A screenshot of a cell phone

Description automatically generatedJUnit 5 (and assertj) and Mockito

**JUnit** and **TestNG** are undoubtedly the two most popular unit-testing frameworks in the Java ecosystem. But JUnit is the defacto standard when it comes to testing in java apps. Junit 5 is the newest iteration

A picture containing indoor, table, bottle, photo

Description automatically generated

Unit test can be classified as regression tests. We write the tests in order to constantly control if our code is working as intended. If at any point in time something changes in the code or the overall behaviour of the program, the test will fail and let us know that we broke a part.

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated**Why use a testing framrwork?**

Let’s say we have a basic calculator class. Our goal is here to verify that the **actual** **value** we get equals to the **expected** **value**. Those 2 terms are really often used in the testing context.

This example shows just one test case out of many. For instance, what about negative numbers? We need to create a new test and pass negative numbers. When you have many of these tests, the developer has to look to the outputs to see how the tests went. So there are certain steps that a developer needs to do if not using a framework.

A screenshot of a cell phone

Description automatically generatedEach highlighted step will be different for each new test case. But what about the others? The other steps are common for every test case.

This is where Junit and other testing frameworks provides us value. They provide the 3 un-highlighted steps. It will run the tests, verify the results and provide a handy report back to the developer.

A picture containing table, paper, bird

Description automatically generated

**What was the problem with JUnit 4?**

A screenshot of a cell phone

Description automatically generatedIt was created many years ago and it couldn’t keep up with the changing technologies and industry. Another big problem was that it has a monolithic architecture, it has one jar file. When you need junit, you get the jar and don’t have the option to just pick a part of it that you actually need. Junit is an open-source project, so people work on it on their free time. Many bugs and features piled up.

The developers actually started a crowd funding campaign. They said “we have work to do and we need money to do this work”. This was the basis of Junit 5.

A close up of a sign

Description automatically generated**JUnit 5 Architecture**

As we said JUnit 4 was just one jar.

Plattform is the core what composes the library of JUnit, which is involved running the tests, providing a way for you to call the test runner, provides an execution context etc. This core platform is not something that developers directy interact with!

What we interact with is the **JUnit API**, which is called **Jupiter**. The API is called Jupiter since they didn’t want to name it Junit, because Junit is the name of the whole framework. (Jupiter is the 5th planet in the solar system).

JUnit 5 is not backward compatible out of the box. So they created another set of APIs to provide backwards compatibility for older tests. It is called **Vintage**. You can have a project where you have Junit 4 and Junit 5 tests both run in parallel.

You also have a library which let you **extend Junit**. Lets say you don’t like the Junit API. You can come up with your own APIs and still use the underline test engine (Platform).

* The test engine is what runs your tests and Jupiter is the API which informs the engine what to run, how to run it etc.
* You can also extend Jupiter itself. Not writing your own API.
* So you need 2 Maven dependencies to write Junit 5 tests, **Jupiter** and **Platform**.
* A lot of IDEs provide integrated support for Junit 5.

**Example Maven Project**

A screenshot of a cell phone

Description automatically generated

A picture containing bird

Description automatically generated

We are adding the 2 dependencies Junit and Jupiter.

* **<scope> test </scope>** This scope indicates that the dependency is not required for normal use of the application, and is only available for the test compilation and execution phases. This scope is not transitive.
* Whatever is in your test scope won’t be included in your final build. (when you do mvn install/deploy)

Asd

Video 7

--- Skip video 8

* If nothing failed (or exceptions thrown) in a method/test, Junit will count it as ‘passed’.   
  “No failures” means success