Testing Document

Noor Ahmed Sadique 215605306
Tariq Qureshey 216789166
Kamal Patel 216280224

Table of Contents

1.0 INTRODUCTION

This Testing document will be regarding test cases within our software that perform crucial tasks to ensure the software runs smoothly. As well as showing the test coverage of our software.

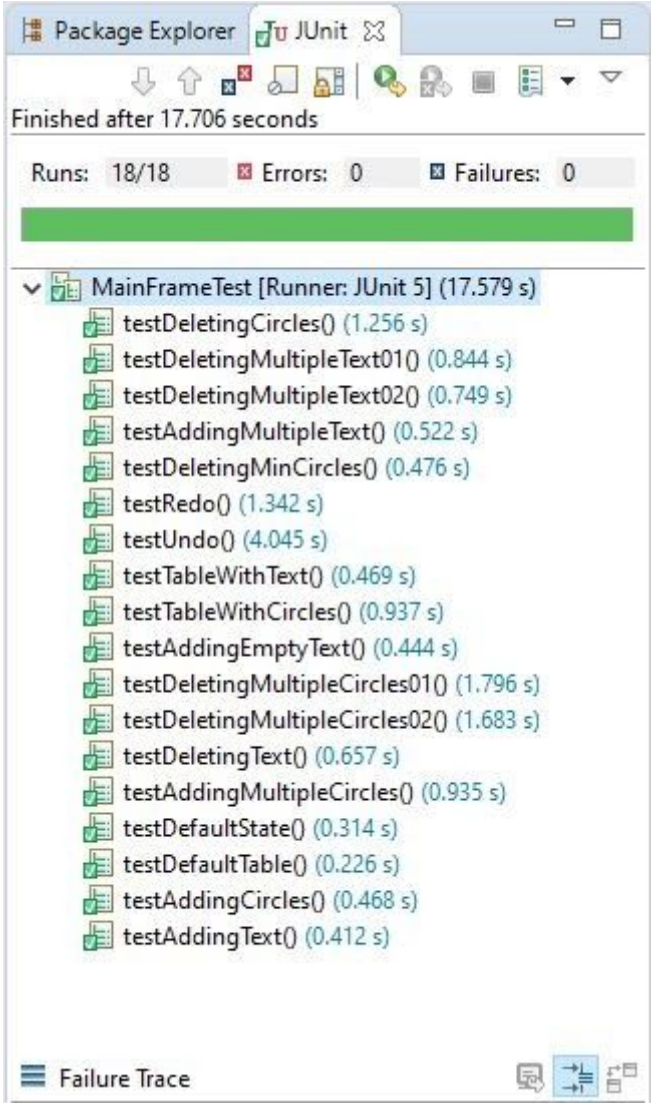
2.0 OBJECTIVES AND TASKS

Objective: We are testing our program with JUnit tests, these tests will ensure that our program runs as expected and the customer will be satisfied with the product.

3.0 TESTING STRATEGY

Our testing strategy including creating a plethora of JUnit tests, covering the most important aspects of our software. These include checking if default fonts are used rather than selected ones, how many circles are being created based on the input from user, and much more. Below, you will see various screenshots and explanations of how we tested our software

3.1 JUNIT TESTING



As you can see above, we tested our program with a plethora of methods. The tests above outline all the possible tests with the MainFrame of the program. These tests are sufficient as it tests the use cases of the program. We derived these test cases by understanding all the possible ways the user might go about using our software. By keeping this in mind, we are able to be certain that our program runs the way it should when presented to the customer.

