Assignment I2: JUnit and SpotBugs

Published Edit ⋮

This is an individual assignment. This assignment will be graded as 0 to max points.   Scroll down for submission instructions at the end.

Assignment 2 builds on Assignment 1. Please make sure you have completed Assignment 1 before starting this assignment. Libraries, services, and frameworks were covered in the class earlier (22nd Sept 2020). Now, suppose you want to provide the tic-tac-toe application as a service. You will need to expose the APIs to your clients. Your clients might want to integrate your game in their own app, or they want to build their own UI and use your APIs to play the game.

This assignment focuses on testing your endpoints so that you can provide a good user experience to your clients. You will perform unit tests and integration tests (API tests) on your application to identify and fix potential bugs.

1. Setup instructions

This assignment builds on the previous assignment. If you have not already completed assignment 1, then please do so before starting assignment 2.

Please follow the instruction below to get started with this assignment:

1. Install the Spotbugs plugin from Eclipse Marketplace. Run Eclipse, then **Help** -> **Eclipse Marketplace**. Use the search box to find 'Spotbugs'. You will find the plugin named "SpotBugs Eclipse Plugin", then click **Install**. Eclipse will prompt you to restart it after the update. You can check that Spotbugs has been installed: Right click on the project and look for **"Spotbugs"** on the list.
2. Install the Emma plugin from Eclipse Marketplace. Run Eclipse, then **Help** -> **Eclipse Marketplace**. Use the search box to find 'Emma'. You will find the plugin named "EclEmma Java Code Coverage", then click **Install**. Eclipse will prompt you to restart it after the update. You can check that Emma has been installed: Right click on the project and look for **"Coverage As"** on the list.
3. To add a test folder, that contains all the test cases, to your project Right Click on the project and select "Properties". Under "Java Build Path" -> "Source" -> "Add Folder..." In the dialog box check "test" -> "java" and select OK. Then select "Apply and close". You will be able to see a new directory under your project "src/test/java"

2. Tasks

Course Chat

You are required to complete the following programming tas

1. Write **JUnit 5** **(https://junit.org/junit5/)** test cases for th
2. Use Emma to check code coverage of the tests.
3. Use SpotBugs to identify potential bugs in your code anc
4. Perform style checks (Checkstyle), as in Assignment 1.

The game will support only one game at a time, i.e. only twc
on completing the game using proper programming standarc
things.

choose which
version of Java
in Eclipse

SC

Sang Jun Chun 9/15/20
10:44 AM

Eclipse ->
preferences ->
java -> installed
JREs

MG

Minxuan Gao 9/15/20
10:45 AM

Maven is a build
automation tool

Send

2.1. Testing with JUnit

JUnit is a testing framework for Java programming language
(https://www.tutorialspoint.com/junit/junit_overview.htm). You
(https://howtodoinjava.com/junit-5-tutorial/).

JUnit offers the following annotations to write tests:

Annotation	Descrip
`@BeforeEach`	The annotated method will be run before ea
`@AfterEach`	The annotated method will be run after each test method in the test class.
`@BeforeAll`	The annotated method will be run before all test methods in the test class. This method must be static.
`@AfterAll`	The annotated method will be run after all test methods in the test class. This method must be static.
`@Test`	It is used to mark a method as JUnit test.
`@Order`	It is used to configure the order in which the annotated method should be executed.

2.1.1 Unit Tests

Let us suppose we have a Calculator class with two methods: **add** and **division**:

```
public class Calculator {

  /**
```

```
   * Returns sum of two integers
   * @param a First number to add
   * @param b Second number to add
   * @return Sum of a and b
   */
  public int add(int a, int b) {
    return a + b;
  }

  /**
   * Returns division of two integers. If denominator is zero, method returns NaN.
   * @param numerator
   * @param denominator
   * @return Result of division
   */
  public double division(int numerator, int denominator) {

    double result = 0.0;

    if(denominator != 0) {
      result = numerator/denominator;
    }else {
      result = Double.NaN;
    }

    return result;
  }

}
```

Now, we want to test the **add** and **division** methods of Calculator class. Using JUnit, the test script would look something like the following:

```
public class CalculatorTest {

  Calculator cal = new Calculator();

  @Test
  public void testAdd() {

    int num1 = 5;
    int num2 = 5;

    int sum = cal.add(num1, num2);

    assertEquals(10, sum);
  }

  @Test
  public void testDivNonZeroDenominator() {

    int num1 = 5;
    int num2 = 5;

    double division = cal.division(num1, num2);

    assertEquals(1, division);
  }

  @Test
  public void testDivZeroDenominator() {

    int num1 = 5;
    int num2 = 0;
    double division = cal.division(num1, num2);

    assertEquals(Double.NaN, division);
  }
```

Observe that we have two test cases for the **division** method. These test cases correspond to the if-else conditions in the method. We want to have several tests for a method to test as much code as possible. This is called code coverage. **Code coverage** is a metric that can help **you** understand how much of your source is tested. In this assignment, we will use the Emma Eclipse plugin to calculate code coverage.

