More on Assertions

Overview

Assertions in JUnit 5 provide a way to validate the expected behaviour of your code. They allow you to check conditions and ensure the outcomes match your expectations. In the context of a Spring Boot application, assertions are commonly used within test classes to verify the behaviour of methods or functionalities in your code.

Here are the key points to understand about Assertions in JUnit5 within Spring Boot:

- **Validation of Expected Results**: Assertions validate that the actual result obtained from executing a method or a piece of code matches the expected result.

- **Error Reporting**: When an assertion fails (i.e., the condition being checked is not true), JUnit 5 provides detailed information about the failure, including expected and actual values. This aids in debugging and identifying issues in the code.

- **Different Types of Assertions**: JUnit 5 provides various assertion methods to cater to different types of validations (e.g., equality checks, null checks, truthy/falsy conditions, etc.). Developers can choose the appropriate assertion based on the scenario they are testing.

- **Integration in Test Methods**: Assertions are integrated within test methods annotated with @Test in JUnit 5. These assertions evaluate the conditions within the test method and determine if the test has passed or failed.

- **Improving Code Quality**: By incorporating assertions within unit tests, developers can ensure that their code meets the specified requirements and behaviour, thereby enhancing the overall quality and reliability of the application.

Understanding and using assertions effectively in unit tests contribute significantly to the robustness and correctness of the codebase in a Spring Boot application. They help catch bugs early in the development process, making the software more maintainable and dependable.

Buddy Application

For practice and to better understand assertions, minor changes have been made to the existing buddy application. Please note that the modified version of the Buddy application is solely intended for practice and can be downloaded from the link below.

Link: BuddyApplication

During this session, we will be focusing on the following assertions:

- ★ **Assertions.assertNotNull();**
- ★ **Assertions.assertNull();**
- ★ **Assertions.assertTrue();**
- ★ **Assertions.assertFalse();**
- ★ **Assertions.assertThat;**

Testing with assertNotNull

- ★ Presented below is the addStudent method for the StudentsService class.
- ★ Here, we will conduct tests to verify the service layer's functionality, ensuring that it returns the Student object upon successful saving in the database.

```
public Student addStudent(Student student) {
            return studentDal.save(student);
      }
```

- ★ We utilise ***assertNotNull*** to validate that the addStudent method returns a non-null value. By employing ***assertEquals***, we verify that the returned entity matches our expected outcome.

```
            @Test
            public void shouldAddStudent() {
                  Student student = new Student(1,"Rakesh",19,"JUnit","Aryabhatta
Hostels","rakesh@gmail.com","09874562134");
                  Mockito.when(studentsDal.save(student)).thenReturn(student);
                  Student resulStudent1 = studentsService.addStudent(student);
                  Assertions.assertNotNull(resulStudent1);
                  Assertions.assertEquals(student,resulStudent1);
            }
```

Testing with assertThat

The **assertThat** method in AssertJ is a powerful tool for making assertions in Java tests. It offers a variety of assertion methods that can be used to verify different conditions in your tests. Here's an overview of some commonly used assertThat methods in AssertJ along with explanations and examples:

- **Equality Assertions**

  - **isEqualTo:** It checks if two objects are equal.

```
    Assertions.assertThat(result).isEqualTo(expectedResult);
```

  - **isNotEqualTo**: It verifies that two objects are not equal.

```
    Assertions.assertThat(result).isNotEqualTo(unexpectedResult);
```

- **Null/Non-Null Assertions**

  - **isNull**: It verifies that an object is null.

```
Assertions.assertThat(someObject).isNull();
```

  - **isNotNull**: It ensures that an object is not null.

```
Assertions.assertThat(someObject).isNotNull();
```

- **Boolean Assertions**

  - **isTrue**: It checks if a boolean expression is true.

```
Assertions.assertThat(isValid).isTrue();
```

  - **isFalse**: It verifies that a boolean expression is false.

```
Assertions.assertThat(isInvalid).isFalse();
```

- **String Assertions**

  - **contains**: It checks if a string contains a specific substring.

```
Assertions.assertThat(myString).contains("substring");
```

  - **startsWith**: It verifies that a string starts with a specified prefix.

```
Assertions.assertThat(myString).startsWith("prefix");
```

  - **endsWith**: Ensures that a string ends with a specified suffix.

```
Assertions.assertThat(myString).endsWith("suffix");
```

- **Numeric Assertions**

  - **isGreaterThan**: It verifies that a number is greater than a given value.

```
Assertions.assertThat(number).isGreaterThan(10);
```

  - **isLessThan**: It checks if a number is less than a specified value.

```
Assertions.assertThat(number).isLessThan(100);
```

These are just a few examples of `assertThat` methods provided by AssertJ. There are many more available in the library, offering a wide range of assertions to verify different conditions in your tests.

Practice Tasks:-

You are expected to test the following method using the specified assertions:

- *TrueOrFalse(boolean result)*:- **assertTrue** / **assertFalse / assertThat**
- *multiplyServiceRate(int num)*:- **assertEquals / assertNotNull / assertThat**
- *getStudentById(int sid)*:- **assertNotNull / assertEquals / assertThat**
- *deleteStudent(int sid)*:- **assertTrue** / **assertFalse / assertThat**
- *getStudents():-* **assertNull / assertNotNull / assertThat**

The **Buddy Application** offers various testing possibilities beyond the specified assertions, serving as learning aids. You're encouraged to explore and apply diverse testing methodologies in different contexts.

Note:

To use assertions from JUnit *Jupiter* (**org.junit.jupiter.api.Assertions**) and *AssertJ* (**org.assertj.core.api.Assertions**) together in a single test case. These assertion libraries serve a similar purpose of validating conditions in tests but offer different assertion methods and functionalities.

Here's an example of how you might use both in a single test case:

```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.assertj.core.api.Assertions.assertThat;

public class MixedAssertionsTest {

    @Test
    public void testMixedAssertions() {
        // Using JUnit Jupiter assertion
        int actualResult = 10 + 5;
        assertEquals(15, actualResult); // JUnit Jupiter assertion

        // Using AssertJ assertion
        String text = "Hello, World!";
        assertThat(text).startsWith("Hello"); // AssertJ assertion
    }
}
```

In this example:

- assertEquals is a method from JUnit Jupiter's assertions package (org.junit.jupiter.api.Assertions). It checks if two values are equal.

- **assertThat** is a method from AssertJ (`org.assertj.core.api.Assertions`). It provides a more fluent and expressive way of making assertions, here checking if the string `text` starts with the substring "Hello".

Both types of assertions can coexist within the same test case. This can be beneficial when you want to leverage the specific features and functionalities provided by different assertion libraries in your tests. However, it's generally recommended to choose one consistent assertion library for a project to maintain readability and avoid confusion.

References:

1. Baeldung

2. Junit

3. JetBrains