3.1.1 TEST CASES DESCRIPTION

Test Case Method	**Brief Description**
testDeletingCircles()	This will test adding one circle and then deleting one circle.
testDeletingMultipleText01()	This will test deleting multiple texts by first adding multiple texts and then deleting them.
testDeletingMultipleText02()	This will also test deleting multiple texts by first adding and then deleting.
testAddingMultipleText()	This will test adding multiple texts.
testDeletingMinCircles()	This will test that the user is not able to delete more circles than minimum circles which are 2. For example, if the user has selected all circles, then all circles will get deleted except 2 circles.
testRedo()	This will test the Redo mechanism. If the user presses Ctrl+Z, to undo the added circle, then pressing Ctrl+Y will redo meaning that the circle will be added again with same properties.
testUndo()	This will test the Undo mechanism. If the user presses Ctrl+Z, then it will undo the change. For example, if the users added a circle and after they press Ctrl+Z, then that circle will be removed, undoing one action.
testTableWithText()	This will test if the texts are added to the table that is on the right-hand side of the app.
testTableWithCircles()	This will test if the Circles are added to the table that is on the right-hand side of the app.
testAddingEmptyText()	This will test that adding an empty text will do nothing meaning that no text is added.
testDeletingMultipleCircles01()	This will test deleting multiple circles by first adding them and then deleting them.
testDeletingMultipleCircles02()	This will test both deleting multiple circles and their minimum properties. It will add 5 circles in total and delete all of them. It is expected to remain 2 circles and the rest of them getting deleted out of 5.
testDeletingText()	This will test deleting the added text.
testAddingMultipleCircles()	This will test adding multiple circles on the app.

testDefaultState()	This will test the initial state when the program is run.
testDefaultTable()	This will test the initial table that is on the right-hand side when the program is run.
testAddingCircles()	This will test adding a single circle on the app.
testAddingText()	This will test adding a single text on the app.

All the above test cases were tested for use cases which included adding single/multiple circles and texts, changing properties for each circle or text, undo/redo mechanism, save, open, about, or close.