2.1.3. API Tests

Now, suppose we want to test the tic-tac-toe service. We will have to test the exposed endpoints to evaluate the application. In this assignment, we will be using Unirest-Java (http://kong.github.io/unirest-java/) to make API calls. Here is a sample code snippet that shows how to test your endpoints:

```
@TestMethodOrder(OrderAnnotation.class)
public class GameTest {

    /**
    * Runs only once before the testing starts.
    */
    @BeforeAll
        public static void init() {
                // Start Server
        PlayGame.main(null);
        System.out.println("Before All");
    }

    /**
    * This method starts a new game before every test run. It will run every time before a test.
    */
    @BeforeEach
    public void startNewGame() {
        // Test if server is running. You need to have an endpoint /
        // If you do not wish to have this end point, it is okay to not have anything in this method.
        HttpResponse response = Unirest.get("http://localhost:8080/").asString();
        int restStatus = response.getStatus();

        System.out.println("Before Each");
    }

    /**
    * This is a test case to evaluate the newgame endpoint.
    */
    @Test
    @Order(1)
    public void newGameTest() {

        // Create HTTP request and get response
        HttpResponse response = Unirest.get("http://localhost:8080/newgame").asString();
        int restStatus = response.getStatus();

        // Check assert statement (New Game has started)
        assertEquals(restStatus, 200);
        System.out.println("Test New Game");
    }

    /**
    * This is a test case to evaluate the startgame endpoint.
    */
    @Test
    @Order(2)
    public void startGameTest() {

        // Create a POST request to startgame endpoint and get the body
        // Remember to use asString() only once for an endpoint call. Every time you call asString(), a
new request will be sent to the endpoint. Call it once and then use the data in the object.
        HttpResponse response = Unirest.post("http://localhost:8080/startgame").body("type=X").asString
();
        String responseBody = response.getBody();

        // --------------------------- JSONObject Parsing ----------------------------------

        System.out.println("Start Game Response: " + responseBody);

        // Parse the response to JSON object
        JSONObject jsonObject = new JSONObject(responseBody);

        // Check if game started after player 1 joins: Game should not start at this point
        assertEquals(false, jsonObject.get("gameStarted"));

        // ---------------------------- GSON Parsing -------------------------

        // GSON use to parse data to object
        Gson gson = new Gson();
        GameBoard gameBoard = gson.fromJson(jsonObject.toString(), GameBoard.class);
        Player player1 = gameBoard.getP1();

        // Check if player type is correct
        assertEquals('X', player1.getType());

        System.out.println("Test Start Game");
    }

    /**
    * This will run every time after a test has finished.
    */
    @AfterEach
    public void finishGame() {
        System.out.println("After Each");
    }

    /**
     * This method runs only once after all the test cases have been executed.
     */
    @AfterAll
    public static void close() {
        // Stop Server
        PlayGame.stop();
        System.out.println("After All");
    }
}
```

2.1.3. Running JUnit Tests

You can run tests through maven. Right click on the project, **Run as** -> **Maven test**

The console will print the final results:

```
Running GameTest

Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.083 sec - in GameTest

Running CalculatorTest

Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.008 sec - in CalculatorTest
```

2.2. SpotBugs

To get started with SpotBugs, right-click on a Java project in Package Explorer, and select the option labeled “Spot Bugs”. SpotBugs will run, and problem markers (displayed in source windows, and also in the Eclipse Problems view) will point to locations in your code, which have been identified as potential instances of bug patterns. Alternatively, you can run it on individual files.

Note: You do not have to worry about the SpotBugs warning in the JUnit tests. Only consider bugs in your tic-tac-toe application.

2.2.1 Generating Spotbugs Report

To generate a Spotbugs report, right-click on pom.xml file, **Run as** -> **Build...** A pop-up window will appear. In the **Goals** add **'site'** and click **Run**.

