Run your unit tests easily

JUnit 5 Test Framework

Óscar Barrios

Test Automation Specialist @SUSE

1 Setup

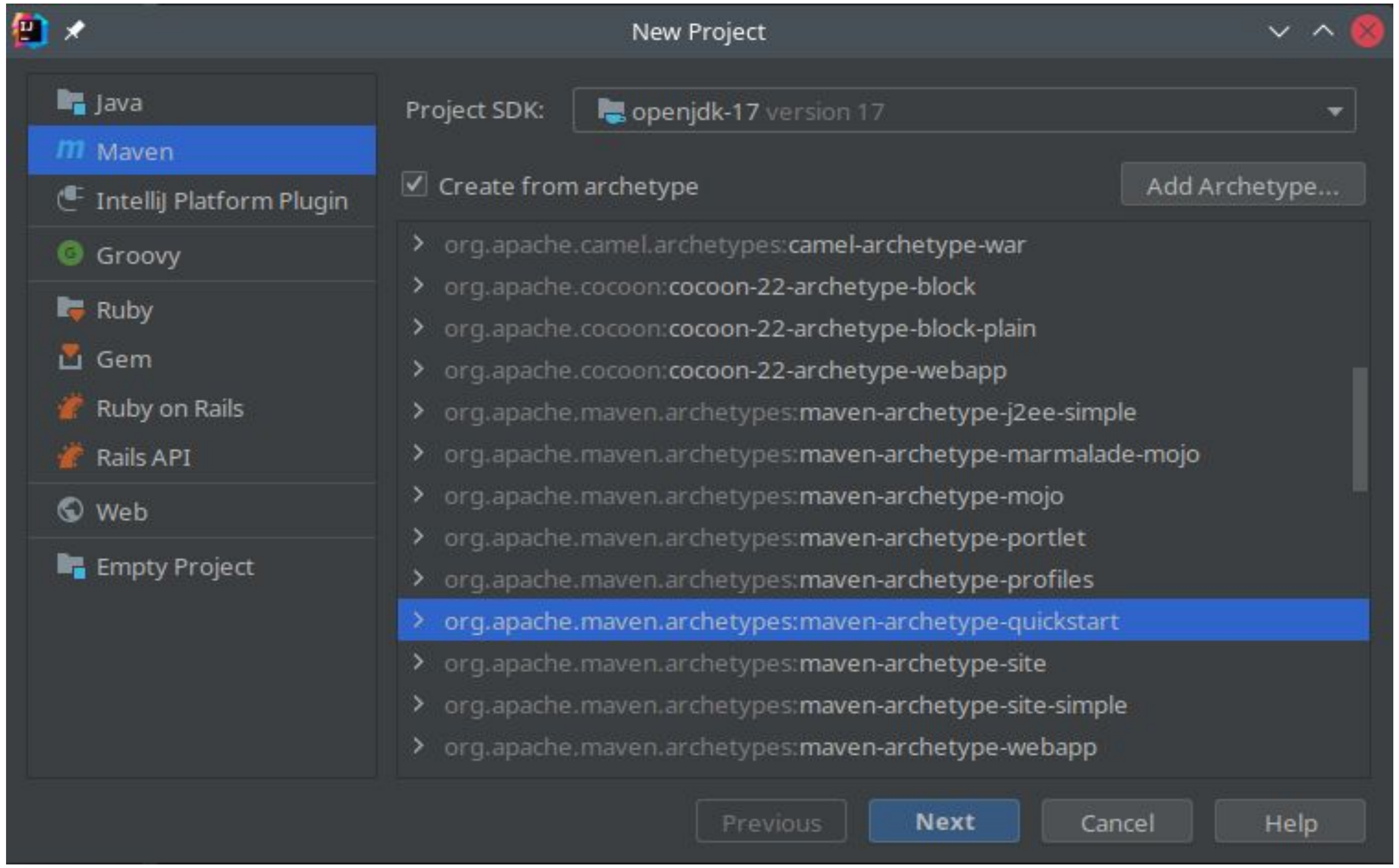
- **IntelliJ IDEA** Community Edition – https://www.jetbrains.com/idea/download/
- **Latest Java JDK** – Download it from IntelliJ using the Download JDK option – https://www.jetbrains.com/idea/guide/tips/download-jdk/ (Wait until the JDK is fully installed)
- Create a new **Maven** project named **JUnit5Tutorial** (don’t use whitespace characters)
- Include Maven dependency **JUnit Jupiter (Aggregator)** (latest available) https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter inside the pom.xml file, section **<dependencies>**, in case you find other dependencies inside this section, remove them.
- Enable **Skip tests** toggle in Maven panel (at top-right corner of the IDE)
- Run **clean** and **install** Maven tasks, to download JUnit dependencies (Wait until the dependencies are fully downloaded)

1 Setup

- If you have issues with the Maven project, try **Maven –> Reload project** (Contextual menu)
- If you have issues with dependencies, it might be an issue in your **Wifi connection**, check it!
- Check also **Preferences › Build, Execution, Deployment › Build Tools › Maven › Repositories**
- If install task in Maven give you warnings, be sure that you remove sample packages and classes, including folder like **resources**
- Check that your pom.xml use has Java 8:

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

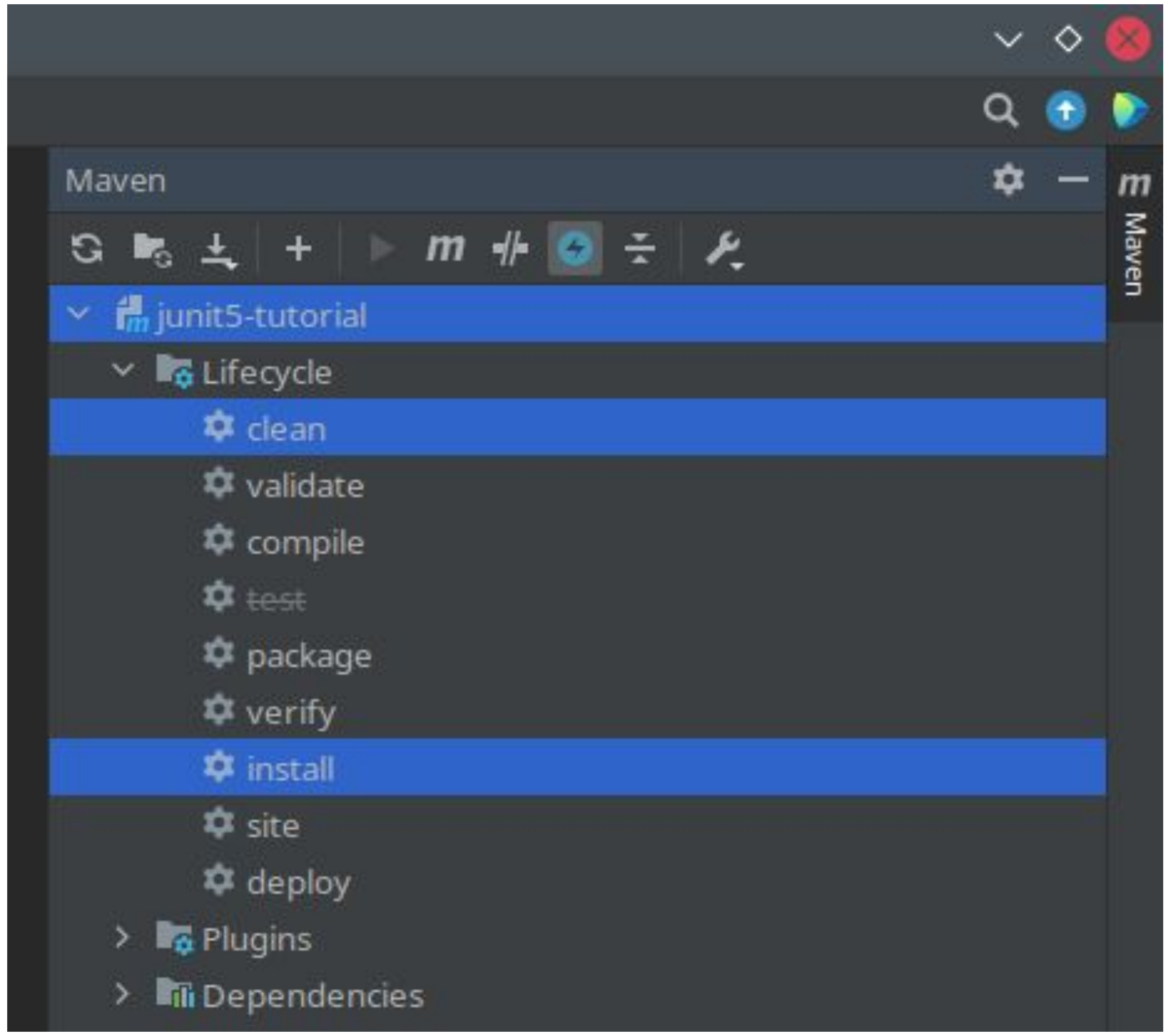
1 Setup



Project SDK: openjdk-17 version 17

☑ Create from archetype

- org.apache.camel.archetypes:camel-archetype-war
- org.apache.cocoon:cocoon-22-archetype-block
- org.apache.cocoon:cocoon-22-archetype-block-plain
- org.apache.cocoon:cocoon-22-archetype-webapp
- org.apache.maven.archetypes:maven-archetype-j2ee-simple
- org.apache.maven.archetypes:maven-archetype-marmalade-mojo
- org.apache.maven.archetypes:maven-archetype-mojo
- org.apache.maven.archetypes:maven-archetype-portlet
- org.apache.maven.archetypes:maven-archetype-profiles
- org.apache.maven.archetypes:maven-archetype-quickstart
- org.apache.maven.archetypes:maven-archetype-site
- org.apache.maven.archetypes:maven-archetype-site-simple
- org.apache.maven.archetypes:maven-archetype-webapp



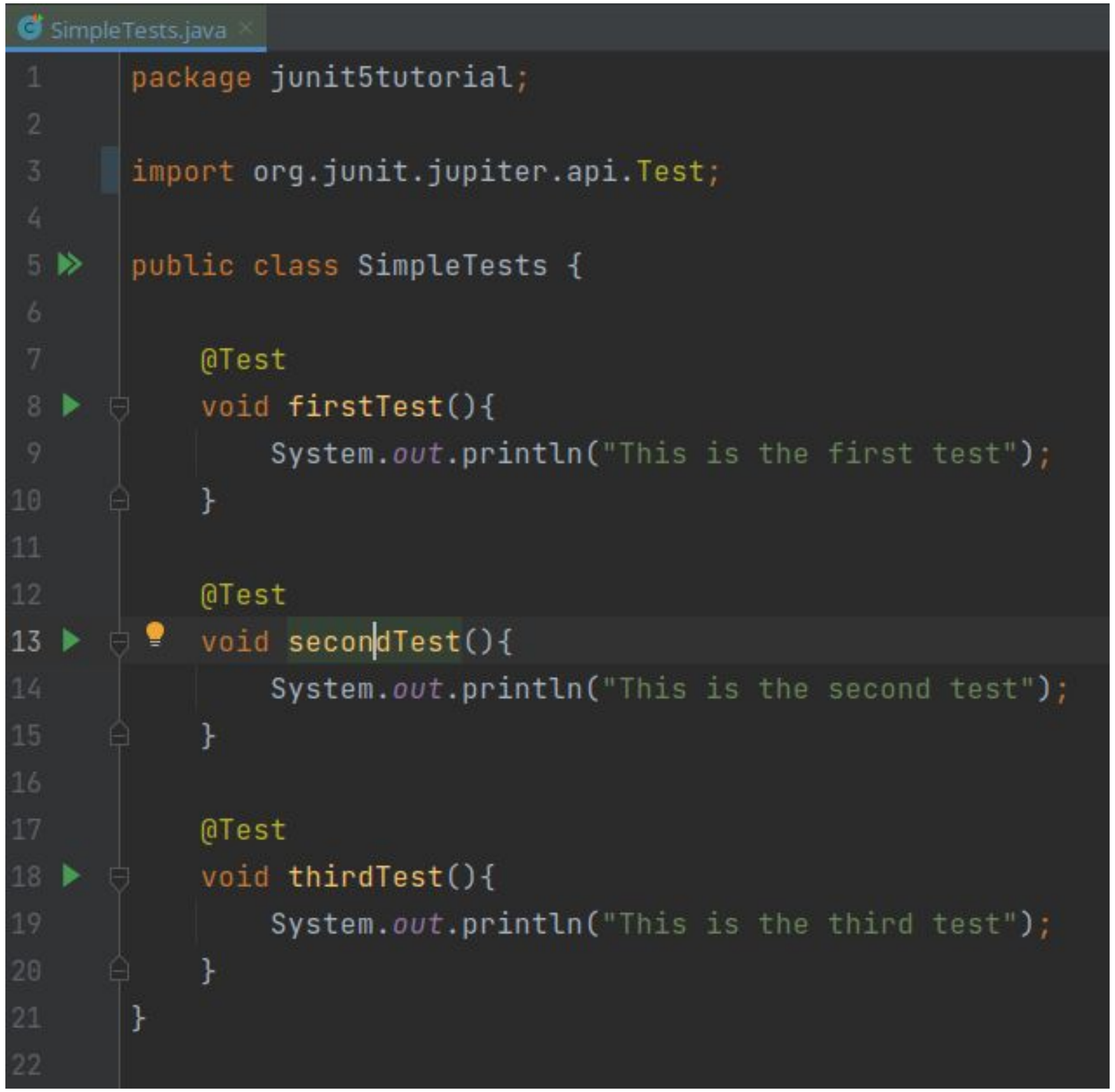
- junit5-tutorial
  - Lifecycle
    - clean
    - validate
    - compile
    - ~~test~~
    - package
    - verify
    - install
    - site
    - deploy
  - Plugins
  - Dependencies

```
<dependencies>
  <!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter -->
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>5.8.0</version>
    <scope>test</scope>
  </dependency>
```

