EE360T/EE382V: Software Testing
Optional Problem Set

Out: Dec 3, 2022; **Due: Dec 10, 2022 11:59pm**
Submission: *.zip via Canvas
Maximum points: 40

In this homework you will use the Selenium browser automation tool (`http://www.seleniumhq.org/`) to automate testing of a simple website. You can download and install Selenium in 2 simple steps:

1. Download the `*.jar` file for the "Selenium Standalone Server" from the Selenium project website: `http://www.seleniumhq.org/download/`
2. Add the `*.jar` file as an external jar for your project on Eclipse (or other IDE). For example, in Eclipse, set it under: "Project/Properties/Java Build Path/Libraries/Add External JARs..."

1 Testing a basic website [20 points]

Overview. The following HTML code with JavaScript defines a webpage for calculating the minimum of three numbers[1]:

```
<!DOCTYPE html>
<html>
  <head>
    <meta name="Course" content="UT EE360T (Spring 2020)">
    <title>Min Calculator</title>
    <script>
      function compute() {
        var x = document.getElementById("x").value;
        var y = document.getElementById("y").value;
        var z = document.getElementById("z").value;
        if (x == "" || isNaN(x) || y == "" || isNaN(y) || z == "" || isNaN(z)) {
          document.getElementById("result").textContent = "Please enter integer values only!";
          return;
        }
        var result = minimum(x, y, z);
        document.getElementById("result").textContent =
           "min(" + x + ", " + y + ", " + z + ") = " + result;
      }
      function minimum(x, y, z) {
        return Math.min(Math.min(x, y), Math.min(x, z));
      }
    </script>
  </head>
  <body>
    <h1>Min Calculator</h1>
    x: <input id="x" placeholder="Enter integer value for x"><br>
    y: <input id="y" placeholder="Enter integer value for y"><br>
    z: <input id="z" placeholder="Enter integer value for z"><br><br>
    <input type="button" id="computeButton" value="Calculate min(x, y, z)" onclick="compute()">
    <br>
    <h2 id="result"></h2>
  </body>
</html>
```

[1] The file `min.html` on Canvas contains this code.

In this part of the homework you will be implementing a program that generates a test suite T for the core functionality of the `min.html` website such that T provides combinatorial coverage with respect to (w.r.t.) the following (simplistic) input domain model (IDM) M:

- There are four inputs: x, y, z, *computeButton*;
- Each of the three inputs x, y, and z have the following input space partition with 4 blocks and their representative values (shown in square brackets):
    1. *not-an-integer* ["infinity"];
    2. *integer* $<$ *0* $[-3]$;
    3. *integer* $==$ *0* $[0]$;
    4. *integer* $>$ *0* $[7]$;
- The only input using the compute button is a *mouse-click event*; thus, conceptually you can view this input space to have two values *click* and *notclick*, which form a partition.

For this IDM there exist a total of $4 \cdot 4 \cdot 4 \cdot 2 = 128$ tests since there are 4 possible values for each of x, y, and z, and 2 possible values for *computeButton*.

Example. Consider an example test t given by the tuple $\langle x = 0, y = 0, z = 0, \mathit{computeButton} = \mathit{click}\rangle$. The following Java method `t0` in class `MinWebTest`[2] shows how to write a JUnit test that represents t for testing the `min.html` page:

```
package pset6;

import static org.junit.Assert.*;

import org.junit.Test;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;

public class MinWebTest {
    @Test public void t0() {
        // execute the test <x = 0, y = 0, z = 0, submitButton = click> and check the output message is correct
        WebDriver wd = new FirefoxDriver(); // launch the browser
        // edit the next line to enter the location of "min.html" on your file system
        wd.get("file:///C:/Users/.../min.html");
        WebElement we = wd.findElement(By.id("x"));
        we.sendKeys("0"); // enter 0 for x
        we = wd.findElement(By.id("y"));
        we.sendKeys("0"); // enter 0 for y
        we = wd.findElement(By.id("z"));
        we.sendKeys("0"); // enter 0 for z
        we = wd.findElement(By.id("computeButton"));
        we.click();
        WebElement result = wd.findElement(By.id("result"));
        String output = result.getText(); // read the output text
        assertEquals("min(0, 0, 0) = 0", output);
        wd.quit(); // close the browser window
    }
}
```

Your task. Your specific task in this part of the homework is to implement a test case generator `MinWebTestGenerator` in Java for testing `min.html` webpage such that:

[2] The file `MinWebTest.java` on Canvas contains this code.

1. The console output of `MinWebTestGenerator.main` method is a Java program `MinWebTestSuite` that represents the test suite T that provides combinatorial coverage w.r.t. IDM M (as defined above); thus, the output must include 128 test methods;

2. The test suite `MinWebTestSuite` launches the web browser **exactly once** for the *entire* execution of all the tests[3];

3. Each test in `MinWebTestSuite` is a valid JUnit test, which (1) (re-)loads the webpage before entering any input; (2) enters an appropriate input into the webpage, and (3) invokes a test assertion that checks the output message on the webpage for an exact match (up to `equals` method)[4]; and

4. The output program `MinWebTestSuite` can be compiled and run as a standalone Java program assuming required Selenium and JUnit libraries are correctly included in the project settings.

