

Title: Test cases & Logic

Author: Nam Cap

Student ID: a1907345

Writing test cases for a GUI that relies on FRP (Functional Reactive Programming) requires a different approach compared to traditional imperative programming. We need to simulate the FRP streams and check if the GUI reacts as expected.

The screenshot I provided indicates that the GPS Tracker GUI is displaying tracker data with latitude, longitude, and distance measurements. This suggests that the application is functioning in terms of displaying real-time data for each tracker. The distance fields all show "0 meters," which may be correct if the trackers have not moved since the application was started or if the application has just been initialized.

1. Setup

Testing setup attempts to create or interact with UI components (JFrame, JPanel, etc.), which is not possible in a headless environment. To address this, I need to modify your tests to either avoid creating actual GUI components or to use a headless-friendly approach.

Here are a few strategies I used:

Mock GUI Components: Instead of creating actual JFrame or JPanel objects, I mocked these objects. This is useful because the objects that behave like GUI components without actually displaying anything.

Check for Headless Environment: In my test setup, check if the environment is headless and skip the GUI-dependent parts of the tests if it is. I can use `GraphicsEnvironment.isHeadless()` to check this.

Headless Property in Tests: When running tests, set the `java.awt.headless` property to true. This tells the Java runtime to not expect a display, mouse, or keyboard.

```
@Before
public void setUp() {
    isHeadless =
GraphicsEnvironment.isHeadless();
    if (!isHeadless) {
        SwingUtilities.invokeLater(() -> {
            gpsGUI = new GpsGUI();
            gpsGUI.initializeComponents();
        });
    } else {
```

```

        // Setup for headless mode
        System.setProperty("java.awt.headless",
"true");

        gpsGUI = new GpsGUI(); // Create a
GpsGUI instance without GUI initialization
        gpsGUI.mockComponents(); // Make sure
this method does not instantiate actual GUI
components
    }
}

```

2. runTest()

The runTest method runs the test logic either directly (in headless mode) or within the Swing event dispatch thread (in GUI mode).

```

private void runTest(Runnable testLogic) {
    if (isHeadless) {
        // Run test logic without involving GUI in
headless mode
        testLogic.run();
    } else {
        // Run test logic normally in GUI mode
        try {
            SwingUtilities.invokeAndWait(testLogic
);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

3. runInAppropriateEnvironment ()

- runInAppropriateEnvironment handles the environment check and setup.
- Each test method uses runInAppropriateEnvironment to ensure the correct environment setup.

```

    private void runInAppropriateEnvironment(Runnable
testLogic) {
        if (GraphicsEnvironment.isHeadless()) {
            System.setProperty("java.awt.headless",
"true");

            GpsGUI.setTestMode(true);
            gpsGUI.mockComponents(); // Assuming
mockComponents method exists in GpsGUI
        } else {
            GpsGUI.setTestMode(false);
            // Setup for non-headless environment
        }

        testLogic.run();
    }

```

4. testDistanceCalculationAndUpdate()

In the testTrackerDisplayUpdate test, I simulate the tracker data for each tracker and invoke the GUI update method. This method (updateTrackerDisplay) should be responsible for updating the tracker labels with the data contained in the Cell<GpsEvent>. This method needs to be implemented in your GpsGUI class.

```

@Test
    public void testDistanceCalculationAndUpdate() {
        runTest(() -> {
            String trackerId = "Tracker1";
            GpsGUI.GpsEvent startEvent = new
GpsGUI.GpsEvent(trackerId, 50.0, 10.0, 100.0);
            GpsGUI.GpsEvent endEvent = new
GpsGUI.GpsEvent(trackerId, 50.001, 10.001, 100.0);

            double distance =
GpsGUI.calculateDistance(startEvent, endEvent);

```

```

        gpsGUI.updateTrackerDistanceDisplay(trackerId, distance);

        JLabel trackerLabel =
gpsGUI.getTrackerLabel(trackerId);
        String expectedDisplay =
String.format("%s: Total Distance: %.2f meters",
trackerId, distance);
        assertEquals(expectedDisplay,
trackerLabel.getText());
    });
}

```

5. testGuiInitialization()

This test method ensures that the main frame, tracker panel, input panel, and potentially other components of my GpsGUI class are correctly instantiated when the application starts.

```

@Test
public void testGuiInitialization() {
    if (!isHeadless) {
        runTest(() ->{
            runInAppropriateEnvironment(() -> {
                assertNotNull("Frame should not be
null", gpsGUI.getFrame());
                assertNotNull("Tracker panel should
not be null", gpsGUI.getTrackerPanel());
                assertNotNull("Input panel should not
be null", gpsGUI.getInputPanel());
            });
        });
    }
}

```

6. testTrackerDisplays()

testTrackerDisplays tests whether the GUI correctly updates the display for each tracker when a new GPS event occurs. This involves simulating GPS events, updating the GUI, and verifying that the GUI reflects these changes accurately. The test only runs when a GUI is available (not in headless mode).

