

Advanced JUnit Testing Exercises

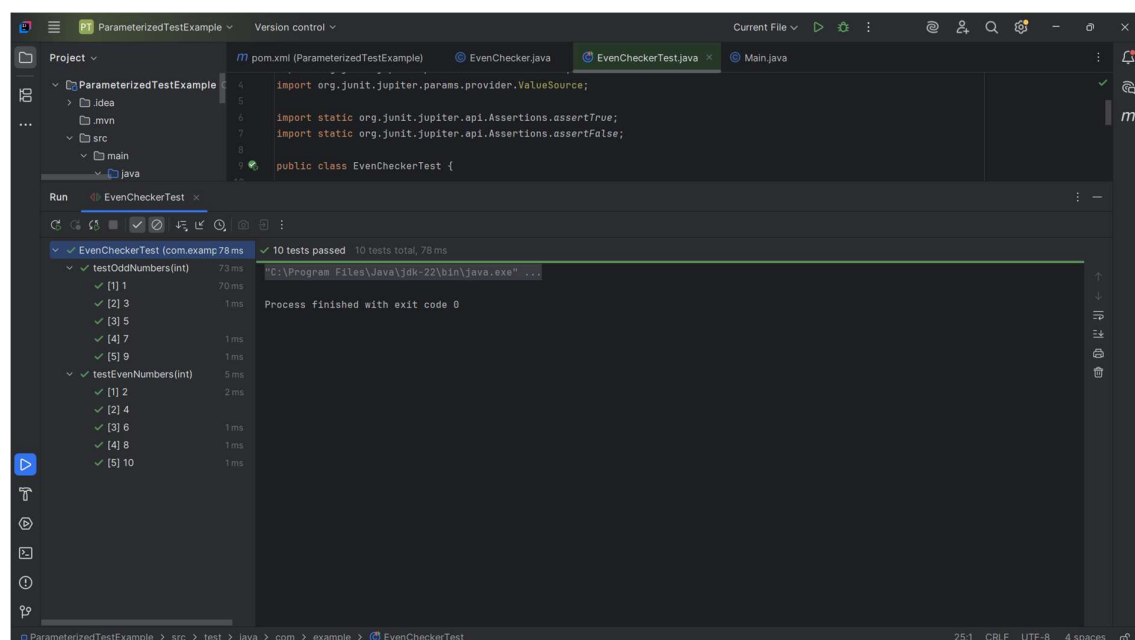
Exercise 1: Parameterized Tests

Scenario: You want to test a method that checks if a number is even. Instead of writing multiple test cases, you will use parameterized tests to run the same test with different inputs.

Steps:

1. Create a new Java class `EvenChecker` with a method `isEven(int number)`.
2. Write a parameterized test class `EvenCheckerTest` that tests the `isEven` method with different inputs.
3. Use JUnit's `@ParameterizedTest` and `@ValueSource` annotations.

Output:



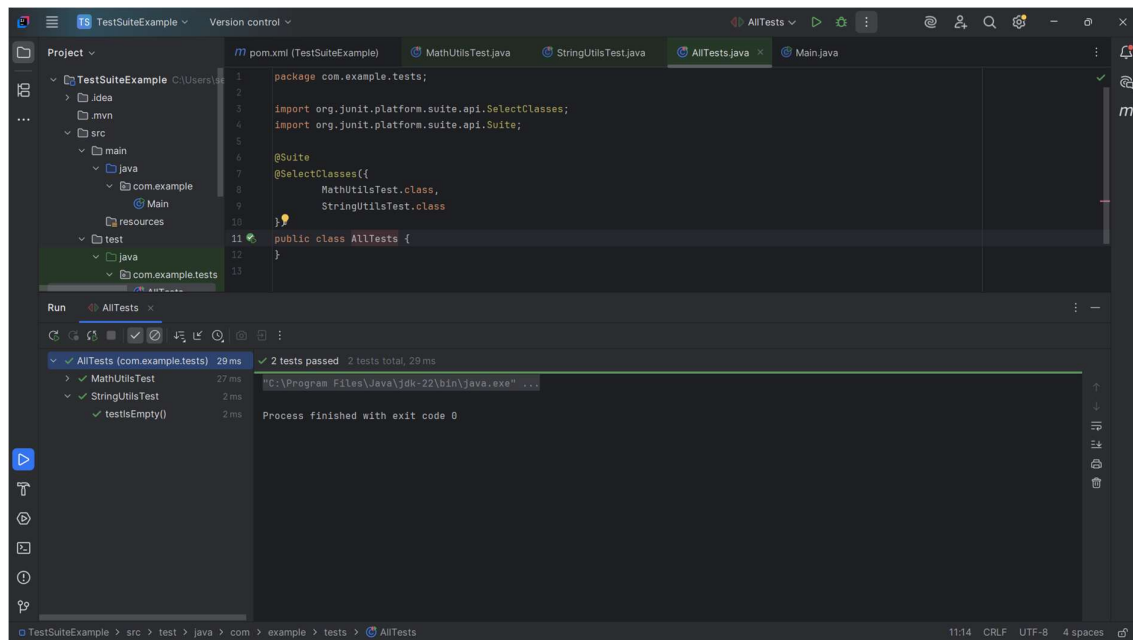
Exercise 2: Test Suites and Categories

Scenario: You want to group related tests into a test suite and categorize them.

Steps:

1. Create a new test suite class `AllTests`.
2. Add multiple test classes to the suite.
3. Use JUnit's `@Suite` and `@SelectClasses` annotations.

Output:



The screenshot shows an IDE with the following components:

- Project Explorer:** Shows a project named 'TestSuiteExample' with a package structure including 'com.example.tests'.
- Code Editor:** Displays the source code for 'AllTests.java'. The code uses JUnit's `@Suite` and `@SelectClasses` annotations to include `MathUtilsTest.class` and `StringUtilsTest.class`.
- Run Console:** Shows the execution results of the test suite. It indicates that 2 tests passed out of 2 tests total, with a total execution time of 29ms.

```
1 package com.example.tests;
2
3 import org.junit.platform.suite.api.SelectClasses;
4 import org.junit.platform.suite.api.Suite;
5
6 @Suite
7 @SelectClasses({
8     MathUtilsTest.class,
9     StringUtilsTest.class
10 })
11 public class AllTests {
12 }
13
```

Run: AllTests (com.example.tests) 29ms

2 tests passed 2 tests total, 29ms

Process finished with exit code 0

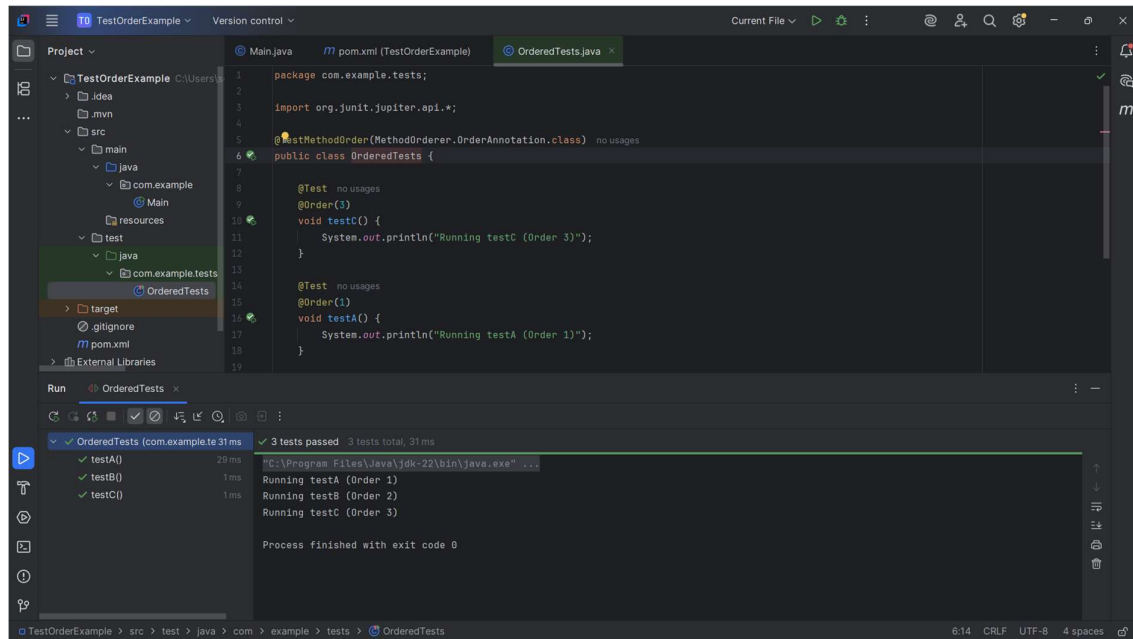
Exercise 3: Test Execution Order

Scenario: You want to control the order in which tests are executed.