2 Write a simple test

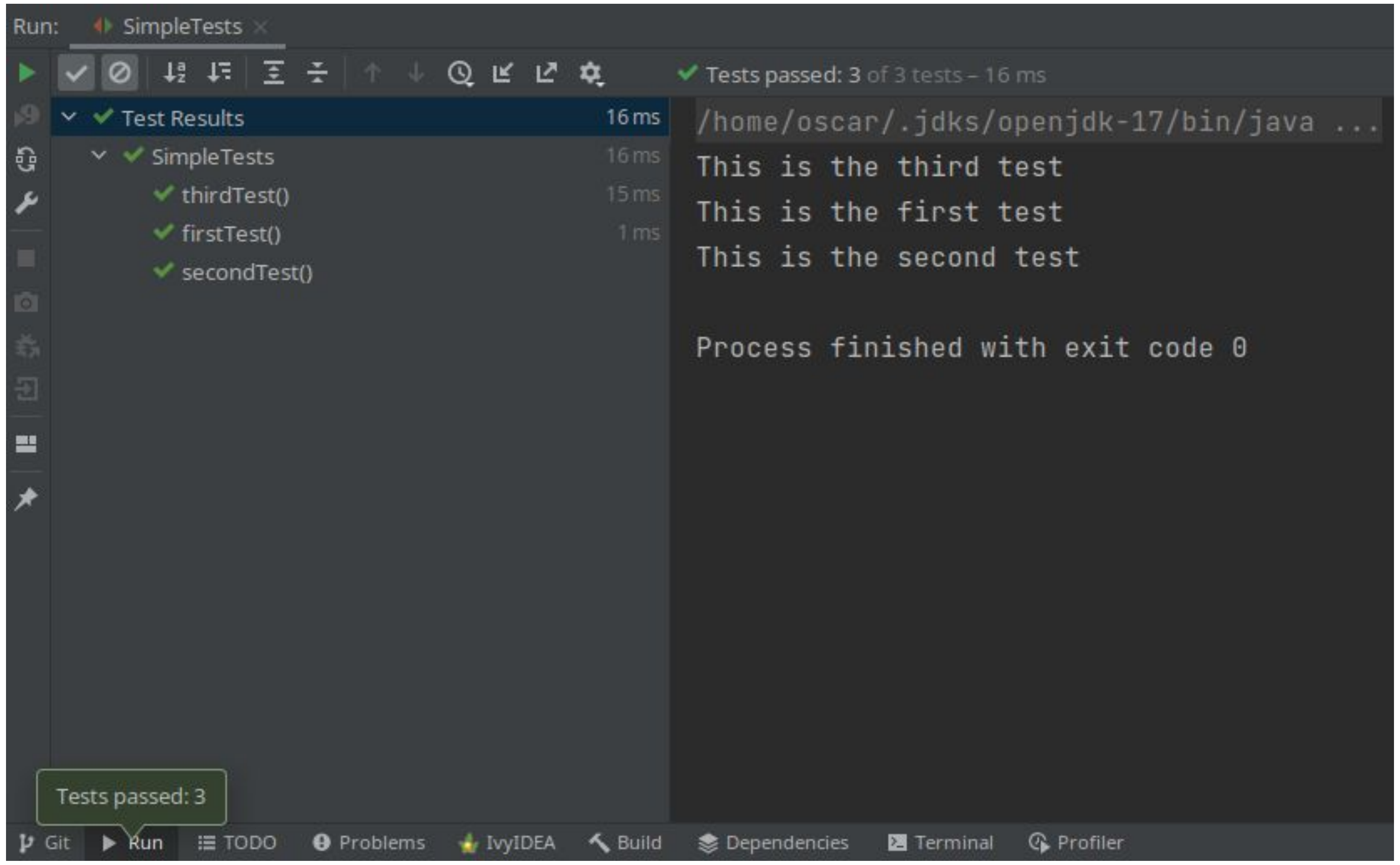
- Create a Package on **src/test/java** named **junit5tutorial**
- Create a Java class on **src/test/java/junit5tutorial** named **SimpleTests**
- Write a method returning **void** (Test methods can only return void)
- Add a **@Test** annotation to your method and import **org.junit.jupiter.api.Test**
- Print some text in order to identify when your test method has been executed
- Write two more tests with different name and output
- Use the **green button** close to the method signature to run a specific test
- Use the green button close to the class signature to run all the tests inside the class

2 Write a simple test



SimpleTests.java

```java
package junit5tutorial;

import org.junit.jupiter.api.Test;

public class SimpleTests {

    @Test
    void firstTest(){
        System.out.println("This is the first test");
    }

    @Test
    void secondTest(){
        System.out.println("This is the second test");
    }

    @Test
    void thirdTest(){
        System.out.println("This is the third test");
    }
}
```



Run: SimpleTests

✓ Tests passed: 3 of 3 tests – 16 ms

```
✓ Test Results        16 ms
  ✓ SimpleTests       16 ms
    ✓ thirdTest()     15 ms
    ✓ firstTest()      1 ms
    ✓ secondTest()
```

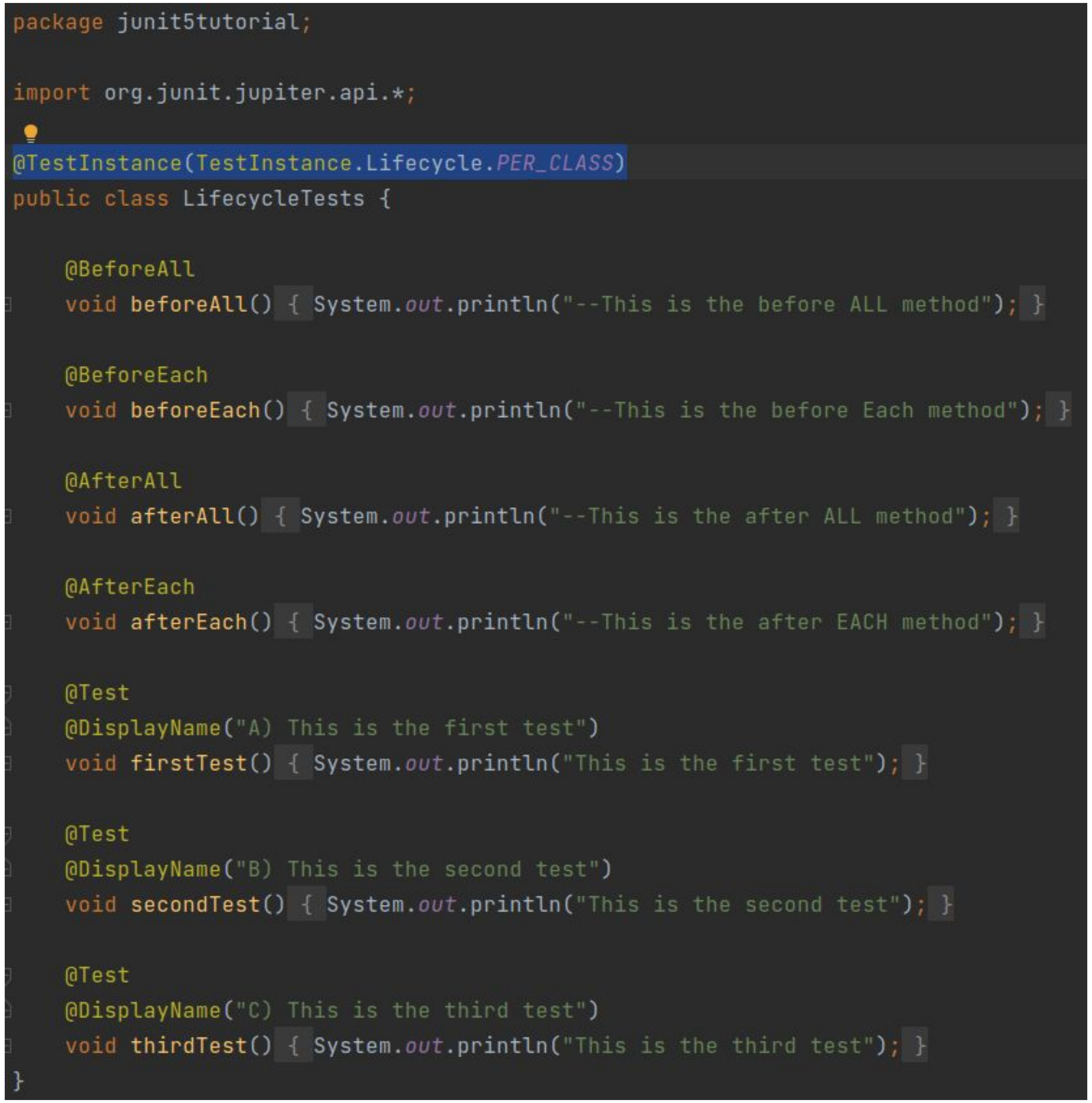
```
/home/oscar/.jdks/openjdk-17/bin/java ...
This is the third test
This is the first test
This is the second test

Process finished with exit code 0
```

Tests passed: 3

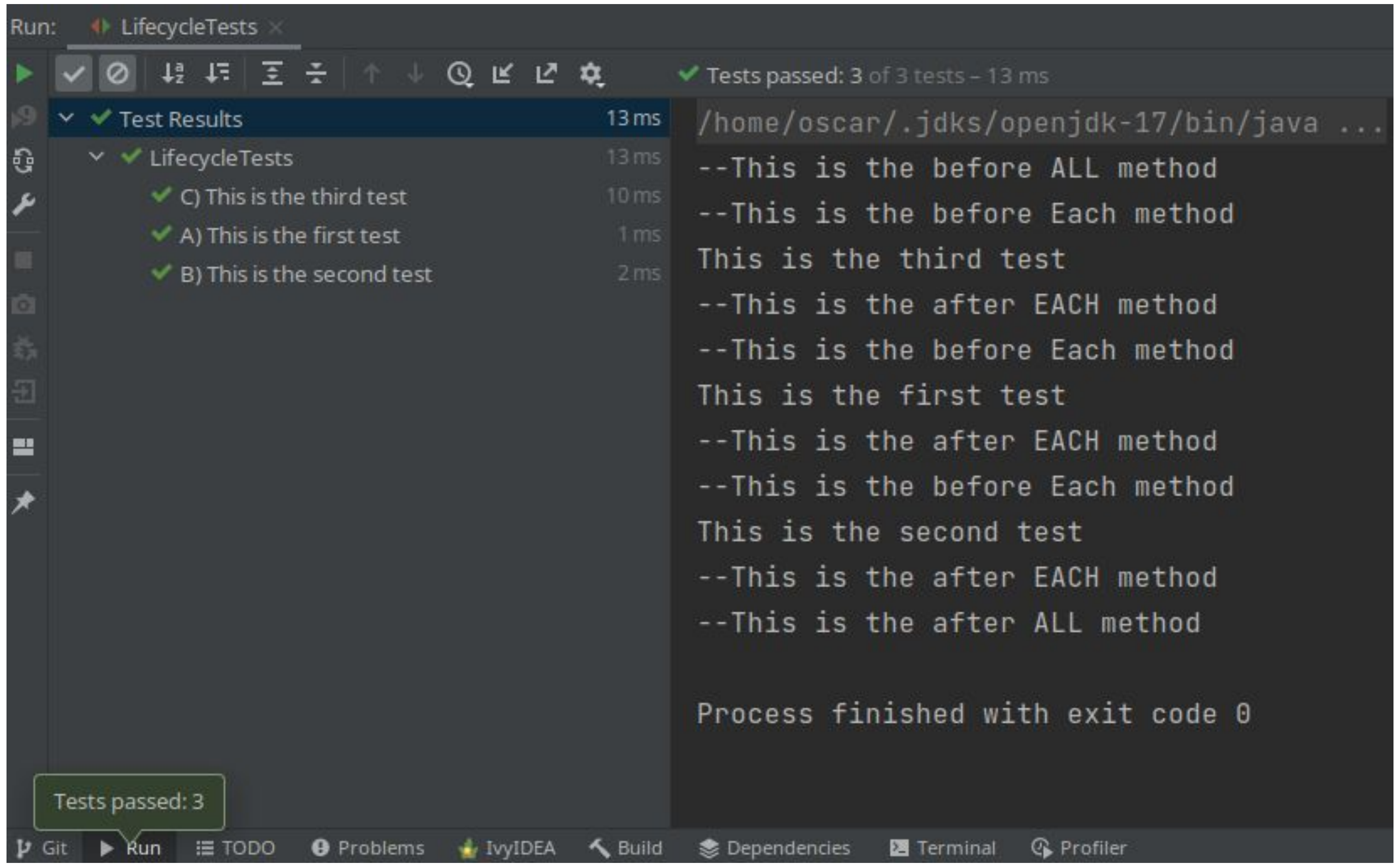
3 Lifecycle methods

- Copy your **SimpleTests** class and pasted as **LifecycleTests** class
- Add methods for each of these annotations: **@BeforeAll , @BeforeEach , @AfterAll , @AfterEach**
- Print a text to identify each of them
- Add this annotation at class level: **@TestInstance(TestInstance.Lifecycle.PER_CLASS)**
- Run the tests on this class and observe the printed output

3 Lifecycle methods



```java
package junit5tutorial;

import org.junit.jupiter.api.*;

@TestInstance(TestInstance.Lifecycle.PER_CLASS)
public class LifecycleTests {

    @BeforeAll
    void beforeAll() { System.out.println("--This is the before ALL method"); }

    @BeforeEach
    void beforeEach() { System.out.println("--This is the before Each method"); }

    @AfterAll
    void afterAll() { System.out.println("--This is the after ALL method"); }

    @AfterEach
    void afterEach() { System.out.println("--This is the after EACH method"); }

    @Test
    @DisplayName("A) This is the first test")
    void firstTest() { System.out.println("This is the first test"); }

    @Test
    @DisplayName("B) This is the second test")
    void secondTest() { System.out.println("This is the second test"); }

    @Test
    @DisplayName("C) This is the third test")
    void thirdTest() { System.out.println("This is the third test"); }
}
```



Run: LifecycleTests

✓ Tests passed: 3 of 3 tests – 13 ms

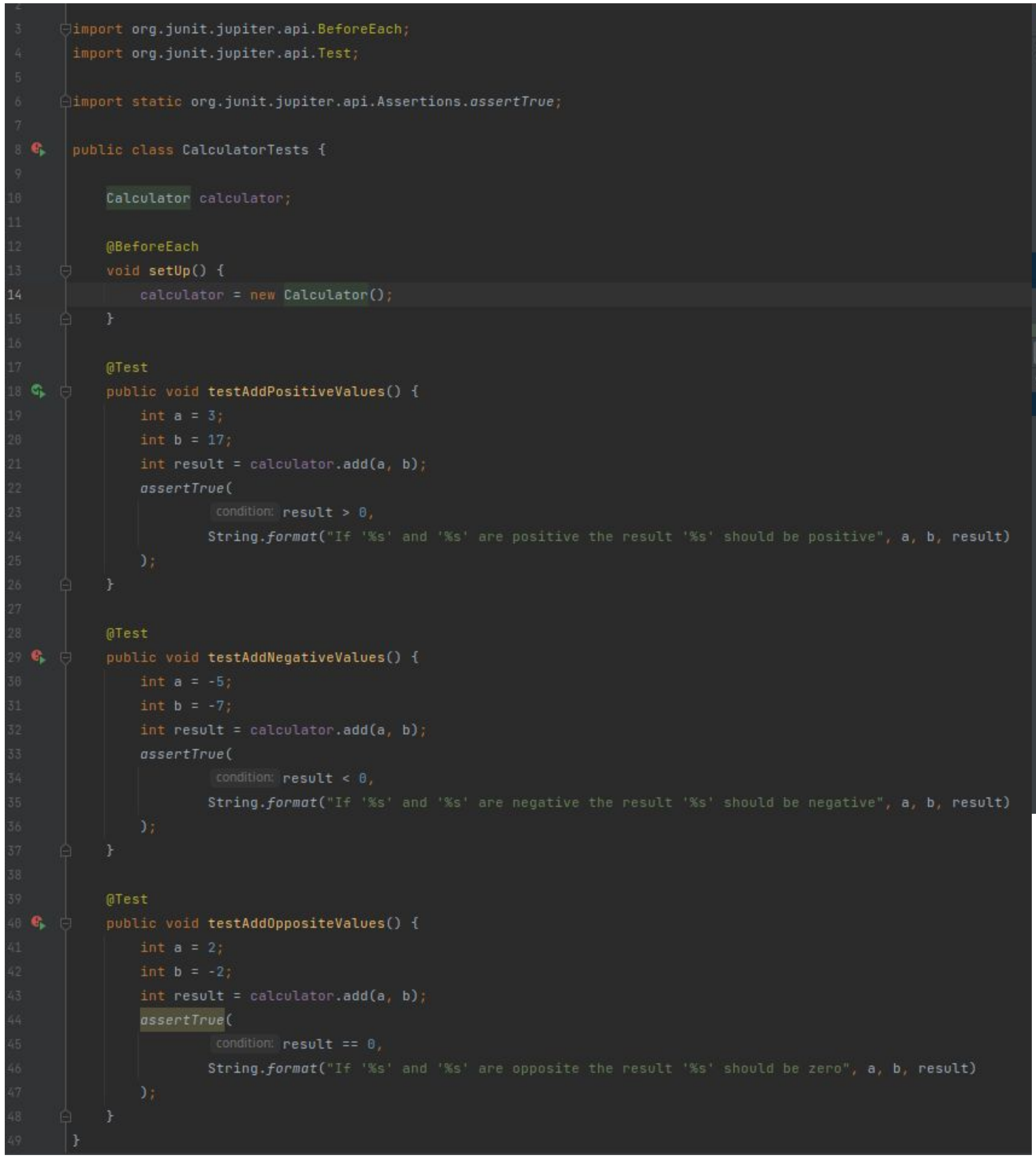
```
✓ Test Results                    13 ms
  ✓ LifecycleTests                13 ms
    ✓ C) This is the third test   10 ms
    ✓ A) This is the first test    1 ms
    ✓ B) This is the second test   2 ms
```

```
/home/oscar/.jdks/openjdk-17/bin/java ...
--This is the before ALL method
--This is the before Each method
This is the third test
--This is the after EACH method
--This is the before Each method
This is the first test
--This is the after EACH method
--This is the before Each method
This is the second test
--This is the after EACH method
--This is the after ALL method

Process finished with exit code 0
```

