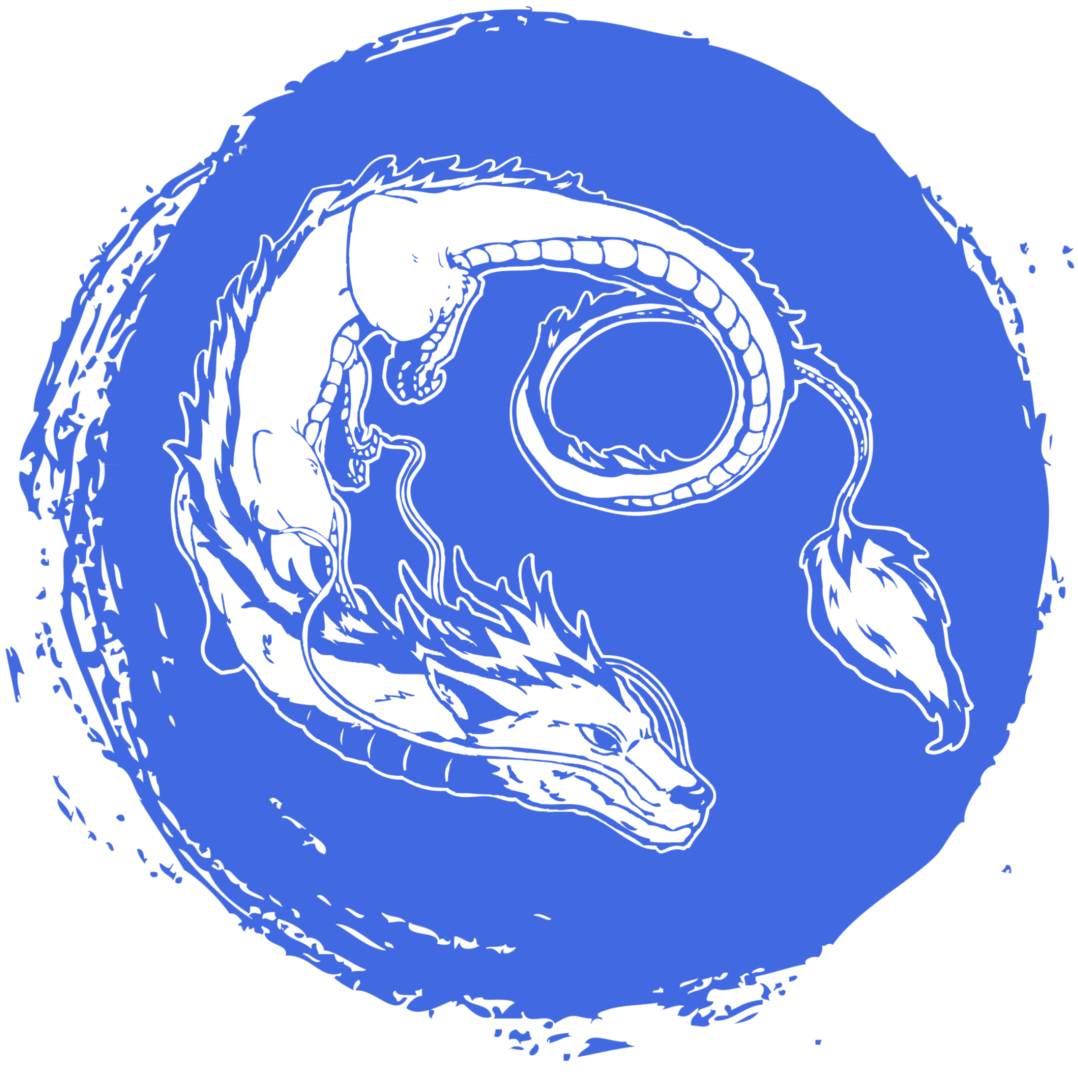
# Java Workshop — Junit

version **#v1.1.0**



**Yet Another Kind of Assistants 2022** <yaka@tickets.assistants.epita.fr>

## Copyright

## Contents

*https://intra.assistants.epita.fr

## 1 Presentation

*JUnit* is an open source framework used to run **unit tests** on Java code. The main difference with your previous tests is the **granularity**: JUnit tests are designed to test only one **unit** at once (for instance, the behavior of a single class, or even a single method), while you were previously testing your program as a whole. JUnit defines **two types** of test files:

- **TestCase**: A class containing test methods. One `TestCase` usually tests the behavior of a whole class.
- **TestSuite**: Used to unify and execute `TestCases`.

