Testing Documentation

Group 4: Andrew Hocking 215752835, Nourin Abd El Hadi 216107021, Anika Prova 216474306, Nabi Khalid 216441677

Table of Contents:

Section 1

Overview of Testing JavaFX Applications with TestFX:

We tested our application using TestFX, a GUI testing framework for JavaFX. TestFX is a relatively new testing framework that is still in its alpha version. However, it provides convenient and user-friendly automation unit testing for JavaFX GUIs.

Section 2

Configuring TestFX

We used Gradle to add a dependency for TestFX and to get its jar files. The following jar files were added to our project:

testfx-junit-4.0.15-alpha.jar

testfx-core-4.0.15-alpha.jar

junit-4.12.jar

hamcrest-core-1.3.jar

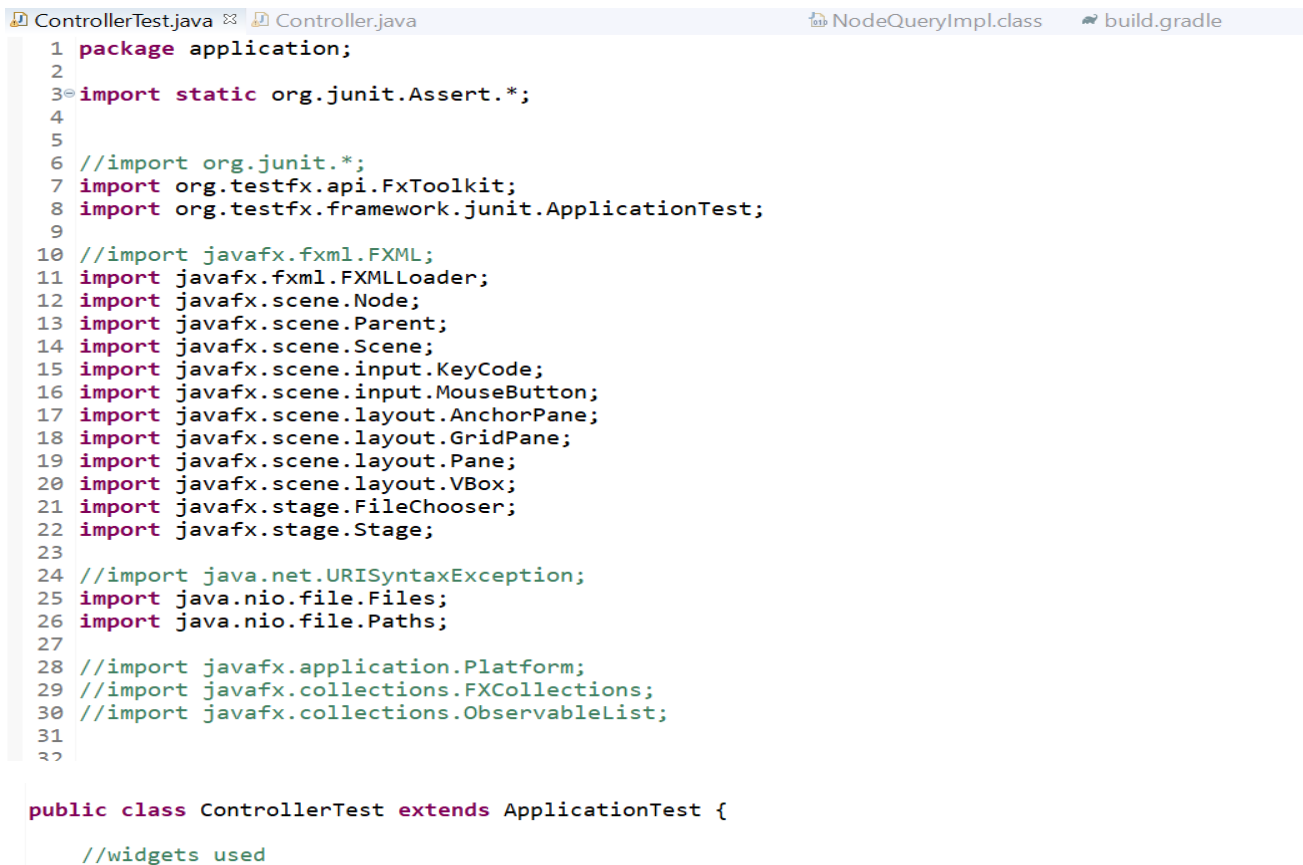assertj-core-3.11.1.jar

```
plugins {
    // Apply the java-library plugin to add support for Java Library
    id 'java-library'
}

repositories {
    // Use jcenter for resolving dependencies.
    // You can declare any Maven/Ivy/file repository here.
    jcenter()
}

dependencies {
    // This dependency is exported to consumers, that is to say found on their compile classpath.
    api 'org.apache.commons:commons-math3:3.6.1'

    // This dependency is used internally, and not exposed to consumers on their own compile classpath.
    implementation 'com.google.guava:guava:28.0-jre'

    // Use JUnit test framework
    testImplementation 'junit:junit:4.12'

    // TestFX
    testCompile "org.testfx:testfx-core:4.0.+"
    testCompile "org.testfx:testfx-junit:4.0.+"
}
```