3.1.2 IMPLEMENTATION IMAGES

Here are the images of the implementation of each test case methods:

```
@Test
void testDefaultState() throws InterruptedException {
    Thread.sleep(100);
    assertEquals(2,mf.CI.size());
    assertEquals(0,mf.TI.size());
    Thread.sleep(100);
}

@Test
void testAddingCircles() throws InterruptedException, AWTException {
    Thread.sleep(100);
    assertEquals(2,mf.CI.size());
    assertEquals(0,mf.TI.size());
    addClick();
    assertEquals(3,mf.CI.size());
    assertEquals(0,mf.TI.size());

}

@Test
void testAddingMultipleCircles() throws InterruptedException, AWTException {
    Thread.sleep(100);
    assertEquals(2,mf.CI.size());
    assertEquals(0,mf.TI.size());
    assertEquals(1,mf.getAddValue());

    mf.addCircleAndText.setValue("3");
    addClick();
    Thread.sleep(500);
    assertEquals(5,mf.CI.size());
    assertEquals(0,mf.TI.size());


}
```

```java
@Test
void testAddingEmptyText() throws InterruptedException, AWTException {
    Thread.sleep(100);
    assertEquals(2,mf.CI.size());
    assertEquals(0,mf.TI.size());

    addTextClick();
    assertEquals(2,mf.CI.size());
    assertEquals(0,mf.TI.size());


}


@Test
void testAddingText() throws InterruptedException, AWTException {
    Thread.sleep(100);
    assertEquals(2,mf.CI.size());
    assertEquals(0,mf.TI.size());

    mf.txtAddText.setText("Hello World!");
    addTextClick();
    assertEquals(2,mf.CI.size());
    assertEquals(1,mf.TI.size());


}



@Test
void testAddingMultipleText() throws InterruptedException, AWTException {
    Thread.sleep(100);
    assertEquals(2,mf.CI.size());
    assertEquals(0,mf.TI.size());

    mf.addCircleAndText.setValue("3");
    mf.txtAddText.setText("Hello World!");
    addTextClick();
    assertEquals(2,mf.CI.size());
    assertEquals(3,mf.TI.size());


}

@Test
void testDefaultTable() throws InterruptedException {
    Thread.sleep(100);
    assertEquals(2,mf.CI.size());
    assertEquals(0,mf.TI.size());

    assertEquals(2, mf.jtable.getRowCount());

}
```

```java
@Test
void testTableWithCircles() throws InterruptedException, AWTException {
    Thread.sleep(100);
    assertEquals(2,mf.CI.size());
    assertEquals(0,mf.TI.size());
    assertEquals(2,mf.jtable.getRowCount());
    mf.jtable.setRowSelectionInterval(0, 1);
    assertTrue(mf.isSameType(mf.CIRCLE_TYPE));

    mf.addCircleAndText.setValue("3");
    addClick();
    Thread.sleep(500);
    mf.jtable.setRowSelectionInterval(0, 4);
    assertEquals(5,mf.CI.size());
    assertEquals(0,mf.TI.size());
    assertEquals(5,mf.jtable.getRowCount());
    assertTrue(mf.isSameType(mf.CIRCLE_TYPE));

}
@Test
void testTableWithText() throws InterruptedException, AWTException {
    Thread.sleep(100);
    assertEquals(2,mf.CI.size());
    assertEquals(0,mf.TI.size());
    assertEquals(2,mf.jtable.getRowCount());
    assertFalse(mf.isSameType(mf.TEXT_TYPE));

    mf.addCircleAndText.setValue("3");
    mf.txtAddText.setText("Hello World!");
    addTextClick();
    assertEquals(2,mf.CI.size());
    assertEquals(3,mf.TI.size());
    mf.jtable.setRowSelectionInterval(2, 4);
    assertEquals(5,mf.jtable.getRowCount());
    assertTrue(mf.isSameType(mf.TEXT_TYPE));

}
```

```java
@Test
void testDeletingMinCircles() throws InterruptedException, AWTException {
    Thread.sleep(100);
    assertEquals(2,mf.CI.size());
    assertEquals(0,mf.TI.size());
    assertEquals(2,mf.jtable.getRowCount());
    mf.jtable.setRowSelectionInterval(0, 0);
    deleteClick();
    assertEquals(2,mf.CI.size());
    assertEquals(0,mf.TI.size());
    assertEquals(2,mf.jtable.getRowCount());

}

@Test
void testDeletingCircles() throws InterruptedException, AWTException {
    Thread.sleep(100);
    assertEquals(2,mf.CI.size());
    assertEquals(0,mf.TI.size());
    assertEquals(2,mf.jtable.getRowCount());
    addClick();
    assertEquals(3,mf.CI.size());
    assertEquals(0,mf.TI.size());
    assertEquals(3,mf.jtable.getRowCount());

    mf.jtable.setRowSelectionInterval(0, 0);
    deleteClick();
    assertEquals(2,mf.CI.size());
    assertEquals(0,mf.TI.size());
    assertEquals(2,mf.jtable.getRowCount());

}
```

```java
@Test
void testDeletingMultipleCircles01() throws InterruptedException, AWTException {
    Thread.sleep(100);
    assertEquals(2,mf.CI.size());
    assertEquals(0,mf.TI.size());
    assertEquals(2,mf.jtable.getRowCount());
    mf.addCircleAndText.setValue("3");
    addClick();
    Thread.sleep(500);
    assertEquals(5,mf.CI.size());
    assertEquals(0,mf.TI.size());
    assertEquals(5,mf.jtable.getRowCount());

    mf.jtable.setRowSelectionInterval(0, 2);
    Thread.sleep(100);
    deleteClick();
    Thread.sleep(500);
    assertEquals(2,mf.CI.size());
    assertEquals(0,mf.TI.size());
    assertEquals(2,mf.jtable.getRowCount());


}
```

```java
@Test