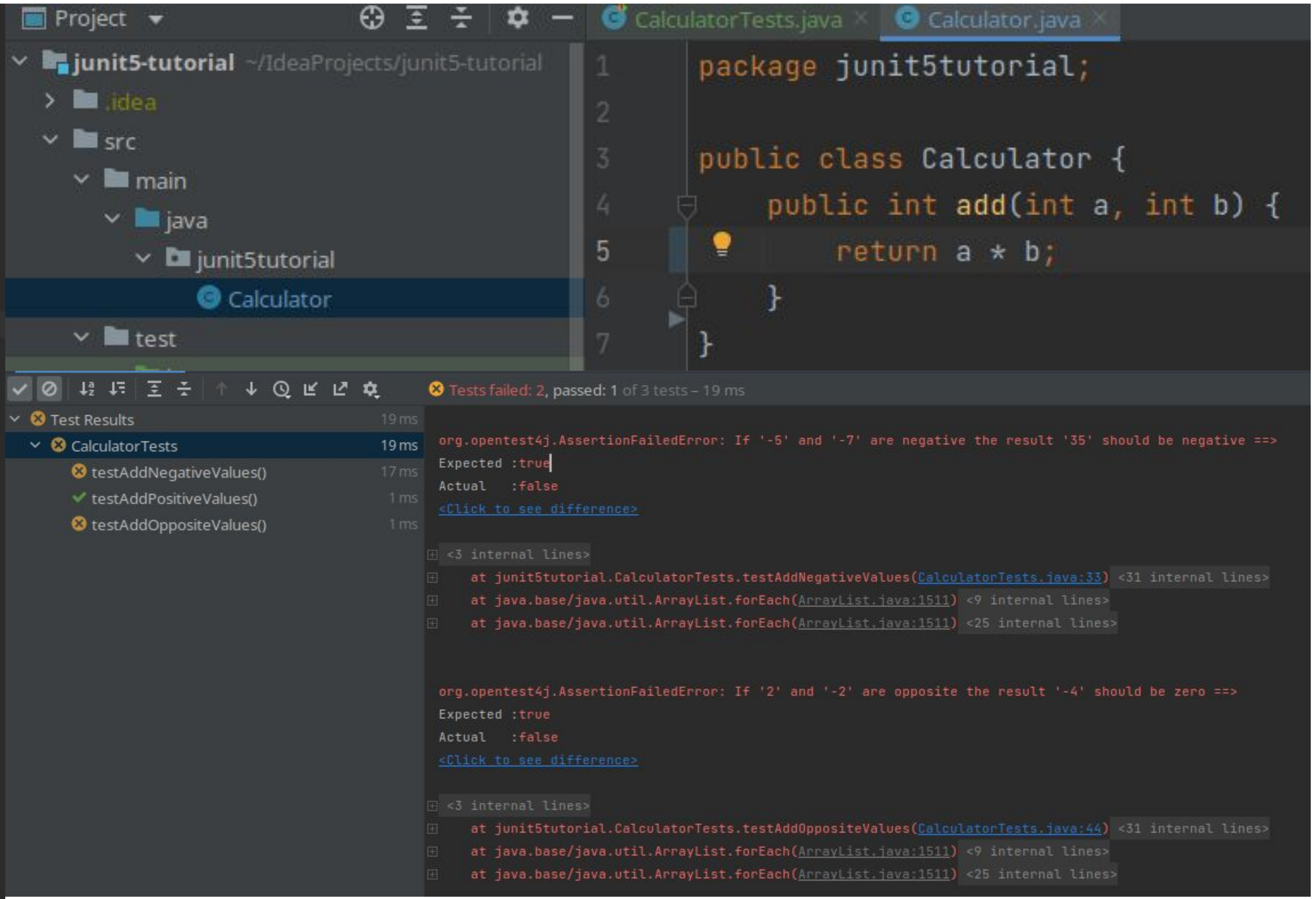
4 Test Driven Development

- Create a Package on **src/main/java** named **junit5tutorial**
- Create a Java class on **src/main/java/junit5tutorial** named **Calculator**
- Create a method with this signature **int add(int a, int b)** but don't implement it yet
- Create a **CalculatorTests** class on **src/test/java/junit5tutorial**
- Let's write tests with these acceptance criteria:
  - The user needs a **int add(int a, int b)** method
  - If **a** and **b** are positive the result should be positive
  - If **a** and **b** are negative the result should be negative
  - If **a** and **b** are opposite the result should be zero
- Implement **add** method, try with **return a * b;** and run the tests
- Implement **add** method satisfying the acceptance criteria

4 Test Driven Development



```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertTrue;

public class CalculatorTests {

    Calculator calculator;

    @BeforeEach
    void setUp() {
        calculator = new Calculator();
    }

    @Test
    public void testAddPositiveValues() {
        int a = 3;
        int b = 17;
        int result = calculator.add(a, b);
        assertTrue(
                condition: result > 0,
                String.format("If '%s' and '%s' are positive the result '%s' should be positive", a, b, result)
        );
    }

    @Test
    public void testAddNegativeValues() {
        int a = -5;
        int b = -7;
        int result = calculator.add(a, b);
        assertTrue(
                condition: result < 0,
                String.format("If '%s' and '%s' are negative the result '%s' should be negative", a, b, result)
        );
    }

    @Test
    public void testAddOppositeValues() {
        int a = 2;
        int b = -2;
        int result = calculator.add(a, b);
        assertTrue(
                condition: result == 0,
                String.format("If '%s' and '%s' are opposite the result '%s' should be zero", a, b, result)
        );
    }
}
```



```java
package junit5tutorial;

public class Calculator {
    public int add(int a, int b) {
        return a * b;
    }
}
```

```
Tests failed: 2, passed: 1 of 3 tests – 19 ms

Test Results                      19 ms
  CalculatorTests                 19 ms
    testAddNegativeValues()       17 ms
    testAddPositiveValues()        1 ms
    testAddOppositeValues()        1 ms
```

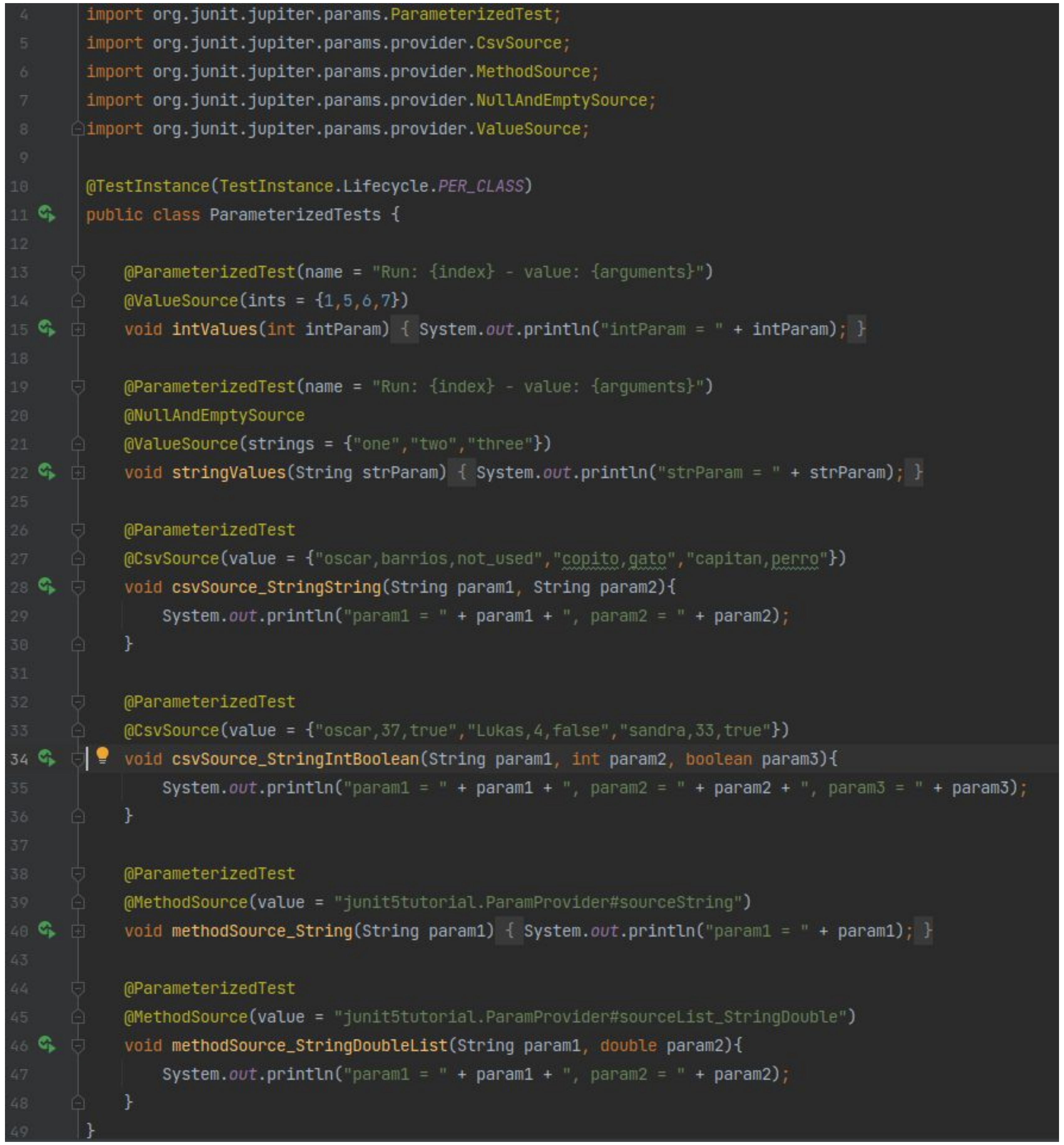
```
org.opentest4j.AssertionFailedError: If '-5' and '-7' are negative the result '35' should be negative ==>
Expected :true
Actual   :false
<Click to see difference>

<3 internal lines>
    at junit5tutorial.CalculatorTests.testAddNegativeValues(CalculatorTests.java:33) <31 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <9 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <25 internal lines>


org.opentest4j.AssertionFailedError: If '2' and '-2' are opposite the result '-4' should be zero ==>
Expected :true
Actual   :false
<Click to see difference>

<3 internal lines>
    at junit5tutorial.CalculatorTests.testAddOppositeValues(CalculatorTests.java:44) <31 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <9 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <25 internal lines>
```

5 Parameterized tests

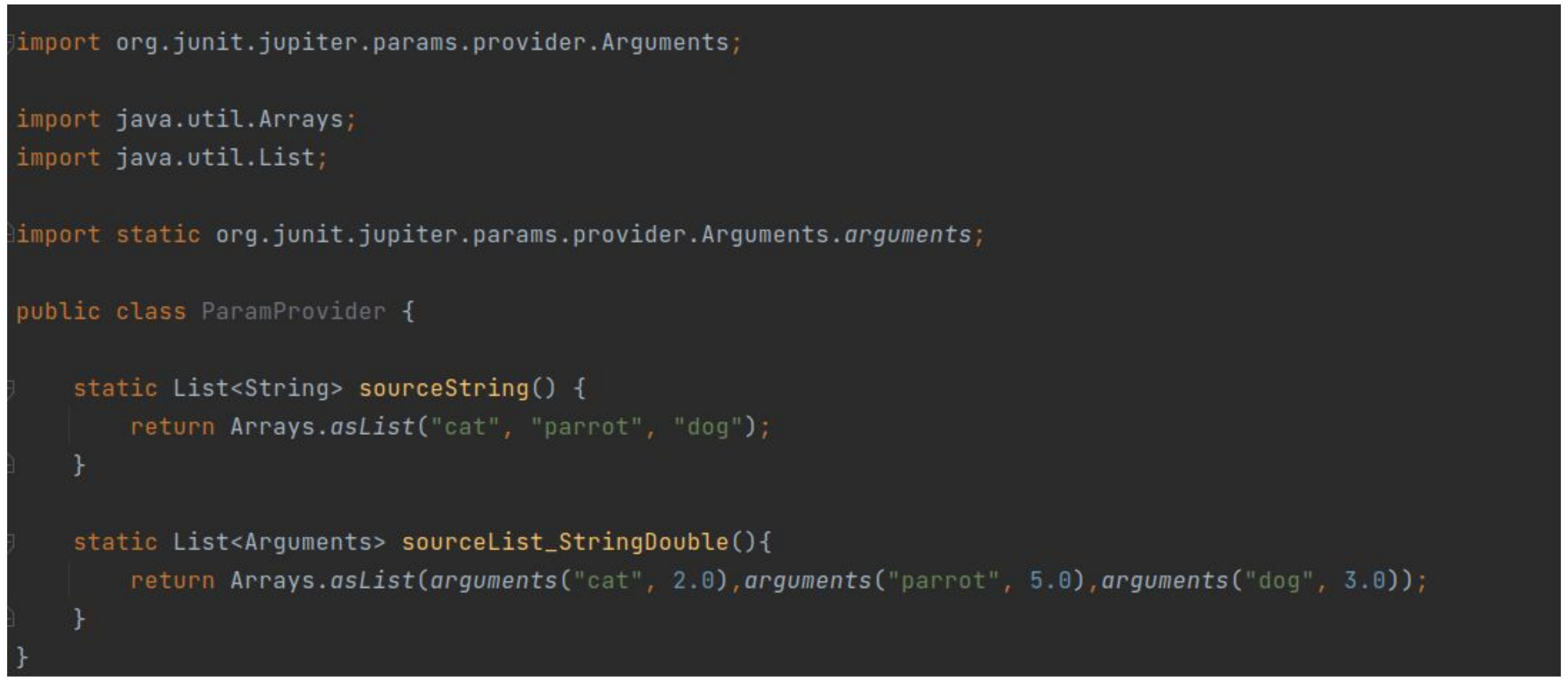
- Create a **ParameterizedTests** Class
- Create a **stringValues** method with a string parameter
- Add **@ParameterizedTest** and **@ValueSource(strings = {"one","two","three"})** annotations
- Print the value of the parameter
- Run the tests and observe the printed output
- Try another method with **@CsvSource(value = {"steve,32,true","captain,1,false","bucky,67,true"})** and three parameters (String param1, int param2, boolean param3)
- Try **@MethodSource(value = "junit5tutorial.ParamProvider#sourceString")**, implementing a Class **ParamProvider** with a method **List<String> sourceString()**

