Testing Java with JUnit and TestNG

Testing Java with JUnit and TestNG

Objectives

- To know what a test framework is and how one works
- To know the test frameworks JUnit4 , JUnit5 and TestNG are
- To create and run a JUnit test class from Visual Studio Code

Testing Java with JUnit and TestNG

INTRODUCTION

The software development process was discussed in the course forum. It includes requirements specification, analysis, design, implementation, testing, deployment, and maintenance. Testing is an essential practice to ensure the quality and reliability of your code. Test frameworks provide a structured and systematic approach to writing, organizing, and running tests for your Java applications. In this text, we will explore two popular Java test frameworks: JUnit and TestNG. These frameworks help you automate the testing process, making it easier to catch bugs and ensure your code functions as expected.

What Are Test Frameworks?

Test frameworks are tools that simplify the creation, execution, and management of test cases. They provide a structured way to:

- Define test cases: Specify what you want to test and what the expected outcomes should be.
- Automate test execution: Run tests automatically, reducing the need for manual testing.
- Report results: Generate detailed reports on test outcomes, helping you identify and fix issues.
- Customize testing behavior: Configure how tests are run, including ordering, parallel execution, and more.

TESTING BASICS

The test framework library is packaged in a jar file. The jar file contains a tool called *test runner*, which is used to run test programs. Suppose you have a class named Process. To test this class, you write a test class named ProcessTest. This test class, called a test runner, contains the methods you write for testing the class Process.

THE TESTING PROCESS

Using a Java test framework like JUnit or TestNG involves several steps, from setting up the framework in your project to writing and running test cases. Following is a step-by-step guide on how to set up and use a Java test framework, assuming you're starting with a new or existing Java project.

1: Project Setup
Create or Open a Java Project: If you don't already have a Java project, create one.

2: Add Test Framework Dependency
Add the the testing framework as a dependency in your project.

3: Write Test Cases

a. Create a new Java class or classes specifically for writing test cases. These classes will typically have methods annotated with @Test (for JUnit and TestNG).

b. In your test class, write test methods that use assertions to verify the correctness of your code. Test methods should be annotated with @Test (JUnit and TestNG) and contain the test logic.

4: Running Tests
Visual Studio Code provides built-in support for running tests. Locate the test class or test methods and run them. Visual Studio Code will execute the tests and display the results.

5: Analyzing Test Results

a. Most test frameworks generate test reports that provide detailed information about test results. These reports can be viewed in Visual Studio Code or generated as HTML or XML files.
b. Analyze the test results to identify any failures or errors. If a test fails, review the failure message and stack trace to understand the issue.

Visual Studio Code Project Setup

Testing Java in Visual Studio Code is enabled by the Test Runner for Java extension. If you installed Visual Studio Code with the Extension Pack for Java, the Test Runner extension was included. The extension supports JUnit 4, JUnit 5 and TestNG.

You can enable a test framework for your unmanaged folder project (a project without any build tools) with just a few steps in the Testing Explorer. You can also enable a test framework for managed folder projects (Gradle and Maven). See Enable testing and adding test framework JARs to your project in the Visual Studio Code documentation.

If your project does not use any build tools, you can enable JUnit 4, JUnit 5 or TestNG via the Testing Explorer.

To Run/Debug test cases see Run/Debug test cases in the Visual Studio Code documentation.

See also

- Testing Explorer
- View test results
- Generate tests

ANNOTATIONS

Annotations in Java are a form of metadata that can be added to Java code to provide additional information about the code's behavior, structure, or characteristics. Annotations do not directly affect the execution of the code; instead, they are used by tools, frameworks, and libraries to configure, process, or generate code automatically. Annotations are typically introduced with the @ symbol.

Here are some key points about annotations in Java:

1. **Annotations are Metadata:** Annotations are used to attach metadata or information about code elements like classes, methods, fields, and parameters. This metadata can be used for various purposes such as documentation, validation, and code generation.

2. **Annotations Are Predefined and Customizable:** Java provides a set of predefined annotations like **@Override**, **@Deprecated**, and **@SuppressWarnings**. Additionally, developers can create their custom annotations to suit specific requirements.

