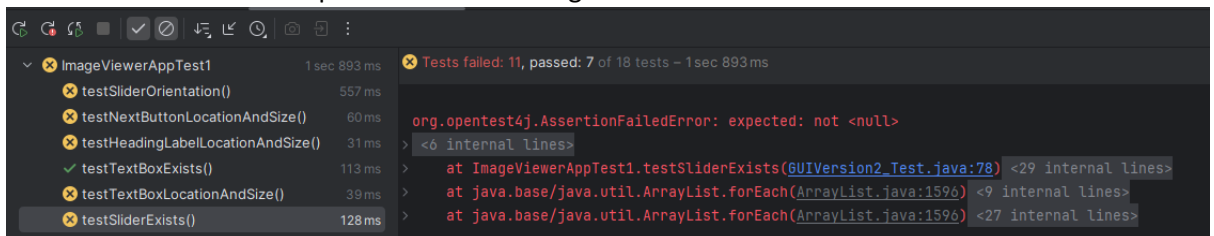
3. `testHeadingLabelLocationAndSize()`: This testcase failed since the GUI elements locations are modified in the Version 2. In the below image, it can be seen that the test result shows the expected location and actual location of the Next Button in Page 1 is different.



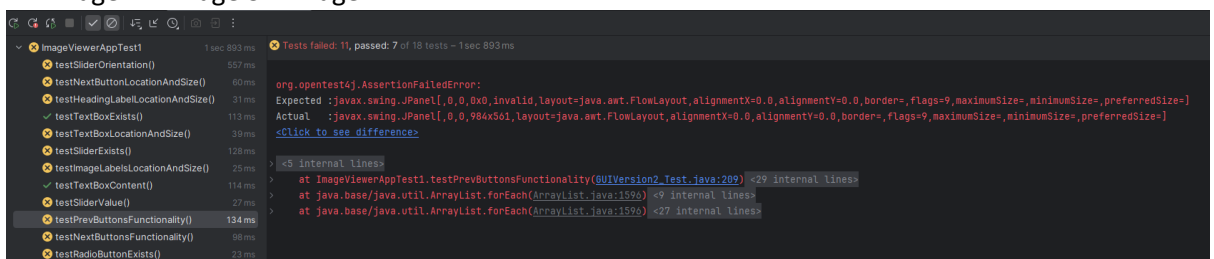
- testTextBoxLocationandSize(): This testcase failed since the GUI elements locations are modified in the Version 2. In the below image, it can be seen that the test result shows the expected location and actual location of the Next Button in Page 1 is different.



- testSliderExists(): This testcase failed because the Slider 3 is removed in GUI Version 2 Page 3. The result shows that it expected a value but it got the value NULL.



- testPrevButtonsFunctionality(): This testcase fails because the flow of the pages in the GUI Version 2 is modified to Page 1 → Page 3 → Page 2 → Page 1 instead of the original flow Page 1 → Page 2 → Page 3 → Page 1.



- testNextButtonsFunctionality(): This testcase fails because the flow of the pages in the GUI Version 2 is modified to Page 1 → Page 3 → Page 2 → Page 1 instead of the original flow Page 1 → Page 2 → Page 3 → Page 1.



```

org.opentest4j.AssertionFailedError:
Expected :javax.swing.JPanel[, 0, 0, 0, invalid, layout=java.awt.FlowLayout, alignmentX=0.0, alignmentY=0.0, borders=, flags=9, maximumSize=, minimumSize=, preferredSize=]
Actual   :javax.swing.JPanel[, 0, 0, 984x561, layout=javax.swing.BoxLayout, alignmentX=0.0, alignmentY=0.0, borders=, flags=8203, maximumSize=, minimumSize=, preferredSize=]
<Click to see difference>
> <5 internal lines>
> at ImageViewerAppTest1.testNextButtonsFunctionality(GUIVersion2_Test.java:198) <29 internal lines>
> at java.base/java.util.ArrayList.forEach(ArrayList.java:1596) <9 internal lines>
> at java.base/java.util.ArrayList.forEach(ArrayList.java:1596) <27 internal lines>

```

8. testRadioButtonExists(): This testcase fails because a Radio Button is removed from Page 1 in GUI Version 2. The result shows that it expected a value but it got the value NULL.

```

org.opentest4j.AssertionFailedError: expected: not <null>
> <6 internal lines>
> at ImageViewerAppTest1.testRadioButtonExists(GUIVersion2_Test.java:53) <29 internal lines>
> at java.base/java.util.ArrayList.forEach(ArrayList.java:1596) <9 internal lines>
> at java.base/java.util.ArrayList.forEach(ArrayList.java:1596) <27 internal lines>

```

9. testCheckBoxExists(): This testcase fails because a Radio Button is removed from Page 2 in GUI Version 2. The result shows that it expected a value but it got the value NULL.

```

org.opentest4j.AssertionFailedError: expected: not <null>
> <6 internal lines>
> at ImageViewerAppTest1.testCheckBoxExists(GUIVersion2_Test.java:67) <29 internal lines>
> at java.base/java.util.ArrayList.forEach(ArrayList.java:1596) <9 internal lines>
> at java.base/java.util.ArrayList.forEach(ArrayList.java:1596) <27 internal lines>