5

Parameterized tests



```java
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;
import org.junit.jupiter.params.provider.MethodSource;
import org.junit.jupiter.params.provider.NullAndEmptySource;
import org.junit.jupiter.params.provider.ValueSource;

@TestInstance(TestInstance.Lifecycle.PER_CLASS)
public class ParameterizedTests {

    @ParameterizedTest(name = "Run: {index} - value: {arguments}")
    @ValueSource(ints = {1,5,6,7})
    void intValues(int intParam) { System.out.println("intParam = " + intParam); }

    @ParameterizedTest(name = "Run: {index} - value: {arguments}")
    @NullAndEmptySource
    @ValueSource(strings = {"one","two","three"})
    void stringValues(String strParam) { System.out.println("strParam = " + strParam); }

    @ParameterizedTest
    @CsvSource(value = {"oscar,barrios,not_used","copito,gato","capitan,perro"})
    void csvSource_StringString(String param1, String param2){
        System.out.println("param1 = " + param1 + ", param2 = " + param2);
    }

    @ParameterizedTest
    @CsvSource(value = {"oscar,37,true","Lukas,4,false","sandra,33,true"})
    void csvSource_StringIntBoolean(String param1, int param2, boolean param3){
        System.out.println("param1 = " + param1 + ", param2 = " + param2 + ", param3 = " + param3);
    }

    @ParameterizedTest
    @MethodSource(value = "junit5tutorial.ParamProvider#sourceString")
    void methodSource_String(String param1) { System.out.println("param1 = " + param1); }

    @ParameterizedTest
    @MethodSource(value = "junit5tutorial.ParamProvider#sourceList_StringDouble")
    void methodSource_StringDoubleList(String param1, double param2){
        System.out.println("param1 = " + param1 + ", param2 = " + param2);
    }
}
```

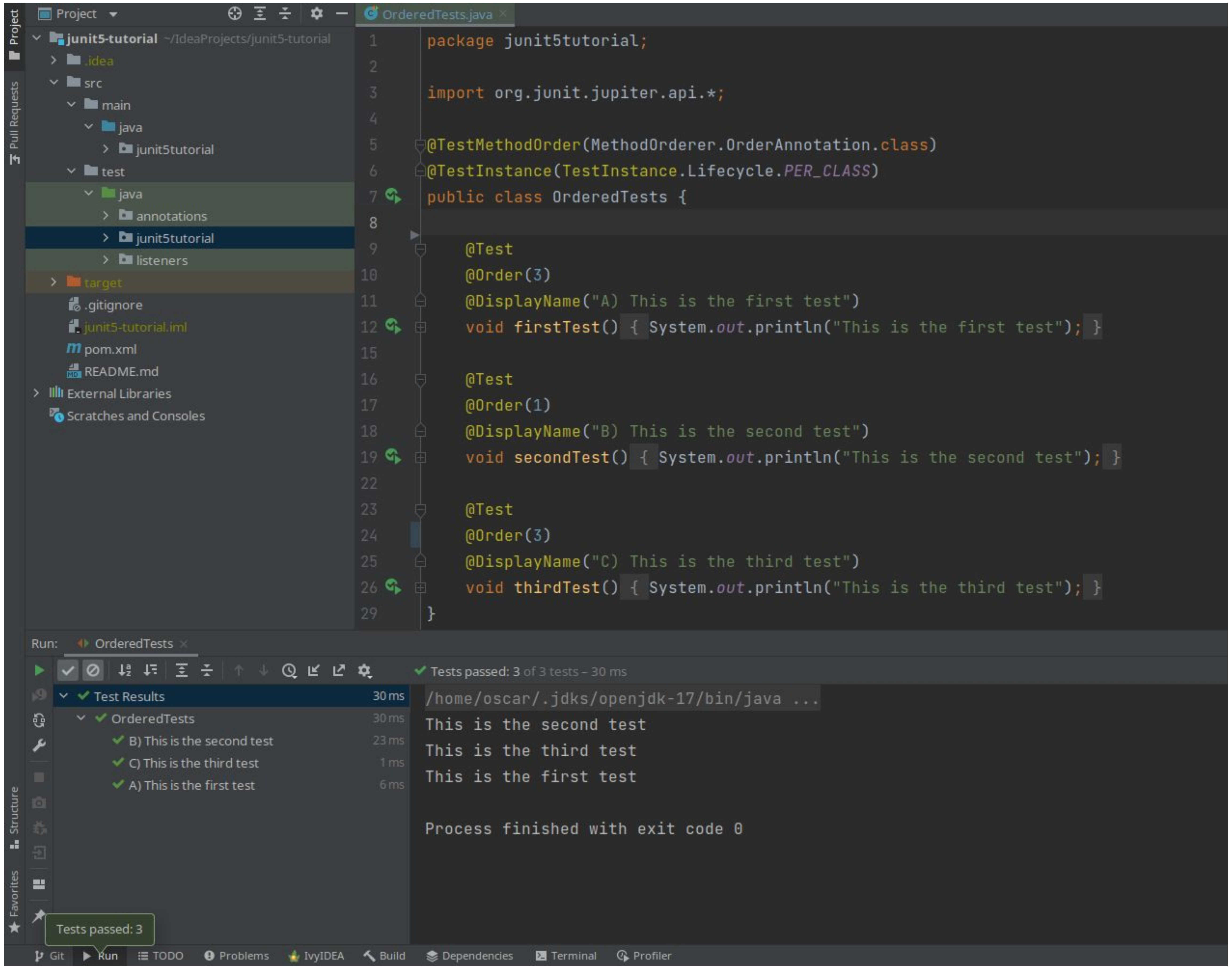


```java
import org.junit.jupiter.params.provider.Arguments;

import java.util.Arrays;
import java.util.List;

import static org.junit.jupiter.params.provider.Arguments.arguments;

public class ParamProvider {

    static List<String> sourceString() {
        return Arrays.asList("cat", "parrot", "dog");
    }

    static List<Arguments> sourceList_StringDouble(){
        return Arrays.asList(arguments("cat", 2.0),arguments("parrot", 5.0),arguments("dog", 3.0));
    }
}
```

6 Ordered tests

- Create a **OrderedTests** Class
- Implement three sample test methods
- Include a **@Order(...)** annotation with different incremental values in your tests
- Run the tests and observe the order they were executed

6 Ordered tests



```java
package junit5tutorial;

import org.junit.jupiter.api.*;

@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
public class OrderedTests {

    @Test
    @Order(3)
    @DisplayName("A) This is the first test")
    void firstTest() { System.out.println("This is the first test"); }

    @Test
    @Order(1)
    @DisplayName("B) This is the second test")
    void secondTest() { System.out.println("This is the second test"); }

    @Test
    @Order(3)
    @DisplayName("C) This is the third test")
    void thirdTest() { System.out.println("This is the third test"); }
}
```

Tests passed: 3 of 3 tests – 30 ms

```
Test Results                      30 ms
  OrderedTests                    30 ms
    B) This is the second test    23 ms
    C) This is the third test      1 ms
    A) This is the first test      6 ms
```

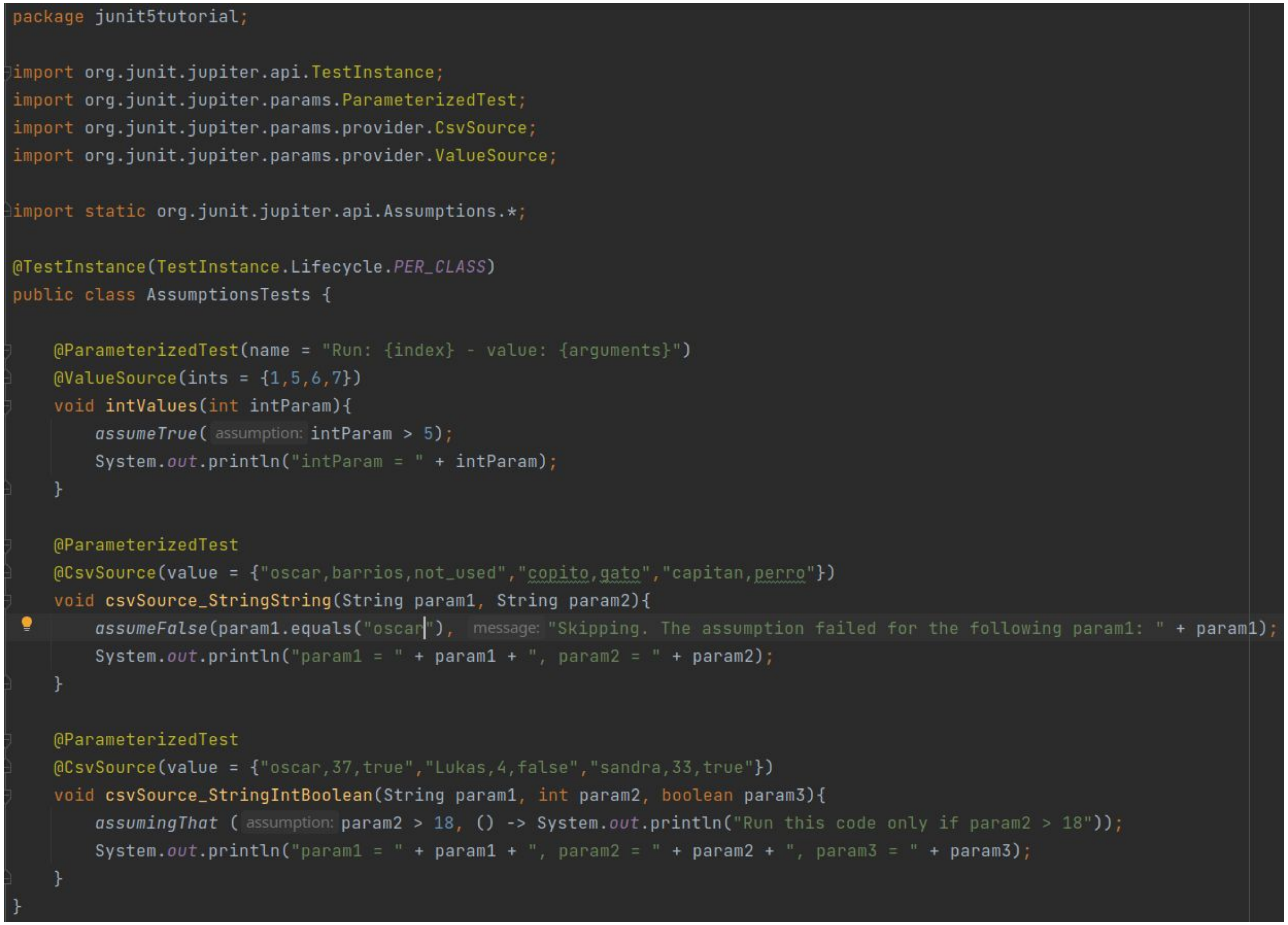
```
/home/oscar/.jdks/openjdk-17/bin/java ...
This is the second test
This is the third test
This is the first test

Process finished with exit code 0
```

Tests passed: 3

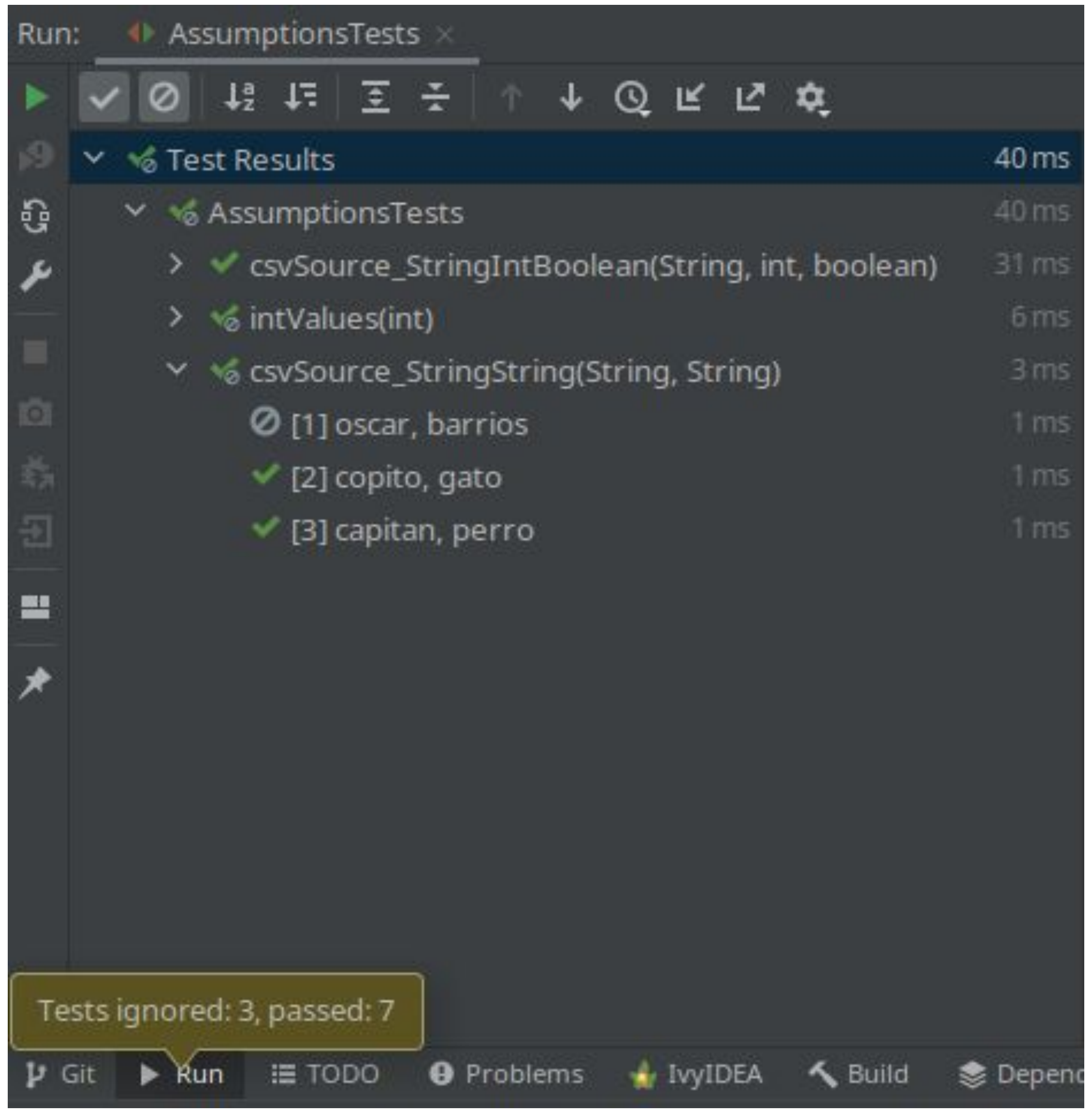
Assumptions (Aborting tests)

- Re-use the **ParameterizedTests** Class and create **AssumptionsTests** Class
- As first line in **stringValues** method, add **assumeTrue(strParam.equals(“three”));**
- Add a static import for **org.junit.jupiter.api.Assumptions.***
- Run this test
- Tests with a value different than **three** are aborted and the rest of lines not executed
- Try **assumeFalse** method
- Try **assumingThat** method. This method will not abort the rest of lines, but just don’t run the executable function passed as parameter.
  - Example: “assumingThat (param > 18, () -> System.out.println("Adult"));