Steps:

1. Create a test class `OrderedTests`.
2. Use JUnit's `@TestMethodOrder` and `@Order` annotations.

Output:



The screenshot shows an IDE window for a project named 'TestOrderExample'. The 'src' directory contains a package 'com.example.tests' with a class 'OrderedTests'. The code in 'OrderedTests.java' is as follows:

```
1 package com.example.tests;
2
3 import org.junit.jupiter.api.*;
4
5 @TestMethodOrder(MethodOrderer.OrderAnnotation.class)
6 public class OrderedTests {
7
8     @Test
9     @Order(3)
10    void testC() {
11        System.out.println("Running testC (Order 3)");
12    }
13
14    @Test
15    @Order(1)
16    void testA() {
17        System.out.println("Running testA (Order 1)");
18    }
19 }
```

The 'Run' tab shows the execution results for 'OrderedTests (com.example.tests)'. It indicates that 3 tests passed in 31 ms. The output log shows the following sequence of test runs:

```
Running testA (Order 1)
Running testB (Order 2)
Running testC (Order 3)

Process finished with exit code 0
```

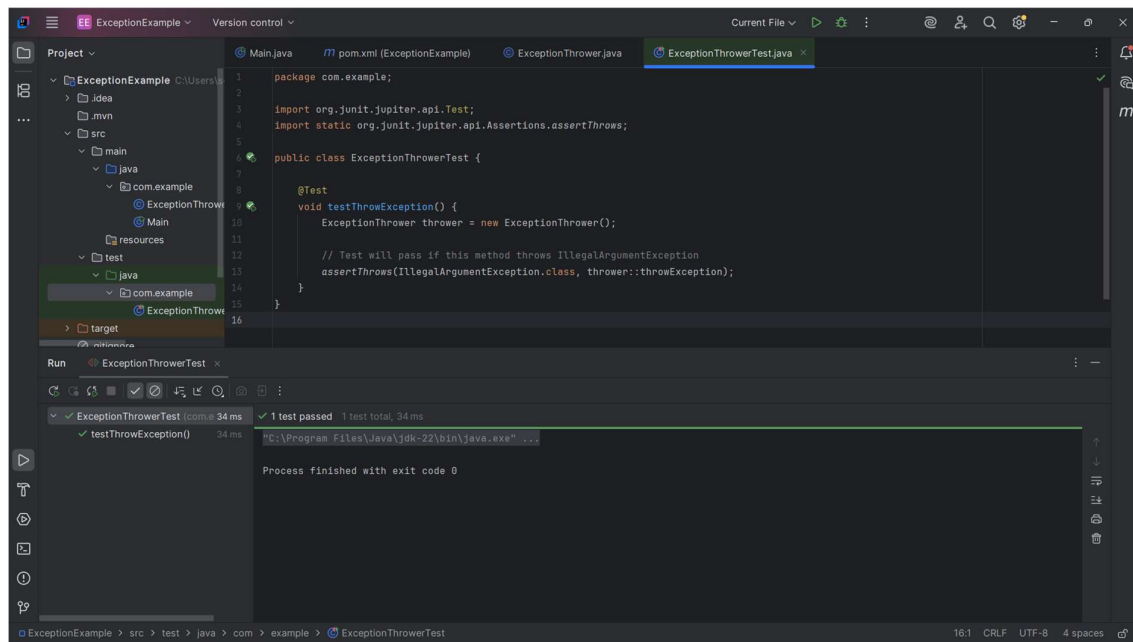
Exercise 4: Exception Testing

Scenario: You want to test that a method throws the expected exception.