Section 3

Writing a TestFX Test Class for our Controller Class

TestFX works with JUnit. A TestFX test class must extend the TestFX class ApplicationTest. This requires the implementation of the start operation that has to load the FXML document under test:

3.1 Step 1: We imported all the dependencies and libraries into our Tester Class.

```java
  1 package application;
  2
  3 import static org.junit.Assert.*;
  4
  5
  6 //import org.junit.*;
  7 import org.testfx.api.FxToolkit;
  8 import org.testfx.framework.junit.ApplicationTest;
  9
 10 //import javafx.fxml.FXML;
 11 import javafx.fxml.FXMLLoader;
 12 import javafx.scene.Node;
 13 import javafx.scene.Parent;
 14 import javafx.scene.Scene;
 15 import javafx.scene.input.KeyCode;
 16 import javafx.scene.input.MouseButton;
 17 import javafx.scene.layout.AnchorPane;
 18 import javafx.scene.layout.GridPane;
 19 import javafx.scene.layout.Pane;
 20 import javafx.scene.layout.VBox;
 21 import javafx.stage.FileChooser;
 22 import javafx.stage.Stage;
 23
 24 //import java.net.URISyntaxException;
 25 import java.nio.file.Files;
 26 import java.nio.file.Paths;
 27
 28 //import javafx.application.Platform;
 29 //import javafx.collections.FXCollections;
 30 //import javafx.collections.ObservableList;
 31
 32

public class ControllerTest extends ApplicationTest {

    //widgets used
```

3.2 Step 2: We need to write a few @Before and @After operations that is completed after each GUI Test case.

```java
@Before
public void setUpClass() throws Exception {
    controller = new ControllerTest();
    ApplicationTest.launch(Main.class);
}

@Override
public void start(Stage stage) throws Exception {
    Parent root = FXMLLoader.load(getClass().getResource("/application/Test.fxml"));

    stage.setScene(new Scene(root));
    stage.show();
    stage.toFront();
}

    //retrieve widgets in the GUI. */
    public <T extends Node> T find(final String query) {
        /* TestFX provides many operations to retrieve elements from the loaded GUI. */
        return lookup(query).query();
    }

    @Before
    public void setUp() {
        /*  retrieving the tested widgets from the GUI. */

        circleLeft=find("#circleLeft");
        circleRight=find("#circleRight");
        pane=find("#pane");
        floatingMenu=find("#floatingMenu");
        buttonGrid=find("#buttonGrid");
        settingsPane=find("#settingsPane");


    //clearing events
        @After
        public void tearDown() throws TimeoutException {
            /* Close the window. It will be re-opened at
            FxToolkit.hideStage();
            release(new KeyCode[] {});
            release(new MouseButton[] {});
        }

    //exit controller
        @After
        public void afterEachTest() throws Exception {
            FxToolkit.hideStage();
            release(new KeyCode[] {});
            release(new MouseButton[] {});
        }
```

The *find* operation is just a shortcut to retrieve widgets. It relies on the very useful *lookup* TestFX operation. The IDs defined in the FXML document are used for the

The *setUp* operation retrieves the widgets of the FXML document that will be used in the tests.