3. **Annotations Are Retained at Runtime (if specified):** By default, annotations are not retained at runtime, but some annotations can be configured to be retained using the **@Retention** annotation with the **RUNTIME** retention policy. This allows runtime inspection and processing of annotations by reflection.

4. **Annotations Have Target Elements:** Annotations can be applied to specific program elements. For example, **@Override** is typically applied to methods, while **@Entity** (used in Java Persistence API) is applied to classes.

5. **Annotations Can Have Values:** Annotations can accept values or parameters that provide additional information or configuration details. These values can be primitive types, strings, enums, classes, or arrays of these types.

6. **Annotations Can Be Used for Code Generation:** Many Java frameworks and tools use annotations to generate code or configuration files automatically. For example, Java Persistence API (JPA) uses annotations to define mappings between Java classes and database tables.

7. **Annotations Are Processed by Annotation Processors:** Annotation processors are tools that can read and process annotations during compilation. They can generate code, perform validation, or generate configuration files based on the annotations present in the code.

Annotations play a crucial role in modern Java development, as they enable better documentation, code analysis, and automation through various tools and frameworks. They enhance code readability and maintainability by providing context and intent within the codebase.

Annotations used for testing in Java are an integral part of testing frameworks. These annotations help define and configure test cases, test suites, and testing behavior.

JUnit Annotations

JUnit 4 is one of the most widely used Java testing frameworks. It provides a simple and efficient way to write and execute tests. JUnit 5 is the next iteration of the JUnit framework and introduces many new features and improvements. It offers greater flexibility and extensibility.

- **@Test**: Indicates that the method is a test case.

```
@Test
public void testAddition() {
    int result = MyMath.add(2, 3);
    assertEquals(5, result);
}
```

- **@Before**: Executed before each test method.

```
@Before
public void setUp() {
    // Initialize resources or perform setup actions
}
```

- **@After**: Executed after each test method.

```
@After
public void tearDown() {
    // Release resources or perform cleanup actions
}
```

- **@BeforeClass**: Executed once before all test methods in the class.

```
@BeforeClass
public static void setUpClass() {
    // Initialize shared resources or perform one-time setup
actions
}
```

- **@AfterClass**: Executed once after all test methods in the class.

```
@AfterClass
public static void tearDownClass() {
    // Release shared resources or perform one-time cleanup
actions
}
```

- **@Ignore**: Skips the annotated test method.

```
@Ignore
public void ignoredTest() {
    // This test will be skipped
}
```

TestNG

TestNG is another popular Java testing framework known for its powerful features, including support for parallel test execution and test parameterization.

- **@Test**: Marks a method as a test case.

```
@Test
public void testAddition() {
    int result = MyMath.add(2, 3);
    assertEquals(result, 5);
}
```

- **@BeforeMethod**: Indicates a method to be run before each test method in the test class, typically used for setup.

```
@BeforeMethod
public void setUp() {
    // Initialize resources or perform setup actions
}
```

- **@AfterMethod**: Indicates a method to be run after each test method in the test class, typically used for cleanup.

```
@AfterMethod
public void tearDown() {
    // Release resources or perform cleanup actions
}
```

- **@BeforeClass**: Indicates a method to be run once before any test methods in the test class, typically used for one-time setup.

```
@BeforeClass
public void setUpClass() {
    // Initialize shared resources or perform one-time setup
}
```

- **@AfterClass**: Indicates a method to be run once after all test methods in the test class, typically used for one-time cleanup.

```
@AfterClass
public void tearDownClass() {
    // Release shared resources or perform one-time cleanup
actions
}
```

- **@Ignore**: Marks a test method to be ignored during test execution.

```
@Test(enabled = false)
public void ignoredTest() {
    // This test will be skipped
}
```

ADDITIONAL READING & VIDEO

JUnit 4

JUnit 5

TestNG

How to Setup JUnit for VS Code | JUnit in Visual Studio Code | Java Test with Visual Studio Code