void testDeletingMultipleCircles02() throws InterruptedException, AWTException {
    Thread.sleep(100);
    assertEquals(2,mf.CI.size());
    assertEquals(0,mf.TI.size());
    assertEquals(2,mf.jtable.getRowCount());
    mf.addCircleAndText.setValue("3");
    addClick();
    Thread.sleep(500);
    assertEquals(5,mf.CI.size());
    assertEquals(0,mf.TI.size());
    assertEquals(5,mf.jtable.getRowCount());

    mf.jtable.setRowSelectionInterval(0, 4);
    deleteClick();
    Thread.sleep(500);
    assertEquals(2,mf.CI.size());
    assertEquals(0,mf.TI.size());
    assertEquals(2,mf.jtable.getRowCount());


}
```

```java
@Test
void testDeletingText() throws InterruptedException, AWTException {
    Thread.sleep(100);
    assertEquals(2,mf.CI.size());
    assertEquals(0,mf.TI.size());
    assertEquals(2,mf.jtable.getRowCount());
    mf.txtAddText.setText("Hello World!");
    addTextClick();
    assertEquals(2,mf.CI.size());
    assertEquals(1,mf.TI.size());
    assertEquals(3,mf.jtable.getRowCount());

    mf.jtable.setRowSelectionInterval(2, 2);
    deleteClick();
    assertEquals(2,mf.CI.size());
    assertEquals(0,mf.TI.size());
    assertEquals(2,mf.jtable.getRowCount());


}
```

```java
@Test
void testDeletingMultipleText01() throws InterruptedException, AWTException {
    Thread.sleep(100);
    assertEquals(2,mf.CI.size());
    assertEquals(0,mf.TI.size());
    assertEquals(2,mf.jtable.getRowCount());
    mf.txtAddText.setText("Hello World!");
    mf.addCircleAndText.setValue("3");
    addTextClick();
    assertEquals(2,mf.CI.size());
    assertEquals(3,mf.TI.size());
    assertEquals(5,mf.jtable.getRowCount());

    mf.jtable.setRowSelectionInterval(2, 4);
    deleteClick();
    assertEquals(2,mf.CI.size());
    assertEquals(0,mf.TI.size());
    assertEquals(2,mf.jtable.getRowCount());


}
```

```java
@Test
void testDeletingMultipleText02() throws InterruptedException, AWTException {
    Thread.sleep(100);
    assertEquals(2,mf.CI.size());
    assertEquals(0,mf.TI.size());
    assertEquals(2,mf.jtable.getRowCount());
    mf.txtAddText.setText("Hello World!");
    mf.addCircleAndText.setValue("7");
    addTextClick();
    assertEquals(2,mf.CI.size());
    assertEquals(7,mf.TI.size());
    assertEquals(9,mf.jtable.getRowCount());

    mf.jtable.setRowSelectionInterval(2, 8);
    deleteClick();
    assertEquals(2,mf.CI.size());
    assertEquals(0,mf.TI.size());
    assertEquals(2,mf.jtable.getRowCount());


}
```

```java
@Test
void testRedo() throws InterruptedException, AWTException {
    Thread.sleep(100);
    assertEquals(2,mf.CI.size());
    assertEquals(0,mf.TI.size());
    assertEquals(2,mf.jtable.getRowCount());


    mf.txtAddText.setText("Hello World!");
    mf.addCircleAndText.setValue("1");
    addTextClick();
    mf.addCircleAndText.setValue("3");
    addClick();
    mf.jtable.setRowSelectionInterval(0, 1);
    deleteClick();
    mf.addCircleAndText.setValue("1");
    addClick();
    mf.txtAddText.setText("Hello World!");
    addTextClick();

    ArrayList<CircleInfo> expCI = mf.CI;
    ArrayList<TextInfo> expTI = mf.TI;

    //8 actions performed, undo these
    mf.frame.requestFocus();
    Robot r = new Robot();
    for(int i = 0; i < 8; i++) {
        r.keyPress(KeyEvent.VK_CONTROL);
        r.keyPress(KeyEvent.VK_Z);
        r.keyRelease(KeyEvent.VK_CONTROL);
        r.keyRelease(KeyEvent.VK_Z);
    }

    ///redo these 8 actions
    for(int i = 0; i < 8; i++) {
        r.keyPress(KeyEvent.VK_CONTROL);
        r.keyPress(KeyEvent.VK_Y);
        r.keyRelease(KeyEvent.VK_CONTROL);
        r.keyRelease(KeyEvent.VK_Y);
    }
    assertEquals(expCI,mf.CI);
    assertEquals(expTI,mf.TI);

}
```

```java
@Test
void testUndo() throws InterruptedException, AWTException {
    MainFrame mf = new MainFrame();
    Thread.sleep(100);
    assertEquals(2,mf.CI.size());
    assertEquals(0,mf.TI.size());
    assertEquals(2,mf.jtable.getRowCount());
    ArrayList<CircleInfo> expCI = mf.CI;
    ArrayList<TextInfo> expTI = mf.TI;
    mf.txtAddText.setText("Hello World!");
    mf.addCircleAndText.setValue("1");
    addTextClick();
    Thread.sleep(500);
    mf.addCircleAndText.setValue("3");
    addClick();
    Thread.sleep(500);
    mf.jtable.setRowSelectionInterval(0, 1);
    deleteClick();
    Thread.sleep(500);
    mf.addCircleAndText.setValue("1");
    addClick();
    Thread.sleep(500);
    mf.txtAddText.setText("Hello World!");
    addTextClick();
    Thread.sleep(500);

    //8 actions performed, undo these
    mf.frame.requestFocus();
    Robot r = new Robot();
    for(int i = 0; i < 8; i++) {
        r.keyPress(KeyEvent.VK_CONTROL);
        r.keyPress(KeyEvent.VK_Z);
        r.keyRelease(KeyEvent.VK_CONTROL);
        r.keyRelease(KeyEvent.VK_Z);
    }
    assertEquals(expCI,mf.CI);
    assertEquals(expTI,mf.TI);
}
```