The *tearDown* operation is used here to clear and *release* all the possible key or mouse events that are still in progress at the end of each unit test.

3.3 Step 3: After writing the before and after operations, we can now start writing the GUI Test case.

Section 4

Writing GUI Unit Tests

4.1

Test Case 1: public void testWidgetsExist()

This test case tests if all the required widgets are present in the Application.

Deriving the test case: The test case was derived keeping in mind the UI features available in our Application. We found the indexes of each widget using

Implementing the test Case:

```java
@Test
public void testWidgetsExist() {
    final String errorMsg = "One or more widget dont exist";

    assertNotNull(errorMsg,circleLeft);
    assertNotNull(errorMsg,circleRight);
    assertNotNull(errorMsg, pane);
    assertNotNull(errorMsg, floatingMenu);
    assertNotNull(errorMsg,buttonGrid);
    assertNotNull(errorMsg, settingsPane);
    assertNotNull(errorMsg,settingsBox);
    assertNotNull(errorMsg,colorLeft);

    assertNotNull(errorMsg,leftSizeSlider);
    assertNotNull(errorMsg,leftSizeField);
    assertNotNull(errorMsg,colorRight);

    assertNotNull(errorMsg,rightSizeSlider);
    assertNotNull(errorMsg,rightSizeField);
    assertNotNull(errorMsg,menuBar);
    assertNotNull(errorMsg, screenshotButton);
    assertNotNull(errorMsg,frameRect);
    assertNotNull(errorMsg,colorBackground);
    assertNotNull(errorMsg,colorTitles);
    assertNotNull(errorMsg,deleteButton);
    assertNotNull(errorMsg,clearButton);
```

Step #	Step Details	Expected results	Actual results	Pass /Fail /Executed /Suspended
1	assertNotNull() method is called and two arguments are passed in: the error message string and the Widget id. assertNotNull asserts that the object is not null.	No UI elements are null	As expected	Pass

Why the test case was sufficient:

The test is sufficient because the requirement is to have all the widgets in the app in the landing page. We checked each widget for nullity.

4.2

Test Case 2: public void testAddingItems()

This test covers testing for the dragging function into the left circle, right circle and the overlapping region.

Deriving the test case: The test was derived by listing all the fields where we are allowing the user to drop the text items and the regions we are allowing the users to drag the text items from. Finally, we tested the navigation of the text items between the respective regions (from the item list to the Venn Diagram fields)

Implementing the test Case:

```java
@Test
//testing the textFields

public void testAddingItems() {
    for (int i = 1; i <= 15; i++) {
        clickOn("#addItemField").write("Item " + i);
        clickOn("#addItemButton");
    }
    for (int i = 0; i < 3; i++) {
        drag("#itemsList").dropTo("#circleLeftItemsList");
        drag("#itemsList").dropTo("#bothItemsList");
        drag("#itemsList").dropTo("#circleRightItemsList");
    }
}
```

Step #	Step Details	Expected results	Actual results	Pass /Fail /Executed /Suspended
1	Use clickOn method to find the widget itemList	Text field gets clicked on and cursor ready to type	As expected	Pass
2	Use write function to input text into the field	Argument gets written on the text field	As expected	Pass
3	Use clickOn method to activate the addItemButton	The text typed into the text field appears in the items list	As expected	Pass
4	Select and drag an item from the items list using the drag function	Individual items from the items list can be dragged to one of the three destination fields	As expected	Pass

5	Use the dropTo method to drop text to the circleLeftItemsList	Individual item gets added to the circleLeftItemsList	As expected	Pass
6	Use the dropTo method to drop text to the circleRightItemsList	Individual item gets added to the circleRightItemsList	As expected	Pass
7	Use the dropTo method to drop text to the bothItemsList	Individual item gets added to the bothItemsList	As expected	Pass

Why the test case was sufficient:

The test adds 15 items to the items list.