**TestCases** are divided up in three parts:

1. Creation of a class instance (and everything else necessary for tests).
2. Call the test method.
3. Comparison of the expected result with the real result.

## 2 JUnit's annotations

JUnit 5.x uses annotations that you can combine to configure the test run:

- `@Test`: Identifies a method as a test method;
- `@Timeout(100)`: Fails if the method takes longer than 100 milliseconds;

```java
@Test
@Timeout(100)
public void someTest() {
    assertThrows(ArithmeticException.class, () -> 10/0);
}
```

- `@BeforeEach`: The method is executed before each test. It is used to prepare the test environment (e.g., read input data, initialize the class);

```java
@BeforeEach
public void setUp() {
    // Code executed before each test
}
```

- `@AfterEach`: This method is executed after each test. It is used to clean up the test environment (e.g., delete temporary data, restore defaults);

```java
@AfterEach
public void tearDown() {
    // Code executed after each test
}
```

- `@BeforeAll`: This method is executed once, before the start of all tests. It is used to perform time intensive activities, for example, to connect to a database. Methods marked with this annotation need to be defined as **static** to work with JUnit;

```
@BeforeAll
public static void setUpClass() {
    // Code executed before the first test
}
```

- `@AfterAll`: This method is executed once, after all tests have been finished. It is used to perform clean-up activities, for example, to disconnect from a database. Methods annotated with this annotation need to be defined as **static** to work with JUnit;

```
@AfterAll
public static void tearDownClass() {
    // Code executed after all tests
}
```

- `@Disabled`: Disables a test class or a test method. This is useful when the underlying code has been changed and the test case has not yet been adapted. Or if the execution time of this test is too long to be included;
- `@DisplayName("Test description")`: Add a custom description displayed in test reports.

All those annotations (and those you can find in the JUnit documentation) are located in the `junit-jupiter-api` module.

## 3 Assertions

### 3.1 The 'assert' keyword

Java provides an assert keyword with the following syntax:

```
assert condition [:object];
```

Condition is a boolean, and object is optional, it could be any object containing information about assertion failure.

By default, assert is disabled. To enable it, you must use the flag `-ea` when calling Java. In IntelliJ IDEA, you can change it in `Run > Edit configuration` and then add `-ea` to the `Vm options` field.

#### 3.1.1 Example

```
assert y != 0 : "y cannot be zero";
a = x / y;
```

### 3.1.2 JUnit assertions

JUnit provides static methods in the org.junit.jupiter.api.Assertions class to test certain conditions, arguments between braces are optional:

- `assertTrue(boolean condition, [String message])`: Checks that the boolean condition is true;
- `assertFalse(boolean condition, [String message])`: Checks that the boolean condition is false;
- `assertEquals(expected, actual, [String message])`: Tests that two values are the same. Note: for arrays the reference is checked not the content of the arrays;
- `assertNotEquals(unexpected, actual, [String message])`: Tests that two values are not equal;
- `assertArrayEquals(expectedArray, actualArray, [String message])`: Checks that the two arrays are equal;
- `assertNull(Object actual, [String message])`: Checks that the object is null;
- `assertNotNull(Object actual, [String message])`: Checks that the object is not null;
- `assertSame(expected, actual, [String message])`: Checks that both variables refer to the same object;
- `assertNotSame(unexpected, actual, [String message])`: Checks that both variables refer to different objects;
- `assertThrows(expectedType, executable, [String message])`: Checks that the given executable throws an exception of the given expectedType;
- `fail(String message)`: Lets the method fail. Might be used to check that a certain part of the code is not reached or to have a failing test before the test code is implemented. The `String` parameter is optional.

