Graphical User Interface Testing

Testing JavaFX GUIs with TestFX

Lorenzo Cimini

June 22, 2022

Outline

1. **GUI Testing**
    - *GUI vs UI*
    - *What is GUI Testing*
    - *Why GUI testing*
    - *What is tested with GUI Testing*
    - *GUI Testing techniques*
2. **TestFx**
    - *TestFX*
    - *Include TestFX in your project*
    - *JavaFX main points*
    - *Structure of a TestFX test class*
    - *How to move inside your JavaFX SceneGraph using FxRobot*
    - *How to assert with TestFX*
    - *Headless testing with Monocle*
3. **Coding**

GUI Testing

GUI vs UI

UI

A **user interface** (UI) is the space where interactions between humans and machines occur. [...][1]

GUI

A **Graphical user interfaces** (GUI) accept input via devices such as a computer keyboard and mouse and provide articulated graphical output on the computer monitor. [...][2]

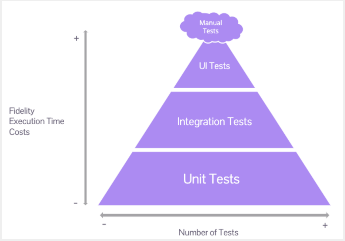
[1] `https://en.wikipedia.org/wiki/User_interface`
[2] `https://en.wikipedia.org/wiki/Graphical_user_interface`

What is GUI Testing

- GUIs, as software and systems, can be imperfect as they are created by human beings
- As a verification technique, the goal is to check the consistency of an implementation with a specification
- In this case the SUT is our GUI thus our goal is to ensure a certain degree of quality in the interaction between our system and the user

Why GUI Testing

- Think as a user
- It's an addition rather than an alternative to the traditional unit test
    - When a problem raises both with unit test and GUI testing, its easier to find the bug with the traditional unit test rather than with the GUI test due the smaller number of 'moving parts'
    - Some parts of the GUI are hard to test with the traditional unit test.



What is tested with GUI testing

GUI's aesthetic

- Verify that all GUI's components are compliant with respect to:
    - Dimension
    - Position
    - Width
    - ...
- Check the screen for spelling mistakes and misaligned elements
- Verify right positioning and size of GUI's sections
- ...

GUI's functionalities

- Verify that error messages are shown when user's input is wrong and vice versa
- Does 'add' button actually add a new entry into the database?
- Are input checks working as expected?
- Does the popup show when an error occured?
- ...

GUI Testing techniques

1. Manual based
   - Doesn’t require coding skills
   - Slow, error prone, ...
2. Record and play
   - Doesn’t need programming skill
3. Code-based
   - Requires coding skills
   - Reusability, less human errors, automatable

TestFX

TestFX

- Open source
- Automated tests for testing JavaFX GUIs
- Multiple testing frameworks supported (JUnit 4, JUnit 5 and Spock)
- TestFX's assertions, Hamcrest matchers or AssertJ assertions
- Screenshots of failed tests
- Headless testing using Monocle

All the examples are built using JUnit5

Include TestFX in your project

TestFX

```
testCompileOnly("org.testfx:testfx-core:4.0.16-alpha")
testImplementation("org.testfx-junit5:4.0.16-alpha"
```

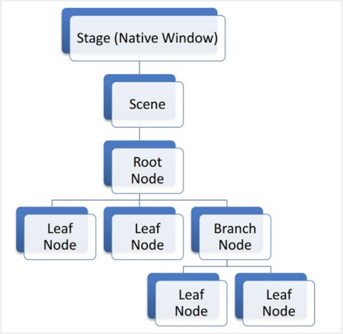
Monocle

```
testImplementation("org.testfx:openjfx-monocle:jdk-11+26"
```

JavaFX main points

SceneGraph

- Stage
- Scene
- RootNode
- BranchNode
- LeafNode



JavaFX main points

.fxml files and Controller

```
<BorderPane fx:id="panel" fx:controller:"com.application.controller.Controller>
    <Button fx:id="cancelButton" onAction="#onCancelButtonClicked"> </Button>
</BorderPane>

```

```
public class Controller{
    @FXML
    public BorderPane panel;

    @FXML
    public Button cancelButton;

    @FXML
    public void onCancelButtonClicked(ActionEvent actionEvent){
        System.exit(0);
        }
    }

```

TestFX test class's structure

```java
@ExtendWith(ApplicationExtension.class)
public class LoggerTest {

    // Will be called with {@code @Before} semantics, i. e. before each test method.
    @Start
    private void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(
            Controller.class.getResource("gui_layout.fxml")
            );
        Scene scene = new Scene(fxmlLoader.load());
        stage.setScene(scene);
        stage.show();
    }

    @Test void testMyGUI(FxRobot robot) {
        // WRITE TEST CODE HERE
    }

```

How to move inside your JavaFX SceneGraph

```
...
<TextField fx:id="usernameInput"></TextField>
...
<Button fx:id="confirmButton" onAction="#onConfirmButtonClicked"> </Button>
...
<Label
    text="USERNAME'S LENGTH MUST BE BETWEEN 4 AND 15"
    visible="false"
    fx:id="usernameErrorLabel">
</Label>