Right-click on the project, and then click **Refresh**. Inside the **target** -> **site** directory, you will find *spotbugs.html.* Open the HTML file in a browser to read the report.

If you need other customization, follow the link **here (https://spotbugs.readthedocs.io/en/stable/eclipse.html)** .

2.3. Emma

To get started with Emma, right-click on a Java project in Package Explorer, and select the option labeled “Coverage As” -> "JUnit Test". Emma will run the tests, and coverage (displayed in source windows, and also in the Eclipse Problems view) will show lines of your application code, which have been executed by the tests.

3. Grading Scheme

We will evaluate your submission on three criteria, as follows:

1. JUnit tests
2. Check Style compliance
3. Spotbugs compliance

(https://github.com/Programming-Systems-Lab/4156-PublicAssignmentBackground/blob/master/Part-2/README.md#51-

junit-tests) 3.1 JUnit Tests

You must write tests for each method in your application. Tests must cover all conditional branches within a method. We have five test cases (From assignment I1) to validate the tic-tac-toe game. Your game must pass these tests as described below:

1. A player cannot make a move until both players have joined the game.
2. After game has started Player 1 always makes the first move.
3. A player cannot make two moves in their turn.
4. A player should be able to win a game.
5. A game should be a draw if all the positions are exhausted and no one has won.

For API testing, you can no longer rely on the UI JavaScript logic. In order to pass the above checks, you will have to add logic in the back-end application. Also note that the above cases involve testing the move endpoint. In the absence of UI, **Postman** (https://www.postman.com/) is your friend. We expect to see one test case for every branch in your method/endpoint.

You do not have to test the other endpoints but you will need to call them. Say, for example, the above tests do not involve the tests for: "What happens if joingame endpoint is called before startgame endpoint?" Also, you do not have to create tests for getters and setters.

Hint: You might want to create an endpoint to get the gameboard. Alternatively, you can create a WebSocket client and use that to get the gameboard. Since some students wanted to learn about WebSockets, we leave this decision to you.

(https://github.com/Programming-Systems-Lab/4156-PublicAssignmentBackground/blob/master/Part-2/README.md#52-check-style-compliance) 3.2 Checkstyle Compliance

In this assignment, we will follow **Google Java Style** (https://google.github.io/styleguide/javaguide.html) . Please go through the document to understand the recommended style and make the necessary changes to your application's code.

(https://github.com/Programming-Systems-Lab/4156-PublicAssignmentBackground/blob/master/Part-2/README.md#53-spotbugs-compliance) 3.3 Spotbugs Compliance

We require that you resolve all issues identified by Spotbugs.

You can ask for help with any of the technologies involved in this assignment by posting questions or problems in this course's individual_project folder on **piazza,** (https://courseworks2.columbia.edu/courses/104335/external_tools/1456) on relevant community forums, on stackoverflow, etc. However, do not ask for help writing the code, this is a violation of the university's academic honesty policy.

Submission instructions: Submit a zip file containing hw1 directory. You may resubmit repeatedly until the deadline. If you were unable to complete this assignment for any reason, submit anyway. In that case you should include a text explaining how far you got and what went wrong.

Points	20
Submitting	a file upload
File Types	zip

Due	For	Available from	Until
Oct 1, 2020	Everyone	Sep 24, 2020 at 12:01am	Oct 8, 2020 at 11:59pm

Assignment 2 Rubric

Criteria	Ratings		Pts
Spotbugs Your application must pass Spotbugs check. We will deduct one point for every Spotbugs warning or error in the tic-tac-toe application.	**4 to >0.0 pts** **Full Marks**	**0 pts** **No Marks**	4 pts
JUnit Tests Coverage Your test cases must cover all methods and conditional branches within methods. We will deduct 0.5 points for every missed method or conditional branch. Each branch/method/endpoint must have its own test. Testing multiple elements in one test method will result in a deduction of 1 point per test method.	**10 to >0.0 pts** **Full Marks**	**0 pts** **No Marks**	10 pts
CheckStyle Compliance 1 point deduction per 2 errors (other than indentation and tabs)	**1 to >0.0 pts** **Full Marks**	**0 pts** **No Marks**	1 pts
Passing all test cases We have created 5 test cases that tests the logic of the game, as defined by the rules. You will be getting one point for each passing test case.	**5 to >0.0 pts** **Full Marks**	**0 pts** **No Marks**	5 pts
			Total Points: 20