JUNIT 5

- Most important concept in unit testing is that we want to test a conceptual unit. Usually this unit is a method, but it could also be a group of methods or a group of classes.

```
<!-- https://mvnrepository.com/artifact/junit/junit -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version>
    <scope>test</scope>
</dependency>
```

Junit 5 commands (org.junit.Assert.*):

- fail(param1) → fails the test
- assertEquals(expected, actual) → actual value is the value that our method returns.
- assertTrue / assertFalse(someCondition) → test passes if someCondition evaluates to true / false.
- assertAll → Evaluates all the expressions at once. Example:

```
// In a grouped assertion all assertions are executed, and all
// failures will be reported together.
assertAll("person",
        () -> assertEquals("Jane", person.getFirstName()),
        () -> assertEquals("Doe", person.getLastName())
);
```

- assertAll → Evaluates all the expressions at once.
- assertArrayEquals(array1, array2) → Test passes if both arrays are equal (same elements, in the same order)

Junit 5 annotations (org.junit.jupiter.Assertions.*):

- **@Test** → Denotes that a method is a test method.
    - @Test(**expected**=NullPointerException.class) → Test fails if the exception is not thrown
- **@ParametrizedTest** → Makes it possible to run a test multiple times with different arguments. **@ValueSource** admitted types: *shorts, bytes, ints, longs, floats, doubles,*

chars, strings, classes. Method requires a parameter that will read the ValueSource array. Example:

```
@ParameterizedTest
@ValueSource(strings = { "cali", "bali", "dani" })
void endsWithI(String str) {
   assertTrue(str.endsWith("i"));
}
```

- **@RepeatedTest**(value) → Works as Test but it repeats the test <value> times.
- **@BeforeEach**, **@AfterEach** → Executes method before and after each @Test method.
- **@BeforeAll**, **@AfterAll** → Executes method before and after ALL @Test methods.
- **@Disabled** → used to disable tests at class or method level.
- **@Timeout**(time) → Used to test performance. Test fails if method doesn't finish within the specified time in milliseconds.

Good practices:

- Test classes should be kept **separate** from the implementation classes. The convention is to have a "test" package at the same level as the "main" package, and from then on we should r**eplicate the package structure in the test package**, same as in the main, as well as the classes names.
- For the test methods, the **naming convention** is: "test" + name of the method we are testing + "_" concrete path we are taking within the method. For example: "testDivisionMethod_divideBy5()"
- Warning! When changing a class in the main folder, **take care in also changing the imports** in the test folder. Sometimes the IDE does not change it automatically.
- **Test suites**: A way in which we can organize and group our test classes, so that we can execute only a group of them if needed. We must create a class, and signal the test classes we would like to group.

```
@RunWith(JUnitPlatform.class)
@SelectClasses({ClassATest.class,ClassBTest.class,
          ClassCTest.class})
public class JUnit5TestSuiteExample
{}
```

  - We can also use test suites with **packages**:

```
@RunWith(JUnitPlatform.class)
@SelectPackages("com.howto.test.examples")
@IncludePackages("com.howto.test.examples.tests")
public class JUnit5TestSuiteExample
{}
```

```
@RunWith(JUnitPlatform.class)
@SelectPackages("com.howto.test.examples")
@ExcludePackages("com.howto.test.examples.tests")
public class JUnit5TestSuiteExample
{
}
```