Steps:

1. Create a class `ExceptionThrower` with a method `throwException`.
2. Write a test class `ExceptionThrowerTest` that tests the method for the expected exception.

Output:



The screenshot shows an IDE with the following content:

```
package com.example;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertThrows;

public class ExceptionThrowerTest {

    @Test
    void testThrowException() {
        ExceptionThrower thrower = new ExceptionThrower();

        // Test will pass if this method throws IllegalArgumentException
        assertThrows(IllegalArgumentException.class, thrower::throwException);
    }
}
```

The Run window shows the test results:

```
Run ExceptionThrowerTest
1 test passed 1 test total, 34 ms
testThrowException() 34 ms
```

The output console shows the command executed and the exit code:

```
C:\Program Files\Java\jdk-22\bin\java.exe ...
Process finished with exit code 0
```

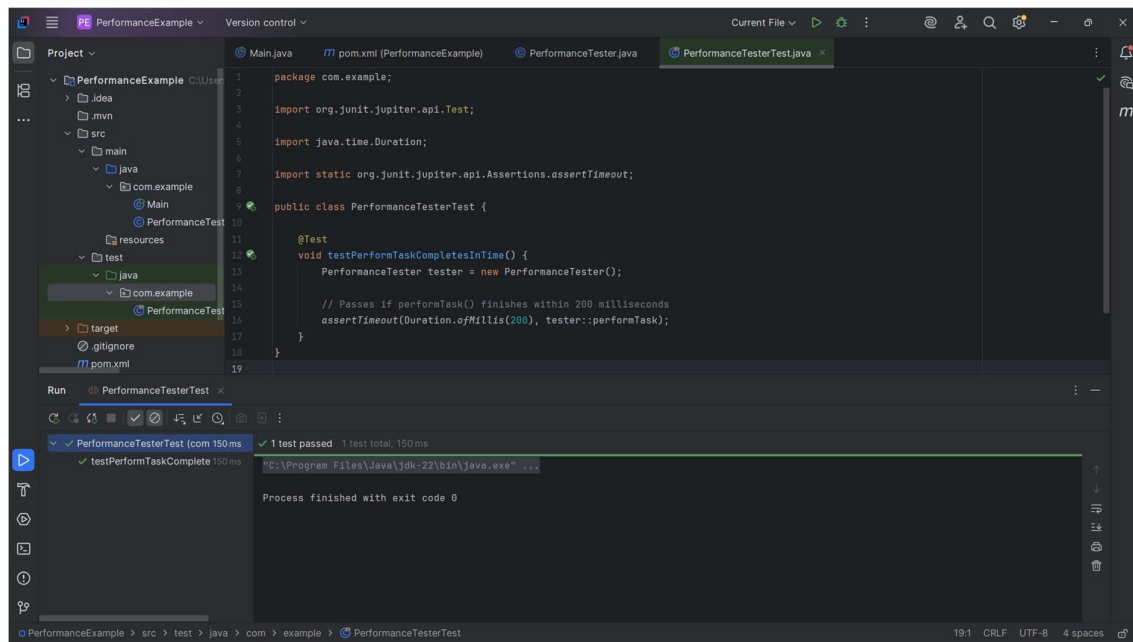
Exercise 5: Timeout and Performance Testing

Scenario: You want to ensure that a method completes within a specified time limit.

Steps:

1. Create a class `PerformanceTester` with a method `performTask`.
2. Write a test class `PerformanceTesterTest` that tests the method for timeout.

Output:



The screenshot shows an IDE window with the following content:

```
package com.example;

import org.junit.jupiter.api.Test;
import java.time.Duration;
import static org.junit.jupiter.api.Assertions.assertTimeout;

public class PerformanceTesterTest {

    @Test
    void testPerformTaskCompletesInTime() {
        PerformanceTester tester = new PerformanceTester();

        // Passes if performTask() finishes within 200 milliseconds
        assertTimeout(Duration.ofMillis(200), tester::performTask);
    }
}
```

The Run tab at the bottom shows the following output:

```
PerformanceTesterTest (com 150ms) ✓ 1 test passed 1 test total, 150ms
testPerformTaskComplete 150ms
"C:\Program Files\Java\jdk-22\bin\java.exe" ...
Process finished with exit code 0
```