7 Assumptions (Aborting tests)



```java
package junit5tutorial;

import org.junit.jupiter.api.TestInstance;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;
import org.junit.jupiter.params.provider.ValueSource;

import static org.junit.jupiter.api.Assumptions.*;

@TestInstance(TestInstance.Lifecycle.PER_CLASS)
public class AssumptionsTests {

    @ParameterizedTest(name = "Run: {index} - value: {arguments}")
    @ValueSource(ints = {1,5,6,7})
    void intValues(int intParam){
        assumeTrue( assumption: intParam > 5);
        System.out.println("intParam = " + intParam);
    }

    @ParameterizedTest
    @CsvSource(value = {"oscar,barrios,not_used","copito,gato","capitan,perro"})
    void csvSource_StringString(String param1, String param2){
        assumeFalse(param1.equals("oscar"), message: "Skipping. The assumption failed for the following param1: " + param1);
        System.out.println("param1 = " + param1 + ", param2 = " + param2);
    }

    @ParameterizedTest
    @CsvSource(value = {"oscar,37,true","Lukas,4,false","sandra,33,true"})
    void csvSource_StringIntBoolean(String param1, int param2, boolean param3){
        assumingThat ( assumption: param2 > 18, () -> System.out.println("Run this code only if param2 > 18"));
        System.out.println("param1 = " + param1 + ", param2 = " + param2 + ", param3 = " + param3);
    }
}
```



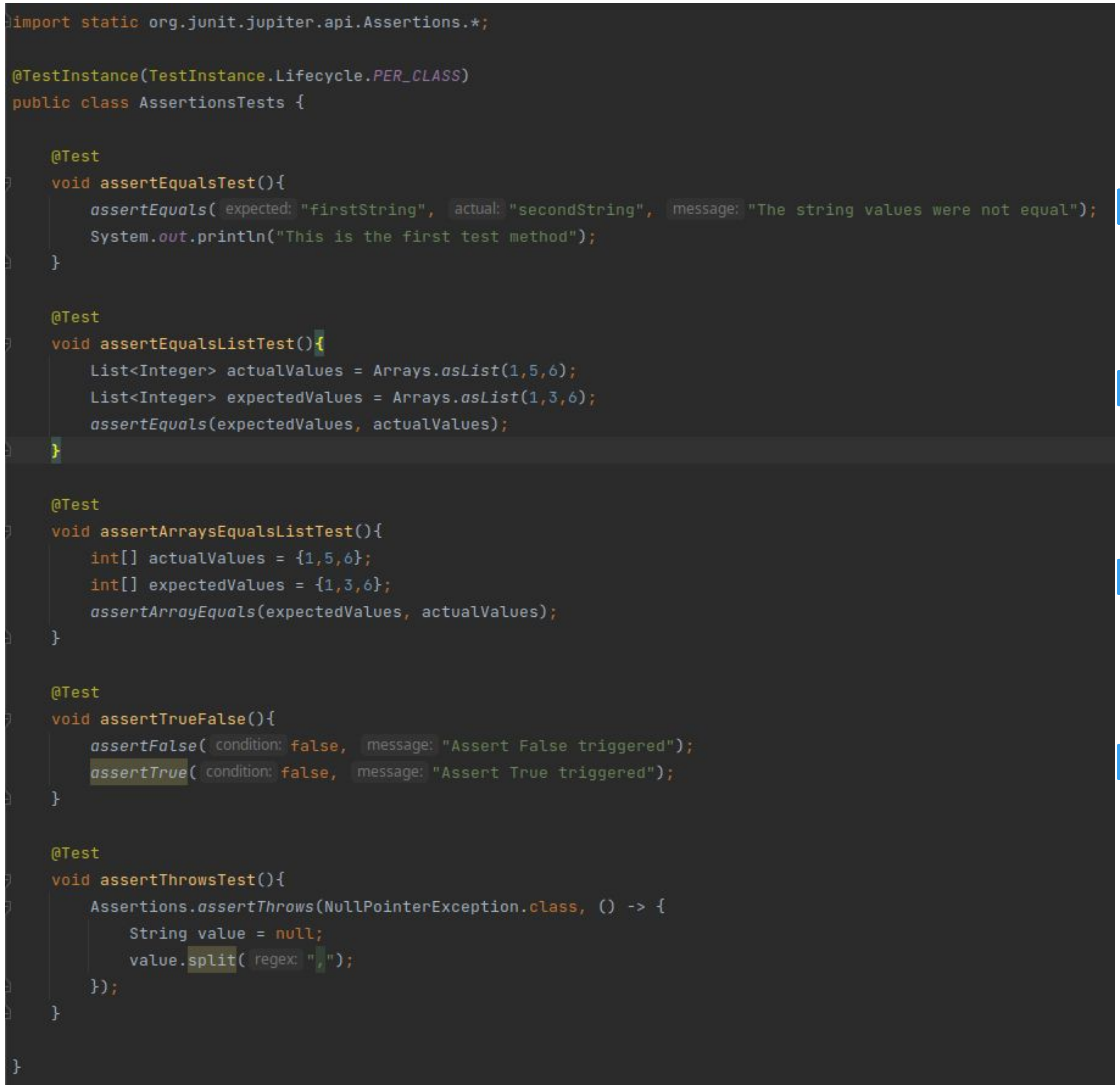
```
Run:  AssumptionsTests
Test Results                                          40 ms
  AssumptionsTests                                    40 ms
    csvSource_StringIntBoolean(String, int, boolean)  31 ms
    intValues(int)                                     6 ms
    csvSource_StringString(String, String)             3 ms
      [1] oscar, barrios                               1 ms
      [2] copito, gato                                 1 ms
      [3] capitan, perro                               1 ms

Tests ignored: 3, passed: 7
```

8 Assertions

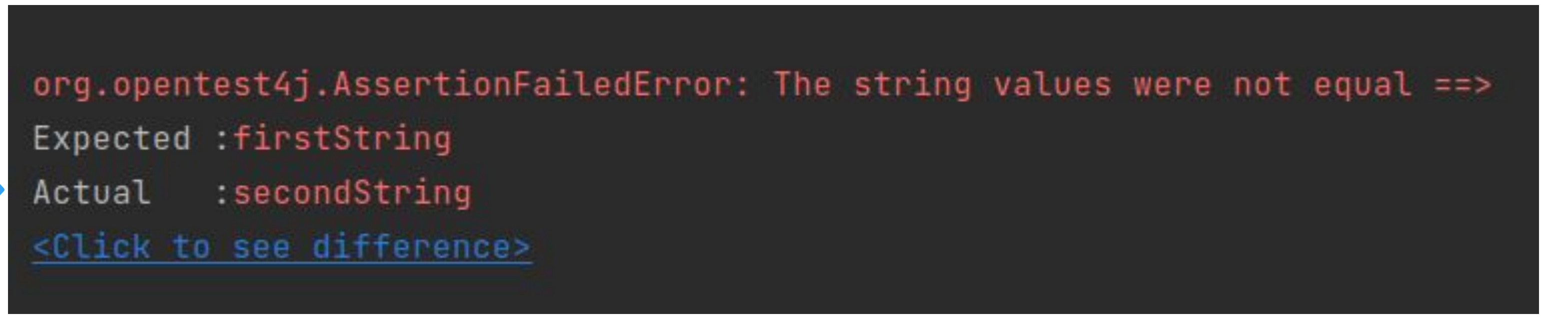
- Create a test class **AssertionsTests**
- **import static org.junit.jupiter.api.Assertions.*;**
- Try **assertEquals**("firstString", "secondString", "The string values were not equal");
- Try again but passing two List<Integer> as parameters
- Try again with two int[] arrays, using **assertArrayEquals**
- Try **assertFalse** and **assertTrue**
- Try **assertThrows** to assert if your executable function throws an expected exception.
  - Example:
    assertThrows(NullPointerException.class, () -> { String value = null; value.split(","); });

8

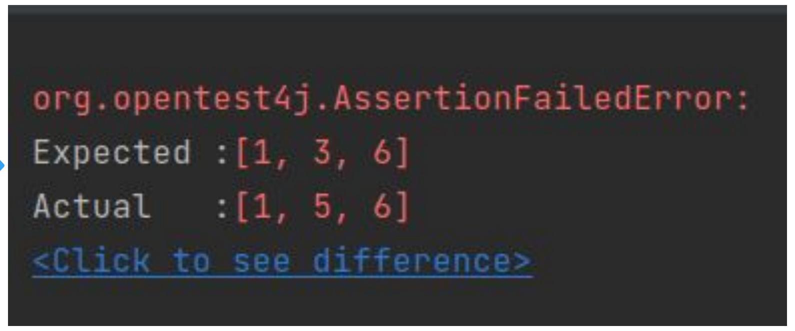
Assertions



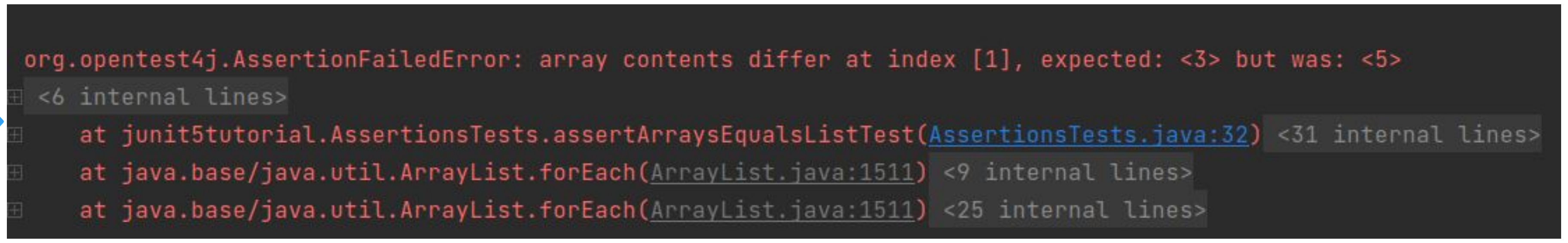
```java
import static org.junit.jupiter.api.Assertions.*;

@TestInstance(TestInstance.Lifecycle.PER_CLASS)
public class AssertionsTests {

    @Test
    void assertEqualsTest(){
        assertEquals( expected: "firstString",  actual: "secondString",  message: "The string values were not equal");
        System.out.println("This is the first test method");
    }

    @Test
    void assertEqualsListTest(){
        List<Integer> actualValues = Arrays.asList(1,5,6);
        List<Integer> expectedValues = Arrays.asList(1,3,6);
        assertEquals(expectedValues, actualValues);
    }

    @Test
    void assertArraysEqualsListTest(){
        int[] actualValues = {1,5,6};
        int[] expectedValues = {1,3,6};
        assertArrayEquals(expectedValues, actualValues);
    }

    @Test
    void assertTrueFalse(){
        assertFalse( condition: false,  message: "Assert False triggered");
        assertTrue( condition: false,  message: "Assert True triggered");
    }

    @Test
    void assertThrowsTest(){
        Assertions.assertThrows(NullPointerException.class, () -> {
            String value = null;
            value.split( regex: ",");
        });
    }

}
```



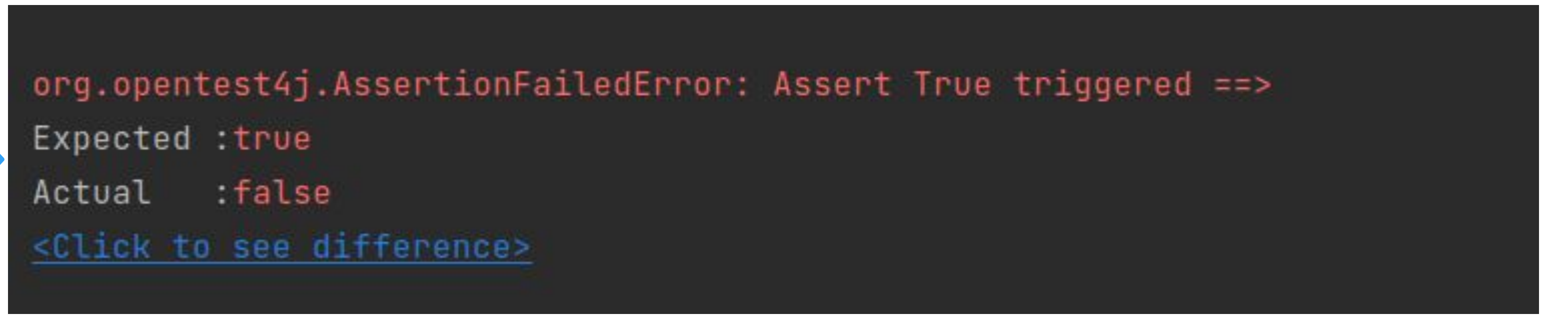
```
org.opentest4j.AssertionFailedError: The string values were not equal ==>
Expected :firstString
Actual   :secondString
<Click to see difference>
```



```
org.opentest4j.AssertionFailedError:
Expected :[1, 3, 6]
Actual   :[1, 5, 6]
<Click to see difference>
```



```
org.opentest4j.AssertionFailedError: array contents differ at index [1], expected: <3> but was: <5>
 <6 internal lines>
    at junit5tutorial.AssertionsTests.assertArraysEqualsListTest(AssertionsTests.java:32) <31 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <9 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <25 internal lines>
```



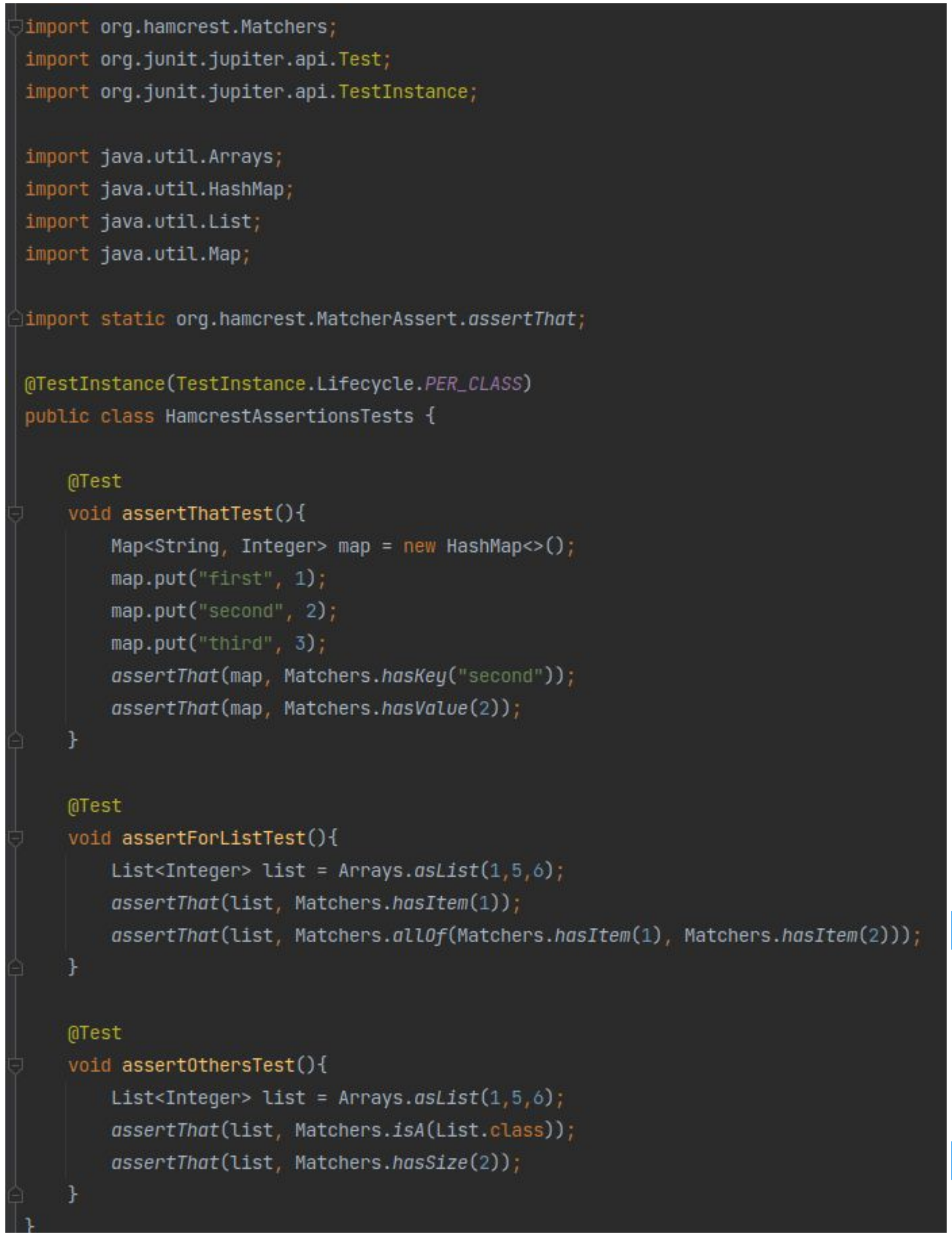
```
org.opentest4j.AssertionFailedError: Assert True triggered ==>
Expected :true
Actual   :false
<Click to see difference>
```