We performed a set of drag and drop function, with three repetitions, allowing a range of 0-15 items to be added to the fields: left circle, right circle and the overlapping region.

In the automation unit testing, we tried to drag items outside the Venn Diagram, and it did not allow it, making sure that this feature is properly functioning.

After iterative testing, we confirmed that the only drag and drop path is properly configured.

4.3

Test Case 3: public void testTitles()

This test case tests the scenario where the user can edit the titles of the main header, left circle and the right circle

Deriving the test: The test was derived by keeping in mind all the fields where we are allowing the user to add/edit title of.

Implementing the test Case:

```
@Test
public void testTitles() {
    clickOn("#title").write("Example Diagram");
    clickOn("#circleLeftTitle").write("Left Circle");
    clickOn("#circleRightTitle").write("Right Circle");
}
```

Step #	Step Details	Expected results	Actual results	Pass /Fail /Executed /Suspended
1	clickOn the title text field to add main title	Text field gets selected	As expected	Pass
2	Type the argument of the write() method into the title text field	Argument is typed	As expected	Pass
3	clickOn the circleLeftTitle field to add title to the left circle	Text field gets selected	As expected	Pass
4	Type the argument of the write() method into the left circle's title text field	Argument is typed	As expected	Pass
5	clickOn the circleRightTitle field to add title to the right circle	Text field gets selected	As expected	Pass
6	Type the argument of the write() method into the right circle's title text field	Argument is typed	As expected	Pass

Why the test case was sufficient:

The test case includes testing all the possible title text Fields where the user can enter text.

It checks for all the errors that can arise, for example, the titles not having proper alignment, the entered titles not being added.

4.4

Test Case 4: public void testInput ()

This test checks if the input text is being entered the items list of the Venn Diagram

Deriving the test case:

The test was derived through considering the input method of the text field and the final outcome, that is, the text appearing in the list after button is clicked.

Implementing the test Case:

```
// Insert more tests here
@Test
public void testInput () {
    //TextField newItem =  (TextField) find("#newItem");

    clickOn("#addItemField").write("This is a test input");
    clickOn("#addItemButton");

}
```

Step #	Step Details	Expected results	Actual results	Pass /Fail /Executed /Suspended
1	Implement the clickOn method to select the add item text field	The text field has a cursor and is ready to enter text.	As expected	Pass
2	Implement the write method to type the passed argument in the field.	The argument gets typed into the text field	As expected	Pass
3	Implement the clickOn method to insert the text field into the items list.	The new item text field gets inserted into the list	As expected	Pass

Why the test case was sufficient:

We considered all errors that could arise. For example, the text not being typed upon clicking the input text field and also the text not being added to the items list after the add button was pressed.

We tested for every scenario and did not leave any room for unnoticed error to arise during inputting text.

4.5

Test Case 5: public void testDeleteButton () {

The test checks if the delete button functions properly and is able to delete the text selected from the items list.

Deriving the test:

The test was derived by keeping the UI feature in mind. Upon clicking on the delete button, the selected text should be removed from the list.

Implementing the test Case:

```java
@SuppressWarnings("unchecked")
//testing the delete button
@Test
public void testDeleteButton () {
//TextField newItem =  (TextField) find("#newItem");

    for (int i = 1; i <= 6; i++) {
        clickOn("#addItemField").write("Item " + i);
        clickOn("#addItemButton");
    }

    WaitForAsyncUtils.waitForFxEvents();

    final int length = ((ListView<String>)(find("#itemsList"))).getItems().size();

    clickOn("#itemsList");
    clickOn("#deleteButton");

    WaitForAsyncUtils.waitForFxEvents();

    assertEquals(((ListView<String>)(find("#itemsList"))).getItems().size(), length-1);

}
// ·   · ·
```

Step #	Step Details	Expected results	Actual results	Pass /Fail /Executed /Suspended
1	Implement the clickOn method to select the add item text field	The text field has a cursor and is ready to enter text.	As expected	Pass
2	Implement the write method to type the passed argument in the field.	The argument gets typed into the text field	As expected	Pass
3	Implement the clickOn method to insert the text field into the items list.	The new item text field gets inserted into the list	As expected	Pass
4	Implement the clickOn method to click on the delete button	The selected item on the list is removed.	As expected	Pass
5	Implement the assertEquals method and check if the ArrayList<String> of the item list now has one less item than the original size of the list.	The list has one fewer item in it	As expected	Pass

Why the test case was sufficient:

This test case overs all the possible scenarios that can arise from the UI feature.