Below are the three helper methods that were used in each of the above test case methods.

```java
public void addClick() throws InterruptedException, AWTException {
    Robot r = new Robot();
    Thread.sleep(100);
    r.mouseMove(1020, 370);
    r.mousePress(InputEvent.BUTTON1_DOWN_MASK);
    r.mouseRelease(InputEvent.BUTTON1_DOWN_MASK);
    Thread.sleep(100);
}

public void deleteClick() throws InterruptedException, AWTException  {
    Robot r = new Robot();

    Thread.sleep(100);
    r.mouseMove(1200, 370);
    r.mousePress(InputEvent.BUTTON1_DOWN_MASK);
    r.mouseRelease(InputEvent.BUTTON1_DOWN_MASK);
    Thread.sleep(100);
}

public void addTextClick() throws InterruptedException, AWTException  {
    Robot r = new Robot();

    Thread.sleep(100);
    r.mouseMove(1020, 470);
    r.mousePress(InputEvent.BUTTON1_DOWN_MASK);
    r.mouseRelease(InputEvent.BUTTON1_DOWN_MASK);
    Thread.sleep(100);
}
```

3.2 TEST COVERAGE

Here are the images of the test coverage for our system with a comparison between minimum usage and average usage. It shows all of the classes in our program with percentages showing the coverages.

3.2.1 MINIMUM USAGE:

Coverage

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
Venn	30.9 %	3,436	7,667	11,103
src/main/java	34.0 %	3,436	6,672	10,108
venn	34.0 %	3,436	6,667	10,103
MainFrame.java	27.6 %	1,900	4,978	6,878
JFontChooser.java	76.9 %	1,115	335	1,450
ReadAndWriteObject.java	0.0 %	0	290	290
Text.java	0.0 %	0	242	242
CircleInfo.java	40.7 %	136	198	334
Draw.java	67.5 %	285	137	422
ObjectInfo.java	0.0 %	0	128	128
WelcomeFrame.java	0.0 %	0	121	121
TextInfo.java	0.0 %	0	112	112
ExpandedArea.java	0.0 %	0	77	77
CustomFilter.java	0.0 %	0	38	38
WelcomeFrameRunner.java	0.0 %	0	8	8
CustomTableModel.java	0.0 %	0	3	3
(default package)	0.0 %	0	5	5
src/test/java	0.0 %	0	995	995

3.2.2 AVERAGE USAGE:

Coverage

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
Venn	63.7 %	7,076	4,027	11,103
src/main/java	70.0 %	7,076	3,032	10,108
venn	70.0 %	7,076	3,027	10,103
MainFrame.java	67.0 %	4,607	2,271	6,878
ReadAndWriteObject.java	30.0 %	87	203	290
WelcomeFrame.java	0.0 %	0	121	121
JFontChooser.java	92.3 %	1,339	111	1,450
Draw.java	76.8 %	324	98	422
ObjectInfo.java	53.9 %	69	59	128
Text.java	79.3 %	192	50	242
CircleInfo.java	85.6 %	286	48	334
TextInfo.java	62.5 %	70	42	112
ExpandedArea.java	83.1 %	64	13	77
WelcomeFrameRunner.java	0.0 %	0	8	8
CustomTableModel.java	0.0 %	0	3	3
CustomFilter.java	100.0 %	38	0	38
(default package)	0.0 %	0	5	5
src/test/java	0.0 %	0	995	995
venn	0.0 %	0	983	983
(default package)	0.0 %	0	12	12