```
@Test
public void testTrackerDisplays() {
    if (!isHeadless) {
        runTest(() -> {
            for (int i = 1; i <= 10; i++) {
                String trackerId = "Tracker" + i;
                JLabel initialLabel =
gpsGUI.getTrackerLabel(trackerId);
                assertNotNull("Label for tracker
should not be null", initialLabel);
                assertEquals("Initial text should
show 0 meters", trackerId + ": Distance 0 meters",
initialLabel.getText());

                GpsGUI.GpsEvent simulatedEvent =
new GpsGUI.GpsEvent(trackerId, 50.0 + i, 10.0 + i,
100.0 + i);
                processGpsEventInSwingThread(simul
atedEvent);

                JLabel updatedLabel =
gpsGUI.getTrackerLabel(trackerId);
                String expectedDisplay =
String.format("Tracker%s: Lat %.1f, Lon %.1f",
trackerId, 50.0 + i, 10.0 + i);
                assertEquals("Tracker display
should be updated with simulated event data",
expectedDisplay, updatedLabel.getText());
            }
        });
    }
}
```

```
}
```

7. processGpsEventInSwingThread()

In my testing framework, this method is used to ensure that the GPS events are processed in the correct thread (EDT) and that the GUI behaves as expected when these events occur. It helps in validating the thread safety and correctness of GUI updates in response to external events, which is crucial in Swing applications.

```
private void
processGpsEventInSwingThread(GpsGUI.GpsEvent event) {
    try {
        SwingUtilities.invokeLaterAndWait(() ->
gpsGUI.processGpsEvent(event));
    } catch (InterruptedException |
InvocationTargetException e) {
        e.printStackTrace();
        // Optionally, handle or rethrow as a
runtime exception
    }
}
```

8. testTrackerDisplayUpdate()

This test verifies that the GUI correctly updates the display labels for each tracker when new GPS event data is received. It is crucial for ensuring that the application accurately reflects changes in GPS data and that the GUI behaves as expected in response to these changes.

```
@Test
public void testTrackerDisplayUpdate() {
    if (!isHeadless) {
        runTest(() -> {
            for (int i = 1; i <= 10; i++) {
                String trackerId = "Tracker" + i;
                // Directly create a GpsEvent with
deterministic data.
            }
        })
    }
}
```

```

        GpsGUI.GpsEvent simulatedEvent = new
GpsGUI.GpsEvent(trackerId, 50.0 + i, 10.0 + i, 100.0 +
i);

        // Invoke the method that updates the
GUI with the simulated event.
        // This method needs to be implemented
in GpsGUI.
        gpsGUI.updateTrackerDisplay(simulatedE
vent);

        // Retrieve the label for this
tracker.
        JLabel trackerLabel =
gpsGUI.getTrackerLabel(trackerId);

        // Check if the tracker label was
updated with the simulated data.
        // The format here should match the
format used in updateTrackerDisplay.
        String expectedText =
String.format("Tracker%s: Lat %.1f, Lon %.1f, Alt %.1f
meters",
                trackerId,
simulatedEvent.getLatitude(),
simulatedEvent.getLongitude(),
                simulatedEvent.getAltitude());
        assertEquals("Tracker label should
display the correct coordinates.",
                expectedText,
trackerLabel.getText());
    }
});
}

```

```
}
```

9. tearDown()

This structure allows each test to run in a clean state, which is important for the reliability and independence of your tests.

```
@After
public void tearDown() {
    // Clean up the FRP environment if needed
    GpsGUI.cleanup(); // Implement cleanup method
in GpsGUI if necessary
}
```

10. How can run my code:

The Makefile provided by 'makefile' and includes specific classpath settings for external libraries (in this case, sodium.jar and swidgets.jar). This Makefile compiles all Java files in the current directory and cleans up the generated .class files.

Type: **'make'** to run the project

```
JCC = javac
JFLAGS = -g
# CLASSPATH =
/path/to/sodium/library/sodium.jar:/path/to/sodium/lib
rary/swidgets.jar.
CLASSPATH =
"C:\\Users\\kxeam\\Downloads\\Compressed\\W09\\sodium.
jar;C:\\Users\\kxeam\\Downloads\\Compressed\\W09\\swid
gets.jar;."

default: all

all:
    $(JCC) $(JFLAGS) -cp $(CLASSPATH) *.java

clean:
```




```
$(RM) *.class
```

run:

```
java -cp $(CLASSPATH) GpsGUI
```

Here is my output:

Here is the output GUI:

 GPS Tracker

—

□

×

Trackers

Tracker Tracker 1: Lat -36.2904798, Lon 172.14323049
Tracker 1: Distance 0 meters
Tracker Tracker 2: Lat 30.56626496, Lon -94.70217804
Tracker 2: Distance 0 meters
Tracker Tracker 3: Lat -54.23002792, Lon 3.93719041
Tracker 3: Distance 0 meters
Tracker Tracker 4: Lat -76.44505001, Lon 125.89120949
Tracker 4: Distance 0 meters
Tracker Tracker 5: Lat 48.21497853, Lon -55.98991149
Tracker 5: Distance 0 meters
Tracker Tracker 6: Lat 64.81795864, Lon -48.36600069
Tracker 6: Distance 0 meters
Tracker Tracker 7: Lat -18.50360966, Lon 177.26439715
Tracker 7: Distance 0 meters
Tracker Tracker 8: Lat -85.86164791, Lon -40.64002044
Tracker 8: Distance 0 meters
Tracker Tracker 9: Lat 30.20287355, Lon -70.52784977
Tracker 9: Distance 0 meters
Tracker Tracker 10: Lat 16.50451939, Lon 93.44687456
Tracker 10: Distance 0 meters

Filter Controls

Latitude:

Longitude:

Apply Filter

Current filter: Invalid

Combined Data

Here is the output in terminal with all Tracker 1 to 10 moving:

```
Generated event for Tracker4: Tracker ID: Tracker4, Latitude: 57.904203, Longitude: -170.056714, Altitude: 251.70
Generated event for Tracker5: Tracker ID: Tracker5, Latitude: 43.020479, Longitude: -174.050836, Altitude: 896.39
Generated event for Tracker6: Tracker ID: Tracker6, Latitude: -48.487824, Longitude: 178.117344, Altitude: 683.07
Generated event for Tracker7: Tracker ID: Tracker7, Latitude: 11.889059, Longitude: 149.178748, Altitude: 21.14
Generated event for Tracker8: Tracker ID: Tracker8, Latitude: -39.021627, Longitude: 67.588293, Altitude: 540.37
Generated event for Tracker9: Tracker ID: Tracker9, Latitude: 76.683967, Longitude: 61.516604, Altitude: 841.48
Generated event for Tracker10: Tracker ID: Tracker10, Latitude: 65.269925, Longitude: -2.662728, Altitude: 900.09
Generated event for Tracker1: Tracker ID: Tracker1, Latitude: 34.585276, Longitude: 26.190643, Altitude: 580.52
Generated event for Tracker3: Tracker ID: Tracker3, Latitude: -59.673825, Longitude: 97.259205, Altitude: 968.67
Generated event for Tracker4: Tracker ID: Tracker4, Latitude: -16.010775, Longitude: 120.612690, Altitude: 313.78
Generated event for Tracker2: Tracker ID: Tracker2, Latitude: -9.018792, Longitude: -160.643395, Altitude: 392.30
Generated event for Tracker6: Tracker ID: Tracker6, Latitude: 38.097609, Longitude: 118.496333, Altitude: 116.16
Generated event for Tracker7: Tracker ID: Tracker7, Latitude: -81.352713, Longitude: -37.830491, Altitude: 987.63
Generated event for Tracker8: Tracker ID: Tracker8, Latitude: -27.777352, Longitude: -141.376260, Altitude: 778.19
Generated event for Tracker5: Tracker ID: Tracker5, Latitude: -30.909474, Longitude: 178.434131, Altitude: 622.38
Generated event for Tracker9: Tracker ID: Tracker9, Latitude: 66.290359, Longitude: -167.129815, Altitude: 943.19
Generated event for Tracker10: Tracker ID: Tracker10, Latitude: 9.245243, Longitude: 64.623423, Altitude: 703.69
Generated event for Tracker1: Tracker ID: Tracker1, Latitude: -28.449715, Longitude: -40.780656, Altitude: 342.71
Generated event for Tracker2: Tracker ID: Tracker2, Latitude: -82.864610, Longitude: 128.712246, Altitude: 442.89
Generated event for Tracker3: Tracker ID: Tracker3, Latitude: 40.420865, Longitude: -23.471211, Altitude: 254.47
Generated event for Tracker4: Tracker ID: Tracker4, Latitude: 58.776306, Longitude: 3.186291, Altitude: 376.88
Generated event for Tracker5: Tracker ID: Tracker5, Latitude: -4.089575, Longitude: 5.847110, Altitude: 277.06
Generated event for Tracker6: Tracker ID: Tracker6, Latitude: 62.704278, Longitude: -138.864998, Altitude: 149.67
Generated event for Tracker7: Tracker ID: Tracker7, Latitude: 86.892732, Longitude: 84.312293, Altitude: 29.20
Generated event for Tracker8: Tracker ID: Tracker8, Latitude: 45.725353, Longitude: -160.375064, Altitude: 981.69
Generated event for Tracker9: Tracker ID: Tracker9, Latitude: -68.288944, Longitude: -161.056617, Altitude: 254.09
Generated event for Tracker10: Tracker ID: Tracker10, Latitude: 64.506735, Longitude: -132.581331, Altitude: 17.21
```