4.5

Test Case 5: public void testclearButton ()

The test checks if the clear button functions properly and is able to clear text lists from the Venn Diagram and move it back to the items list.

Deriving the test:

The test was derived based on the pre and post state of the ArrayList<String> of the Venn Diagram and the Items list after clicking on the clear button.

Implementing the test Case:

```
//clear button
@SuppressWarnings("unchecked")
@Test
public void testClearButton () {
    for (int i = 1; i <= 15; i++) {
        clickOn("#addItemField").write("Item " + i);
        clickOn("#addItemButton");
    }

    for (int i = 0; i < 3; i++) {
        drag("#itemsList").dropTo("#circleLeftItemsList");
        drag("#itemsList").dropTo("#bothItemsList");
        drag("#itemsList").dropTo("#circleRightItemsList");
    }
    clickOn("#clearButton");

    WaitForAsyncUtils.waitForFxEvents();

    assertEquals(((ListView<String>)(find("#circleLeftItemsList"))).getItems().size(), 0);
    assertEquals(((ListView<String>)(find("#circleRightItemsList"))).getItems().size(), 0);
    assertEquals(((ListView<String>)(find("#bothItemsList"))).getItems().size(), 0);


}
```

Step #	Step Details	Expected results	Actual results	Pass /Fail /Executed /Suspended
1	Implement the clickOn method to select the add item text field	The text field has a cursor and is ready to enter text.	As expected	Pass
2	Implement the write method to type the passed argument in the field.	The argument gets typed into the text field	As expected	Pass
3	Implement the clickOn method to insert the text field into the items list.	The new item text field gets inserted into the list	As expected	Pass

	Implement the drag method followed by the dropTo method to insert the itemsList in the Venn Diagram	circleLeftItemsList/ bothItemsList / circleRightItems list contains the itemsList dragged into it		
4	Implement the clickOn method to click on the clear button	The selected item on the list is removed.	As expected	Pass
	Wait for the JavaFX feature to be completely executed			
5	Implement the assertEquals method and check if the length of the ArrayList<String> of the circleLeftItemsList/ bothItemsList / circleRightItems are each 0	The circle lists should be empty (no items in diagram)	As expected	Pass

4.6

Test Case 6: public void testColourPicker ():

This test case checks the colour fill of the main front-end figures: The left circle and the right circle.

Deriving the test:

The test was derived by first setting the fill of the shapes and then testing if the colors have been set property by comparing the colors with known color value.

Implementing the test:

This test has not yet been fully implemented, but the steps will be as shown:

Step #	Step Details	Expected results	Actual results	Pass /Fail /Executed /Suspended
1	Change colour value using the colour picker for the left circle	Left circle colour will change to the selected colour		
2	Change colour value using the colour picker for the right circle	Right circle colour will change to the selected colour		

Section 5

Testing Coverage Metrics

Test coverage is given details about the level to which the written coding of our Venn Diagram Application has been tested. Here are the details provided:

Problems @ Javadoc Declaration **Console** Gradle Executions Git Staging Coverage

ControllerTest (23-Feb-2020 10:19:02 PM)

Element	Covera...	Covered Ins...	Missed Instr...	Total Instruc...
> Venn	40.7 %	1,254	1,828	3,082

Conclusion:

Most of the UI features were tested. Coverage can be further improved by including tests for front-end features like font size, font-color etc. However, since TestFX is an automation unit testing, we can observe and conclude that the front-end features were up to the mark and satisfactory.