Hamcrest assertions

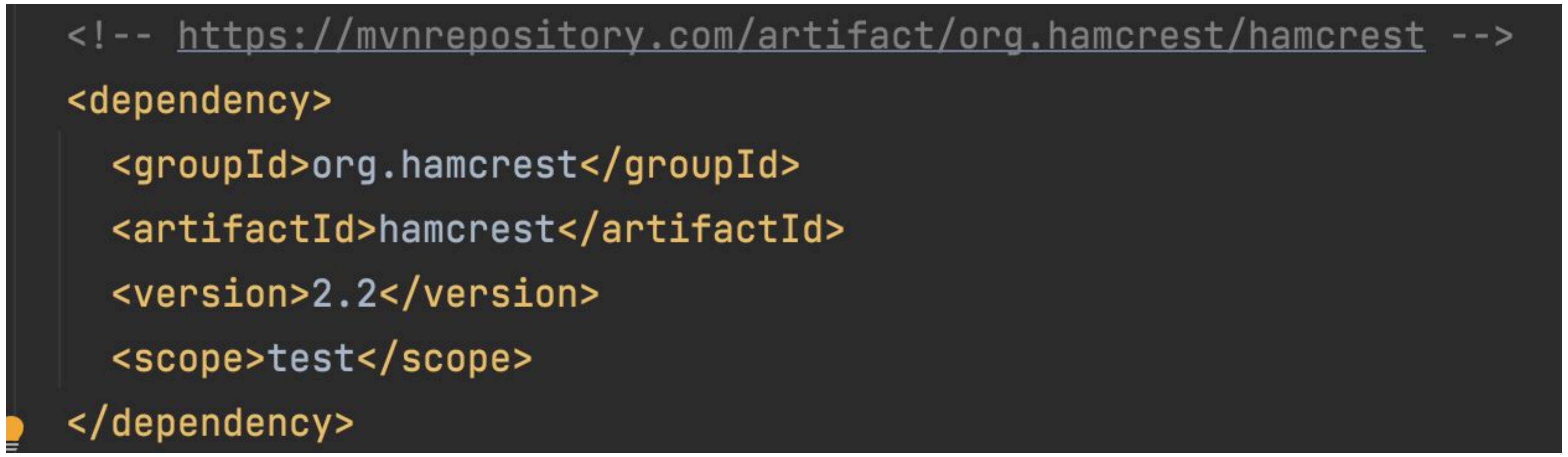
- Add Maven dependency: https://mvnrepository.com/artifact/org.hamcrest/hamcrest/2.2
- Create a test class **HamcrestAssertionsTests**
- **import static org.hamcrest.MatcherAssert.assertThat;**
- Try **assertThat** passing a **Map<String, Integer>**
- As matcher use: assertThat(map, **Matchers.hasKey**("second"));
- As matcher use: assertThat(map, **Matchers.hasValue**(2));
- Try with List<Integer>
- Try **Matchers.hasItem** or **Matchers.allOf**(Matchers.hasItem(1), Matchers.hasItem(2))
- Play with other Matchers like **Matchers.isA** and **Matchers.hasSize**

9

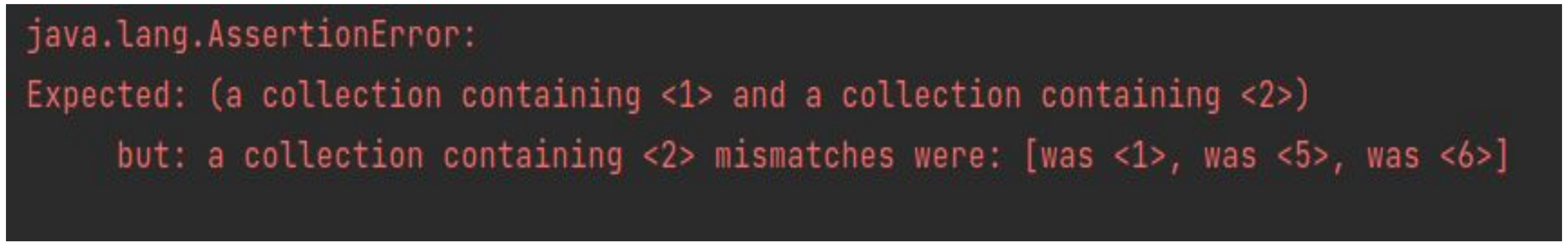
Hamcrest assertions



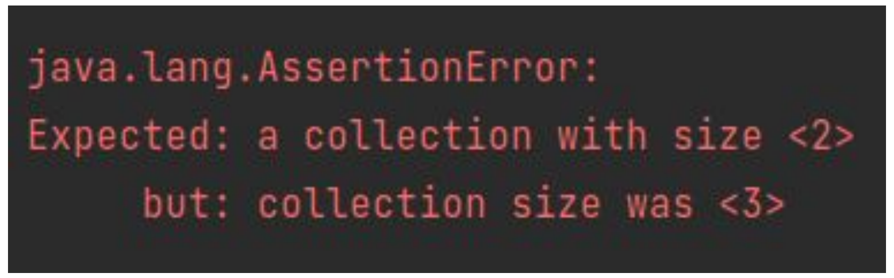
```java
import org.hamcrest.Matchers;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestInstance;

import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import static org.hamcrest.MatcherAssert.assertThat;

@TestInstance(TestInstance.Lifecycle.PER_CLASS)
public class HamcrestAssertionsTests {

    @Test
    void assertThatTest(){
        Map<String, Integer> map = new HashMap<>();
        map.put("first", 1);
        map.put("second", 2);
        map.put("third", 3);
        assertThat(map, Matchers.hasKey("second"));
        assertThat(map, Matchers.hasValue(2));
    }

    @Test
    void assertForListTest(){
        List<Integer> list = Arrays.asList(1,5,6);
        assertThat(list, Matchers.hasItem(1));
        assertThat(list, Matchers.allOf(Matchers.hasItem(1), Matchers.hasItem(2)));
    }

    @Test
    void assertOthersTest(){
        List<Integer> list = Arrays.asList(1,5,6);
        assertThat(list, Matchers.isA(List.class));
        assertThat(list, Matchers.hasSize(2));
    }
}
```



```xml
<!-- https://mvnrepository.com/artifact/org.hamcrest/hamcrest -->
<dependency>
  <groupId>org.hamcrest</groupId>
  <artifactId>hamcrest</artifactId>
  <version>2.2</version>
  <scope>test</scope>
</dependency>
```



```
java.lang.AssertionError:
Expected: (a collection containing <1> and a collection containing <2>)
     but: a collection containing <2> mismatches were: [was <1>, was <5>, was <6>]
```

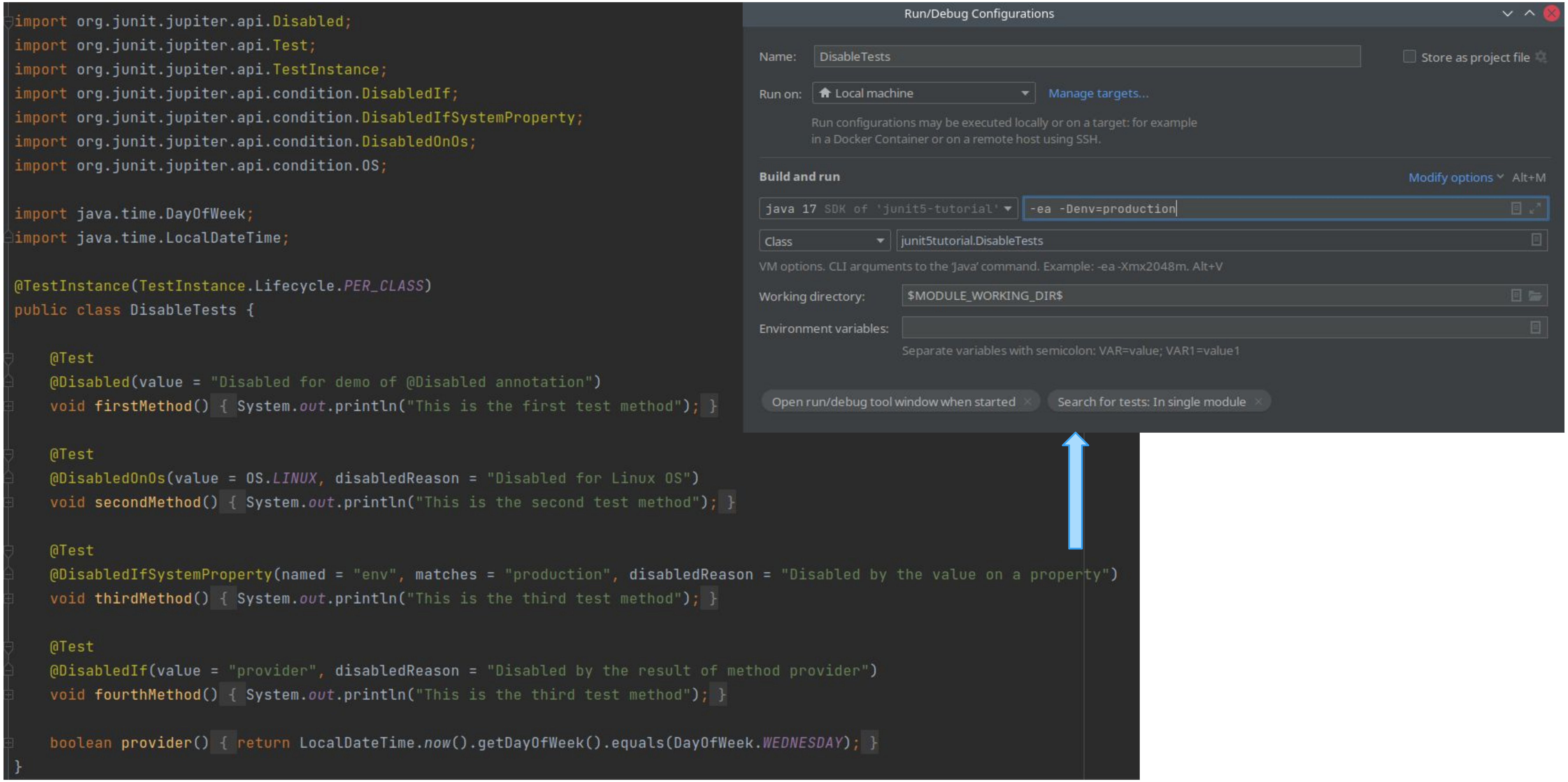


```
java.lang.AssertionError:
Expected: a collection with size <2>
     but: collection size was <3>
```

Disabled tests

- Create a test class **DisableTests**
- Use **@Disabled(value = "Disabled for demo of @Disabled annotation")** in a new test
- Observe that this test has not been executed
- Try **@DisabledOnOs(value = OS.LINUX, disabledReason = "Disabled for Linux OS")**
- Edit configuration -> VM Options -> **“-ea -Denv=production”**
- Try **@DisabledIfSystemProperty(named = "env", matches = "production", disabledReason = "Disabled by the value on a property")**
- boolean provider(){ return LocalDateTime.now().getDayOfWeek().equals(DayOfWeek.WEDNESDAY); }
- Try **@DisabledIf(value = "provider", disabledReason = "Disabled by the result of method provider")**

10 Disabled tests



```java
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestInstance;
import org.junit.jupiter.api.condition.DisabledIf;
import org.junit.jupiter.api.condition.DisabledIfSystemProperty;
import org.junit.jupiter.api.condition.DisabledOnOs;
import org.junit.jupiter.api.condition.OS;

import java.time.DayOfWeek;
import java.time.LocalDateTime;

@TestInstance(TestInstance.Lifecycle.PER_CLASS)
public class DisableTests {

    @Test
    @Disabled(value = "Disabled for demo of @Disabled annotation")
    void firstMethod() { System.out.println("This is the first test method"); }

    @Test
    @DisabledOnOs(value = OS.LINUX, disabledReason = "Disabled for Linux OS")
    void secondMethod() { System.out.println("This is the second test method"); }

    @Test
    @DisabledIfSystemProperty(named = "env", matches = "production", disabledReason = "Disabled by the value on a property")
    void thirdMethod() { System.out.println("This is the third test method"); }

    @Test
    @DisabledIf(value = "provider", disabledReason = "Disabled by the result of method provider")
    void fourthMethod() { System.out.println("This is the third test method"); }

    boolean provider() { return LocalDateTime.now().getDayOfWeek().equals(DayOfWeek.WEDNESDAY); }
}
```

Run/Debug Configurations

Name: DisableTests Store as project file

Run on: Local machine Manage targets...

Run configurations may be executed locally or on a target: for example in a Docker Container or on a remote host using SSH.