## 3.2 With your IDE, your best friend

With any IDE, you can easily get the result of your tests: simply run your program as a JUnit Test with `Run as … > JUnit Test`, or the local equivalent. By default, you will have a red bar in case of error, and a simple message in case of success.

To run tests, just add this in the dependencies section in your `pom.xml` file:

```
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>5.7.1</version>
    <scope>test</scope>
</dependency>
```

You can also add JUnit dependencies to your `pom.xml` by pressing `Alt + insert`, select `dependency` and enter `junit` in the search bar. Select the `junit-jupiter` artifact (your IDE should suggest the most recent version). This artifact will pull automatically other artifacts that you need such as `junit-jupiter-api` for the anotations for example.

### 3.2.1 Example

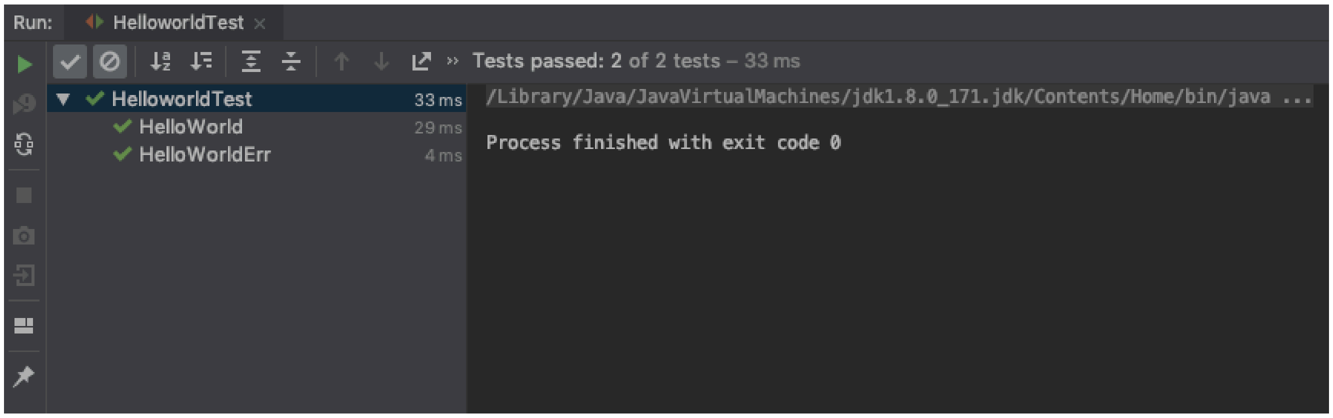
```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class TestFoobar {

    @Test
    public void booleanTest() {
        // Checks that the boolean condition is true.
        assertTrue(condition, [message]);
    }

    @Test
    public void equalTest() {
        // Tests that two values are the same.
        assertsEquals(expected, actual, [message]);
    }

    @Test
    public void nullTest() {
        // Checks that the object is null.
        assertNull(object, [message]);
    }

    @Test
    public void sameTest() {
        // Checks that both variables refer to the same object.
        assertSame(expected, actual, [message]);
    }
}
```

### 3.2.2 Running Tests

With Maven, you can easily run your tests.

```bash
# Run all the unit test classes.
$ mvn test

# Run a single test class.
$ mvn -Dtest=TestApp1 test

# Run multiple test classes.
$ mvn -Dtest=TestApp1,TestApp2 test

# Run a single test method from a test class.
$ mvn -Dtest=TestApp1#methodname test

# Run all test methods that match pattern 'testHello*' from a test class.
$ mvn -Dtest=TestApp1#testHello* test

# Run all test methods match pattern 'testHello*' and 'testMagic*' from a test class.
$ mvn -Dtest=TestApp1#testHello*+testMagic* test
```

With IntelliJ, you can see the test results from the editor.



```
Tests passed: 2 of 2 tests – 33 ms

▼ ✓ HelloworldTest   33 ms
    ✓ HelloWorld     29 ms
    ✓ HelloWorldErr   4 ms
```

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_171.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

*Don't be afraid, I just wanna help you.*