The following code gives a skeletal `MinWebTestGenerator`[5] implementation, which you must build on:

```
package pset6;

public class MinWebTestGenerator {
    public static void main(String[] a) {
        String suite = new MinWebTestGenerator().createTestSuite();
        System.out.println(suite);
    }

    String createTestSuite() {
        StringBuilder sb = new StringBuilder();
        sb.append(packageDecl());
        sb.append("\n");
        sb.append(imports());
        sb.append("\n");
        sb.append(testsuite());
        return sb.toString();
    }

    String packageDecl() {
        return "package pset6;\n";
    }

    String imports() {
        return "import static org.junit.Assert.*;\n\n"
                + "import org.junit.Test;\n\n"
                + "import org.openqa.selenium.By;\n"
                + "import org.openqa.selenium.WebDriver;\n"
                + "import org.openqa.selenium.WebElement;\n"
                + "import org.openqa.selenium.firefox.FirefoxDriver;\n";
    }

    String testsuite() {
        StringBuilder sb = new StringBuilder();
        sb.append("public class MinWebTestSuite {\n");

        // your code goes here
        // ...

        sb.append("}\n");
        return sb.toString();
    }

    // implement any helper methods that you need in this class
}
```

[3] Read about *test fixtures*, e.g., @`BeforeClass`, in JUnit: `https://github.com/junit-team/junit/wiki/Test-fixtures`

[4] If the input does not click the *computeButton*, no test assertion is needed.

[5] The file `MinWebTestGenerator.java` on Canvas contains this code.

Files to submit. For this part of the homework please submit the following 2 Java files:

1. Your complete `MinWebTestGenerator.java` file that builds on the given skeleton; and
2. A new `MinWebTestSuite.java` file that contains the console output for running your implementation of `MinWebTestGenerator`.

Needless to say, you must make sure both your files compile and the compiled programs run!

2 Regression testing of a basic website [20 points total]

Consider now adding new functionality to `min.html` from Question 1. Specifically, consider the following HTML with JavaScript code `minandmax.html`[6] that adds two *radio buttons* to `min.html`, so the user can choose whether to compute minimum or maximum (which is the default choice as indicated by the `checked="checked"` setting) of the three input integers:

```
<!DOCTYPE html>
<html>
  <head>
    <meta name="Course" content="UT EE360T (Spring 2020)">
    <title>Min and Max Calculator</title>
    <script>
      function minSelected() {
        document.getElementById("computeButton").value = "Calculate min(x, y, z)";
      }
      function maxSelected() {
        document.getElementById("computeButton").value = "Calculate max(x, y, z)";
      }
      function compute() {
        var x = document.getElementById("x").value;
        var y = document.getElementById("y").value;
        var z = document.getElementById("z").value;
        if (x == "" || isNaN(x) || y == "" || isNaN(y) || z == "" || isNaN(z)) {
          document.getElementById("result").textContent = "Please enter integer values only!";
          return;
        }
        var minormax = document.getElementById("min").checked;
        var result = (minormax) ? minimum(x, y, z) : maximum(x, y, z);
        document.getElementById("result").textContent =
           ((minormax) ? "min" : "max") + "(" + x + ", " + y + ", " + z + ") = " + result;
      }
      function minimum(x, y, z) {
        return Math.min(Math.min(x, y), Math.min(x, z));
      }
      function maximum(x, y, z) {
        return Math.max(Math.max(x, y), Math.max(x, z));
      }
    </script>
  </head>
  <body>
    <h1>Min and Max Calculator</h1>
    <input type="radio" name="func" id="min" onclick="minSelected()">Min
    <input type="radio" name="func" id="max" checked="checked" onclick="maxSelected()">Max
    <br><br>
    x: <input id="x" placeholder="Enter integer value for x"><br>
    y: <input id="y" placeholder="Enter integer value for y"><br>
    z: <input id="z" placeholder="Enter integer value for z"><br><br>
    <input type="button" id="computeButton" value="Calculate min(x, y, z)" onclick="compute()">
    <br>
    <h2 id="result"></h2>
  </body>
</html>
```

[6] The file `minandmax.html` on Canvas contains this code.

2.1 Regression failure? [5 points]

Consider running the test suite you generated using your implementation of `MinWebTestGenerator` in Question 1 of this homework against the webpage `minandmax.html`. As clearly identified comments in your `MinWebTestSuite.java` file that you submit as part of your solution to Question 1, report the number of tests that fail if you replace the code in the file `min.html` with the code in the `minandmax.html` file, i.e., you run your old tests against the new modified webpage. Additionally, report also as clearly identified comments in the same file whether these failures are bugs in `minandmax.html` page or faulty tests.

2.2 New test generator [15 points]

Consider once again the IDM M from Question 1. Augment M to M' to account for the new group of 2 radio buttons with id's `min` and `max` such that exactly one of the buttons is always selected. Conceptually, the input space for this group is partitioned by two values: (1) `min` is selected; and (2) `max` is selected. Implement a new test generator `MinAndMaxWebTestGenerator`, which provides combinatorial coverage w.r.t. the augmented IDM M' for testing `minandmax.html`; for reference, the output of `MinAndMaxWebTestGenerator` must include $128 \cdot 2 = 256$ tests.
Files to submit. For this part of the homework, please submit the following two Java files:

1. Your `MinAndMaxWebTestGenerator.java` file; and
2. A new `MinAndMaxWebTestSuite.java` file that contains the console output for running your implementation of `MinAndMaxWebTestGenerator`.

Once again, needless to say, you must make sure both your files compile and the compiled programs run!