Build and run Modify options Alt+M

java 17 SDK of 'junit5-tutorial' `-ea -Denv=production`

Class junit5tutorial.DisableTests

VM options. CLI arguments to the 'java' command. Example: -ea -Xmx2048m. Alt+V

Working directory: $MODULE_WORKING_DIR$

Environment variables:

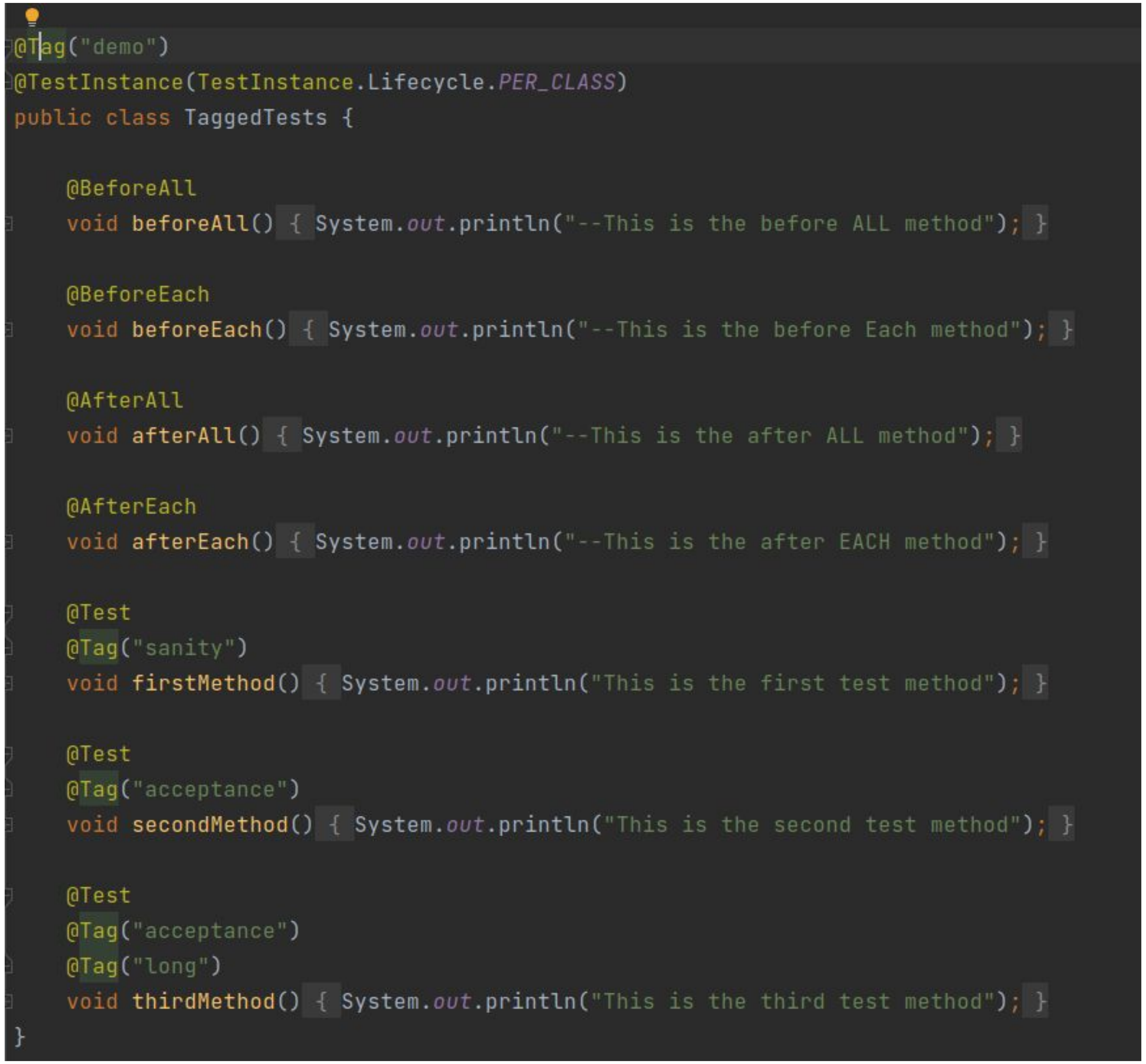
Separate variables with semicolon: VAR=value; VAR1=value1

Open run/debug tool window when started Search for tests: In single module

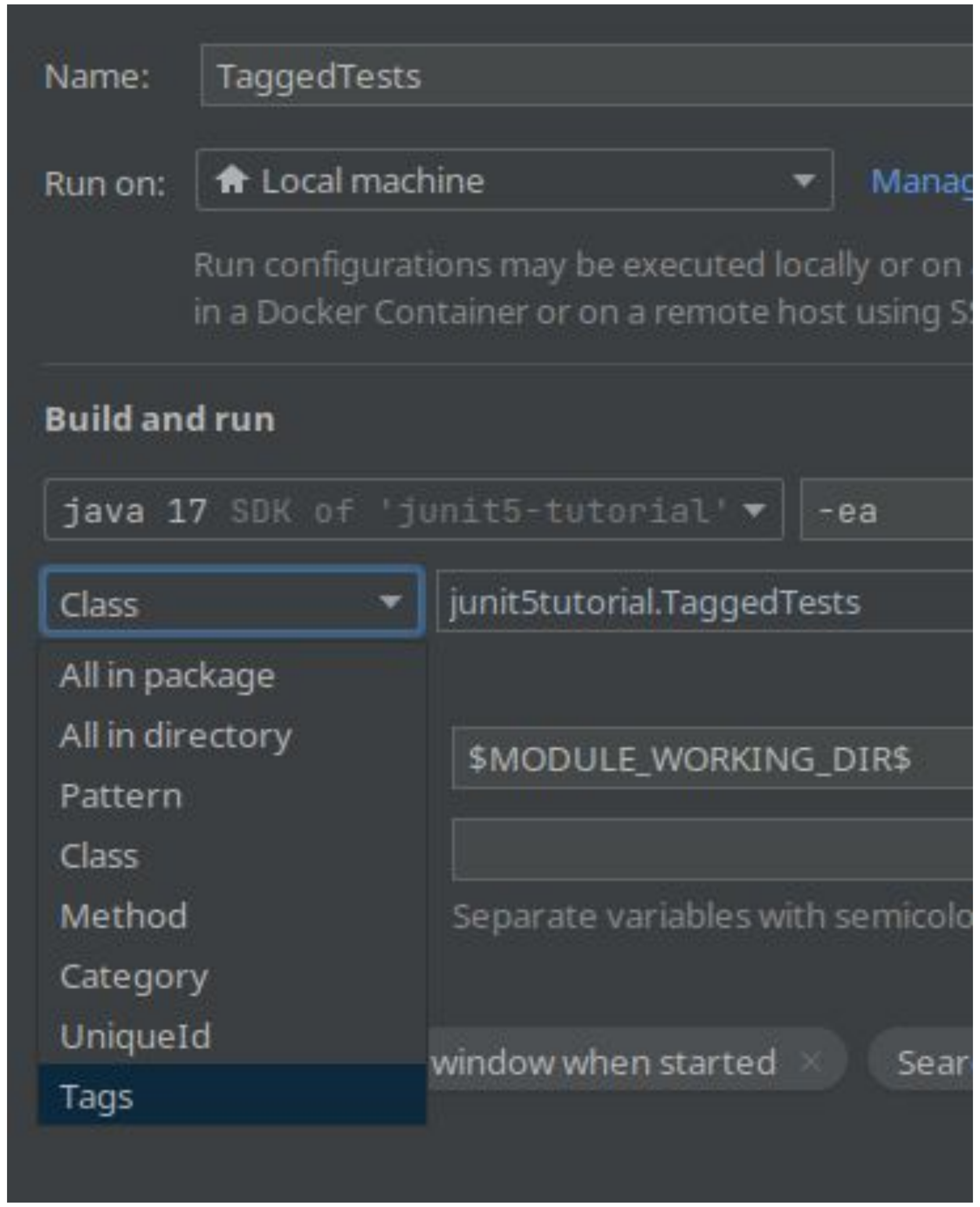
Tags

- Create a test class **TaggedTests**
- Write three test methods
- Add a **@Tag(“api”)** annotation on two of them
- Add a **@Tag(“database”)** annotation in the last test
- You can also try adding a Tag annotation at class level
- Edit configuration -> Build and run -> Change the type of resource (by default Method or Class) -> Select Tags -> Write “api” -> Run and Observe
- Try operands: **“api & database”**, **“api | database”** and **“!api”**
- Try from a console with maven: **mvn test -Dgroups="database"**

11 **Tags**



```java
@Tag("demo")
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
public class TaggedTests {

    @BeforeAll
    void beforeAll() { System.out.println("--This is the before ALL method"); }

    @BeforeEach
    void beforeEach() { System.out.println("--This is the before Each method"); }

    @AfterAll
    void afterAll() { System.out.println("--This is the after ALL method"); }

    @AfterEach
    void afterEach() { System.out.println("--This is the after EACH method"); }

    @Test
    @Tag("sanity")
    void firstMethod() { System.out.println("This is the first test method"); }

    @Test
    @Tag("acceptance")
    void secondMethod() { System.out.println("This is the second test method"); }

    @Test
    @Tag("acceptance")
    @Tag("long")
    void thirdMethod() { System.out.println("This is the third test method"); }
}
```



Name: TaggedTests

Run on: Local machine Manag

Run configurations may be executed locally or on
in a Docker Container or on a remote host using S

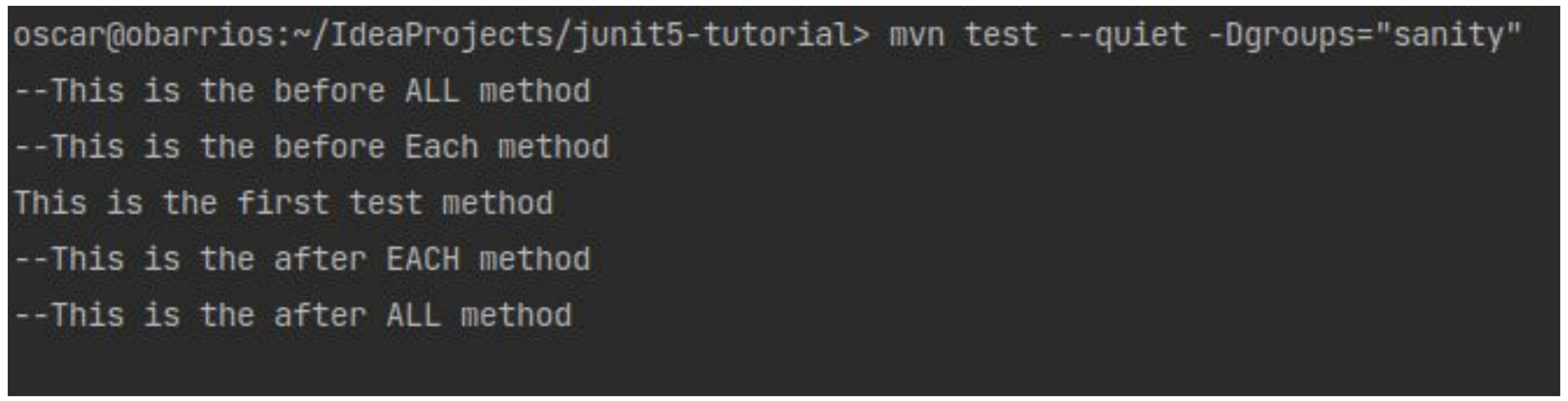
Build and run

java 17 SDK of 'junit5-tutorial' -ea

Class junit5tutorial.TaggedTests

$MODULE_WORKING_DIR$

Separate variables with semicolo



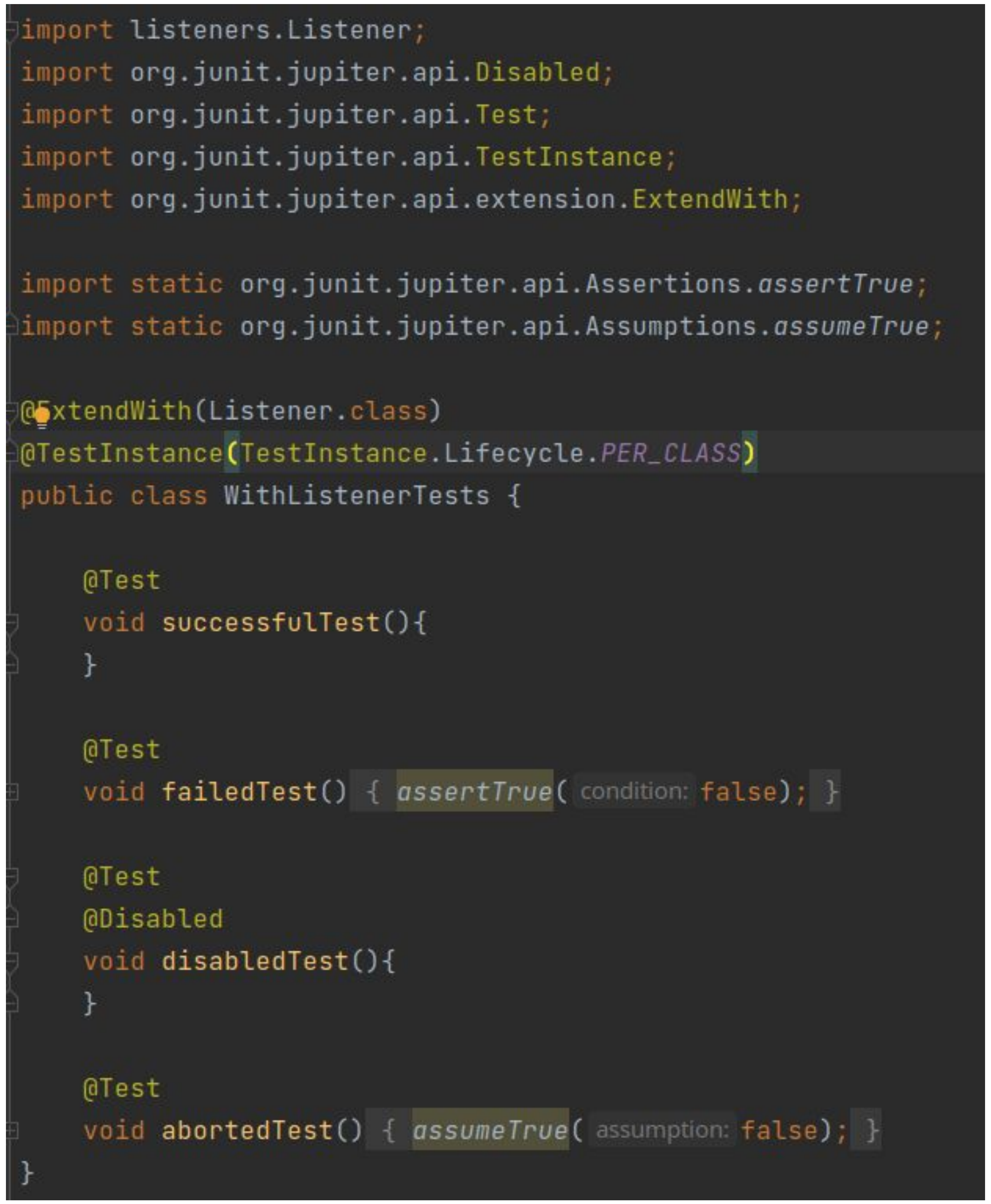
```
oscar@obarrios:~/IdeaProjects/junit5-tutorial> mvn test --quiet -Dgroups="sanity"
--This is the before ALL method
--This is the before Each method
This is the first test method
--This is the after EACH method
--This is the after ALL method
```

Test Watcher (Listener)

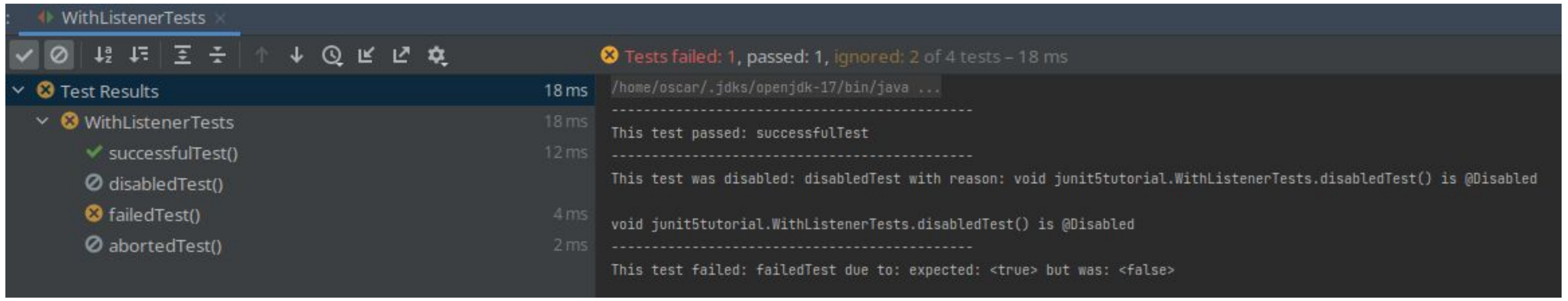
- Create a test class **WithListenerTests**
- Include the **@ExtendWith(Listener.class)** annotation at class level
- Create a class **Listener** which implements the interface **TestWatcher**
- Override all the methods for this interface include a printed output
- Write tests methods which final state is **successful, failed, disabled** and **aborted**
- Run these tests and observe that the methods in your **Listener** are executed