```

```
@DisplayName("Testing that username error label is not appearing with correct usernames")
@ParameterizedTest
@ValueSource(strings = {"user", "usern", "username", "usernameusernam"})
void usernameInputTestWithCorrectData(String username, FxRobot robot){
    robot.clickOn(robot.lookup("#usernameInput")
         .queryAs(TextField.class)).write(username);

    robot.clickOn(robot.lookup("#confirmButton")
         .queryButton());

    verifyThat("#usernameErrorLabel", (Label label) -> !label.isVisible());

}

```

How to assert with TestFX

From the documentation

All TestFX tests should use verifyThat(Node, Matcher, Function) when writing tests, so that the developer can use org.testfx.util.DebugUtils to provide additional info as to why a test failed.

```java
@DisplayName("Failing test that shows the screenshot feature")
@Test void exampleOfFailingTestWithScreenshot(FxRobot robot){
    verifyThat("#cancelButton",
               Node::isDisabled,
               saveNode(
                       robot.lookup("#passwordInput").queryAs(TextField.class),
                       SCREENSHOT_FAILING_TEST_PATH
               ));
    }

```

```java
@DisplayName("Failing test that show the screenshot feature")
@Test void exampleOfFailingTestWithScreenshot(FxRobot robot){
    assertThat(robot.lookup("#cancelButton").queryAs(Button.class).isDisabled()).
      isEqualTo(true);
    }
```

Headless testing with Monocle

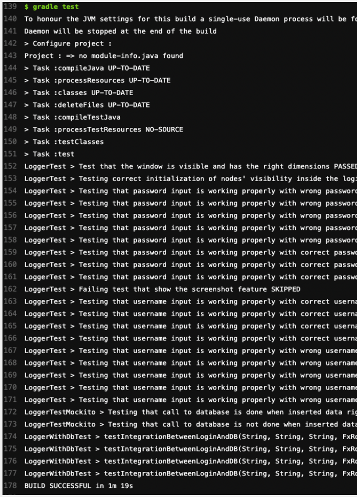
```
tasks.withType(Test){
    useJUnitPlatform()

    testLogging{
        events "passed", "skipped", "failed"
    }

    if (!project.hasProperty("noHeadless")) {
        jvmArgs "-Dheadless=true"
    }
}

```

```
@BeforeAll
public static void setupSpec() throws Exception {
    if (Boolean.getBoolean("headless")) {
        System.setProperty("testfx.robot", "glass");
        System.setProperty("testfx.headless", "true");
        System.setProperty("prism.order", "sw");
        System.setProperty("prism.text", "t2k");
        System.setProperty("java.awt.headless", "true");
    }
    registerPrimaryStage();
    }
```



```
$ gradle test
To honour the JVM settings for this build a single-use Daemon process will be fo
Daemon will be stopped at the end of the build
> Configure project :
Project : => no module-info.java found
> Task :compileJava UP-TO-DATE
> Task :processResources UP-TO-DATE
> Task :classes UP-TO-DATE
> Task :deleteFiles UP-TO-DATE
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses
> Task :test
LoggerTest > Test that the window is visible and has the right dimensions PASSED
LoggerTest > Testing correct initialization of nodes' visibility inside the logi
LoggerTest > Testing that password input is working properly with wrong passwor
LoggerTest > Testing that password input is working properly with wrong passwor
LoggerTest > Testing that password input is working properly with wrong passwor
LoggerTest > Testing that password input is working properly with wrong passwor
LoggerTest > Testing that password input is working properly with wrong passwor
LoggerTest > Testing that password input is working properly with correct passw
LoggerTest > Testing that password input is working properly with correct passw
LoggerTest > Testing that password input is working properly with correct passw
LoggerTest > Failing test that show the screenshot feature SKIPPED
LoggerTest > Testing that username input is working properly with correct userna
LoggerTest > Testing that username input is working properly with correct userna
LoggerTest > Testing that username input is working properly with correct userna
LoggerTest > Testing that username input is working properly with correct userna
LoggerTest > Testing that username input is working properly with wrong username
LoggerTest > Testing that username input is working properly with wrong username
LoggerTest > Testing that username input is working properly with wrong username
LoggerTest > Testing that username input is working properly with wrong username
LoggerTest > Testing that username input is working properly with wrong username
LoggerTestMockito > Testing that call to database is done when inserted data rig
LoggerTestMockito > Testing that call to database is not done when inserted data
LoggerWithDbTest > testIntegrationBetweenLoginAndDB(String, String, String, FxRo
LoggerWithDbTest > testIntegrationBetweenLoginAndDB(String, String, String, FxRo
LoggerWithDbTest > testIntegrationBetweenLoginAndDB(String, String, String, FxRo
LoggerWithDbTest > testIntegrationBetweenLoginAndDB(String, String, String, FxRo
BUILD SUCCESSFUL in 1m 19s
```

Useful links

- https://github.com/TestFX/TestFX
- https://testfx.github.io/TestFX/docs/javadoc/testfx-core/javadoc/org.testfx/module-summary.html
- https://gitlab.com/lorenzocim/logger
- https://openjfx.io/openjfx-docs/
- https://martinfowler.com/articles/practical-test-pyramid.html#TheTestPyramid

Coding