```

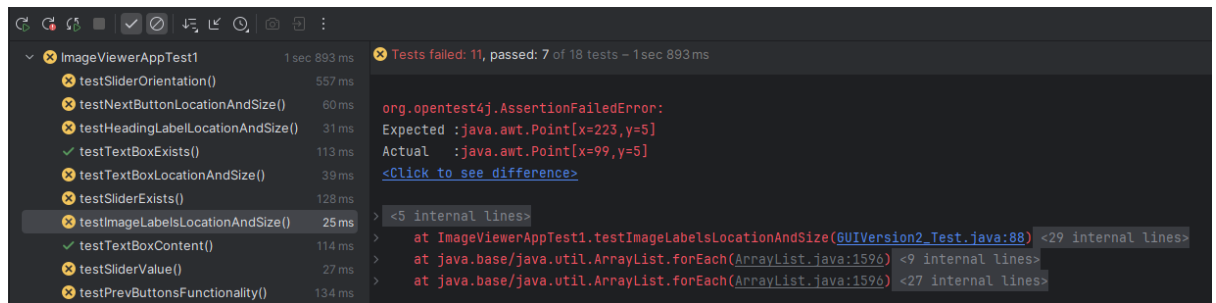
10. testSliderValue(): This testcase fails due to the different default value in the Slider in the Page 2 of the GUI Version 2. The value is 60 as opposed to the expected value 30.

```

org.opentest4j.AssertionFailedError:
Expected :30
Actual   :60
<Click to see difference>
> <5 internal lines>
> at ImageViewerAppTest1.testSliderValue(GUIVersion2_Test.java:234) <29 internal lines>
> at java.base/java.util.ArrayList.forEach(ArrayList.java:1596) <9 internal lines>
> at java.base/java.util.ArrayList.forEach(ArrayList.java:1596) <27 internal lines>

```

11. testImageLabelsLocationandSize(): This testcase fails due to the varied locations of the images in the GUI version 2 compared to Version 1.



## Assessment of the Tool:

1. **Set of Features and Functionalities Provided:**  
IntelliJ has a great variety of features and functionalities by providing code suggestions and information on each of the APIs being used making it easier to code. IntelliJ provides great support in writing and running testcases using frameworks. Its sample testcases help so that the user can easily follow and maintain the testcases. IntelliJ's code debugger inspects the code thoroughly and shows potential errors and warnings in the GUI components, helping the users to write robust code. Throughout the development and testcase generation, IntelliJ provides helpful tips and quick fixes for many errors/warnings which mitigated the problems in the GUI developed for this assignment.
2. **Types of Coverage:** IntelliJ provides various kinds of coverage such as Unit Testing, Integration Testing and UI testing. This allows users to achieve great degree of testing for the GUI applications. The sample testcases provided cover all kinds of testing in turn making it easy for the users to develop the testcases. As seen in the above testcases, IntelliJ allows creation of testcases to check the behaviour of each GUI component individually as seen in checking the existence of elements. Similarly, it also provides support for interactions between components such as clicking the Next Button takes the user from one page to another in the above GUI. Having both individual and combined testing capabilities provides users with great coverage of the GUI.
3. **Reuse of Test Cases:** IntelliJ helps users by facilitating the reuse of test cases and the test case inheritance. It provides users with ways to reuse test code logic across multiple Java test classes. Also, IntelliJ's quick refactor mechanism helps in reuse already written test code and improving efficiency.
4. **Test Results Produced:** IntelliJ provides good information regarding the test runs, test failures, errors occurred and the code in which they occurred which provides users with great detail on how to debug the issue. Also, it provides the complete code flow helping users understand the exact root cause of the issue and solve the issue. It also provides pointers to avoid errors and warnings and code improvements. One issue that was faced while using IntelliJ for this assignment, if there are multiple failures in one testcase, IntelliJ shows the first error which makes the users need to re-run the tests multiple times to ensure all errors are resolved. If all the errors are shown at once, users can easily fix them at once increasing the productivity.

5. **Ease Of Usage:** IntelliJ provides a wide range of plugins and languages support. In addition to that, it gives the implementation information for various APIs of a language/technology which aids users in writing code efficiently and effectively. It has great documentation and provides valuable online resources which can guide users and developers to achieve their tasks. While writing the code for the GUI versions and the test cases, the pointers and analysis provided by IntelliJ's code inspection and debugger tools helped a lot.
6. **Type of GUI elements that can be tested:** IntelliJ supports the testing of various GUI elements such as radio buttons, sliders, buttons, labels, text fields, check boxes among others. Developers can write testcases to check the functionality of individual element and the combination of two or more elements ensuring the complete coverage of the functionality of all the elements.

In total, IntelliJ provides a great way to write and test GUI elements and applications effectively and seamlessly by providing excellent features, extensive documentation, detailed test results and suggestions for improving the quality of the code. It can be used by novice and expert users as its robustness and guidance helps the novice users and the expert users can benefit from its versatile and powerful features.