12

Test Watcher (Listener)



```java
import listeners.Listener;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestInstance;
import org.junit.jupiter.api.extension.ExtendWith;

import static org.junit.jupiter.api.Assertions.assertTrue;
import static org.junit.jupiter.api.Assumptions.assumeTrue;

@ExtendWith(Listener.class)
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
public class WithListenerTests {

    @Test
    void successfulTest(){
    }

    @Test
    void failedTest() { assertTrue( condition: false); }

    @Test
    @Disabled
    void disabledTest(){
    }

    @Test
    void abortedTest() { assumeTrue( assumption: false); }
}
```

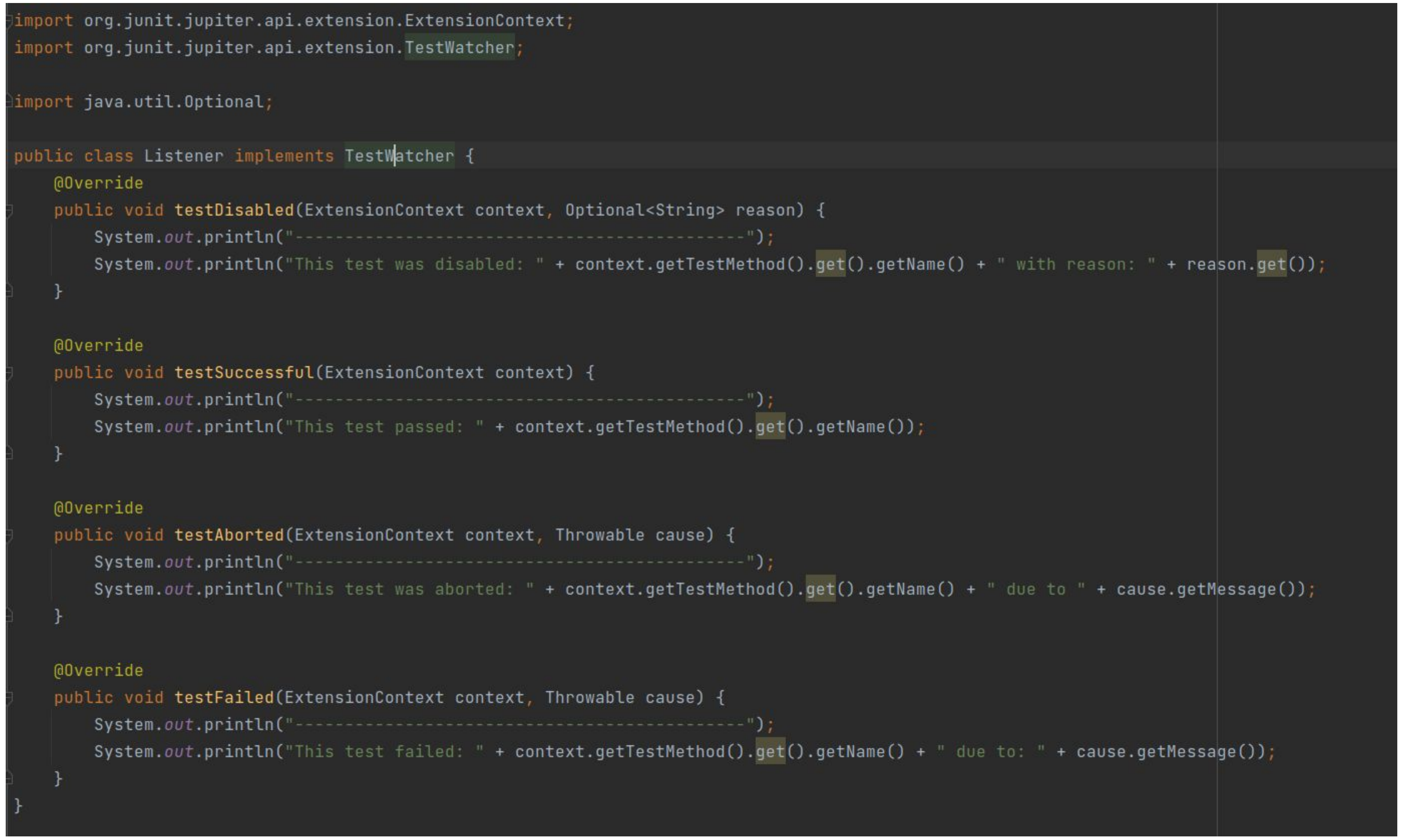


WithListenerTests

Tests failed: 1, passed: 1, ignored: 2 of 4 tests – 18 ms

```
Test Results                18 ms
  WithListenerTests         18 ms
    successfulTest()        12 ms
    disabledTest()
    failedTest()             4 ms
    abortedTest()            2 ms
```

```
/home/oscar/.jdks/openjdk-17/bin/java ...
---------------------------------------------
This test passed: successfulTest
---------------------------------------------
This test was disabled: disabledTest with reason: void junit5tutorial.WithListenerTests.disabledTest() is @Disabled

void junit5tutorial.WithListenerTests.disabledTest() is @Disabled
---------------------------------------------
This test failed: failedTest due to: expected: <true> but was: <false>
```



```java
import org.junit.jupiter.api.extension.ExtensionContext;
import org.junit.jupiter.api.extension.TestWatcher;

import java.util.Optional;

public class Listener implements TestWatcher {
    @Override
    public void testDisabled(ExtensionContext context, Optional<String> reason) {
        System.out.println("---------------------------------------------");
        System.out.println("This test was disabled: " + context.getTestMethod().get().getName() + " with reason: " + reason.get());
    }

    @Override
    public void testSuccessful(ExtensionContext context) {
        System.out.println("---------------------------------------------");
        System.out.println("This test passed: " + context.getTestMethod().get().getName());
    }

    @Override
    public void testAborted(ExtensionContext context, Throwable cause) {
        System.out.println("---------------------------------------------");
        System.out.println("This test was aborted: " + context.getTestMethod().get().getName() + " due to " + cause.getMessage());
    }

    @Override
    public void testFailed(ExtensionContext context, Throwable cause) {
        System.out.println("---------------------------------------------");
        System.out.println("This test failed: " + context.getTestMethod().get().getName() + " due to: " + cause.getMessage());
    }
}
```

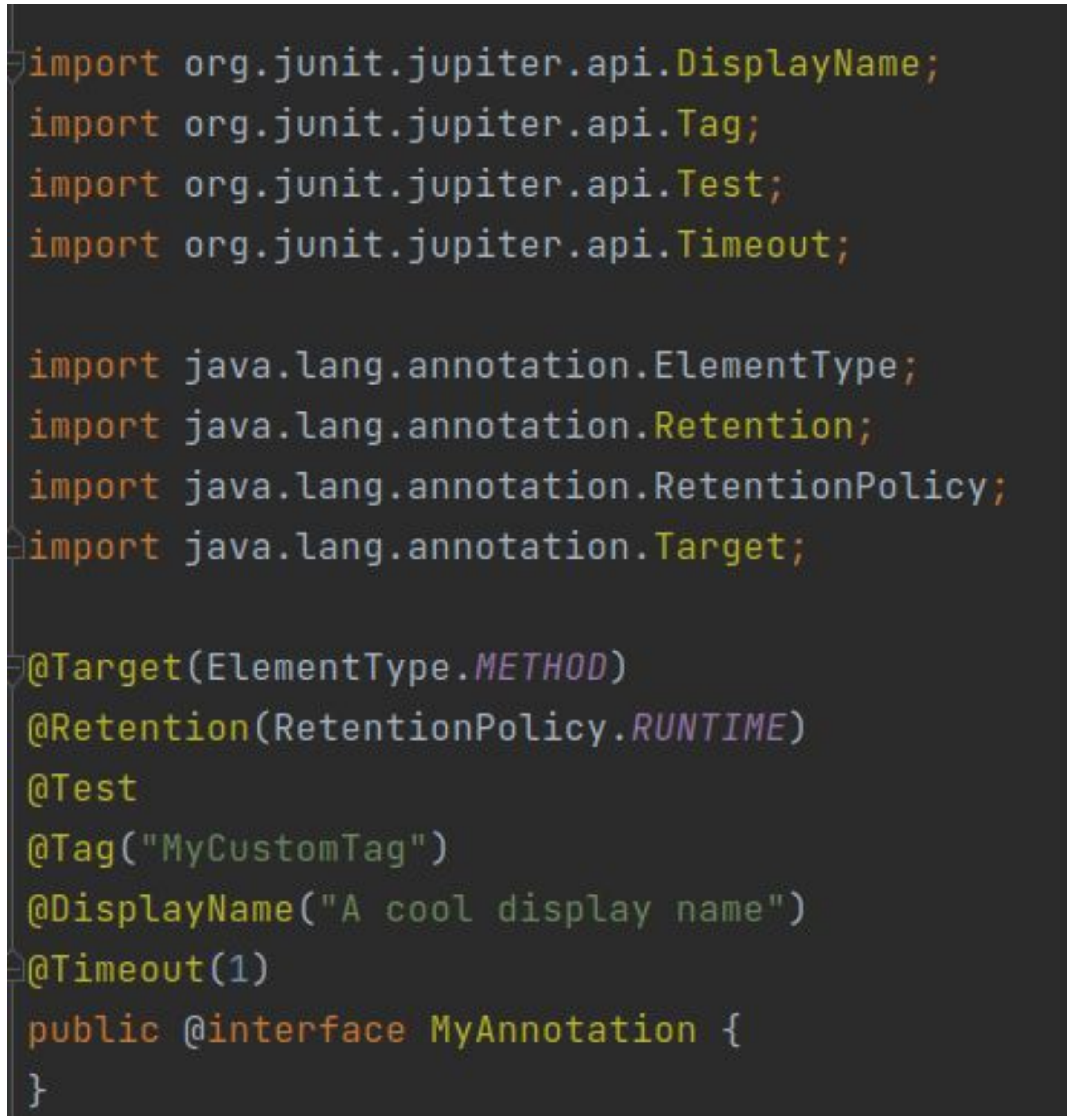
13 Timeout a test

- Create a test class **TimeoutTests**
- Write a test with **@Timeout(value = 1500, unit = TimeUnit.MILLISECONDS)**
- Print a text in the test
- Add a **Thread.sleep(3000);**
- Run and observe
- Try with a different unit
- Try without unit parameter (seconds as default)

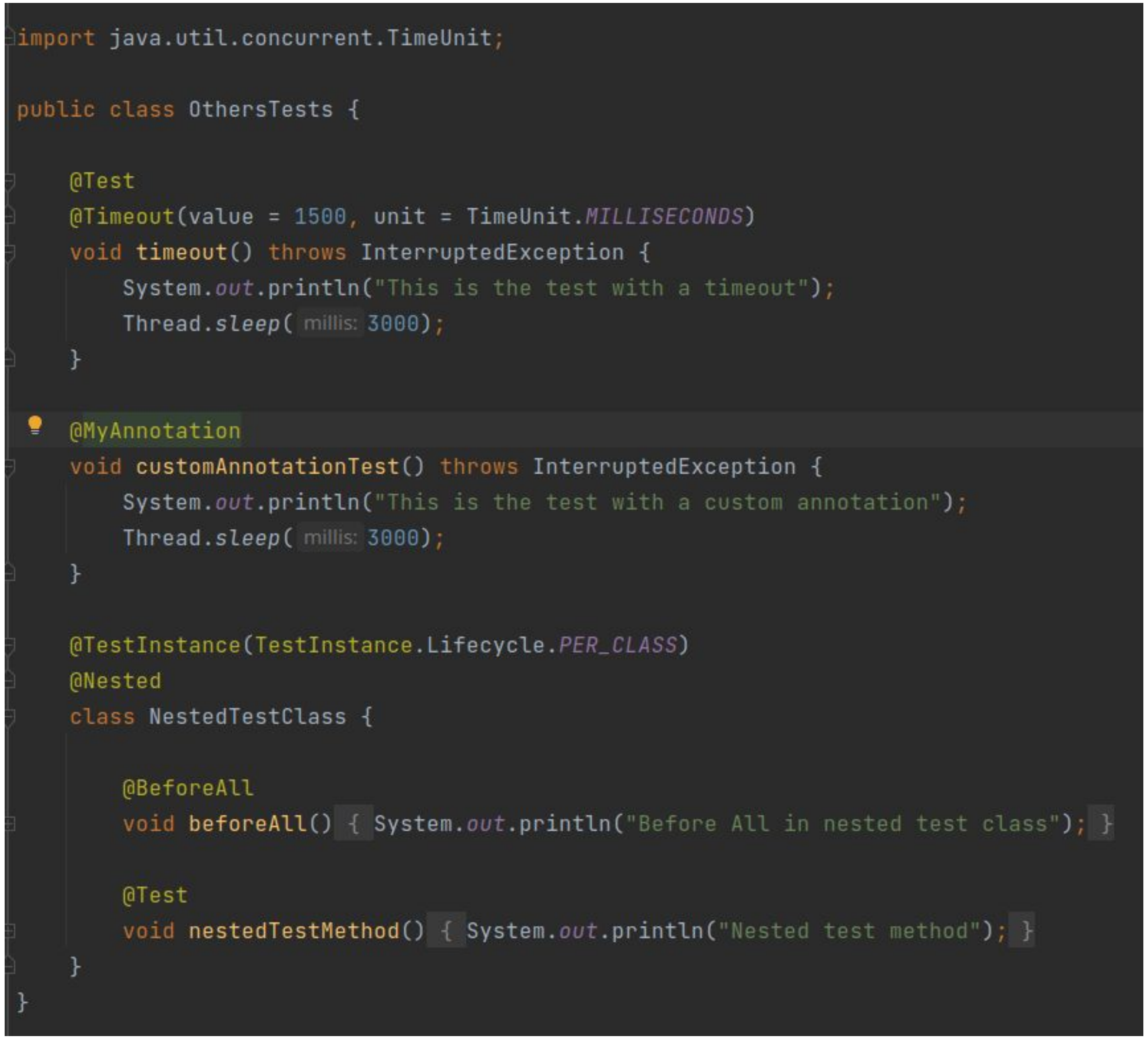
Custom annotation

- Create an interface **public @interface MyAnnotation {}**
- Add **@Target(ElementType.METHOD)** annotation
- Add **@Retention(RetentionPolicy.RUNTIME)** annotation
- Add **@Test** annotation
- Add other annotations like a **Tag, DisplayName, Timeout, and so on**
- Create a test class **CustomAnnotationTests**
- Use your annotation in one of your tests, then run the test and observe how it behaves
- Curious about Java Annotations? https://en.wikipedia.org/wiki/Java_annotation

Custom annotation



```java
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Tag;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Timeout;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@Test
@Tag("MyCustomTag")
@DisplayName("A cool display name")
@Timeout(1)
public @interface MyAnnotation {
}
```



```java
import java.util.concurrent.TimeUnit;

public class OthersTests {

    @Test
    @Timeout(value = 1500, unit = TimeUnit.MILLISECONDS)
    void timeout() throws InterruptedException {
        System.out.println("This is the test with a timeout");
        Thread.sleep( millis: 3000);
    }

    @MyAnnotation
    void customAnnotationTest() throws InterruptedException {
        System.out.println("This is the test with a custom annotation");
        Thread.sleep( millis: 3000);
    }

    @TestInstance(TestInstance.Lifecycle.PER_CLASS)
    @Nested
    class NestedTestClass {

        @BeforeAll
        void beforeAll() { System.out.println("Before All in nested test class"); }

        @Test
        void nestedTestMethod() { System.out.println("Nested test method"); }
    }
}
```

Course code

https://github.com/srbarrios/junit5-tutorial

Run your unit tests easily

JUnit 5
Test Framework

Óscar Barrios

Test Automation Specialist @SUSE