



Java Advanced

1. Maven en JUnit

**DE HOGESCHOOL
MET HET NETWERK**

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook



INHOUD

- 1. Herhaling: klasse, overerving en polymorfisme.
- 2. Wat is Maven?
- 3. Wat is JUnit?

1. Herhaling

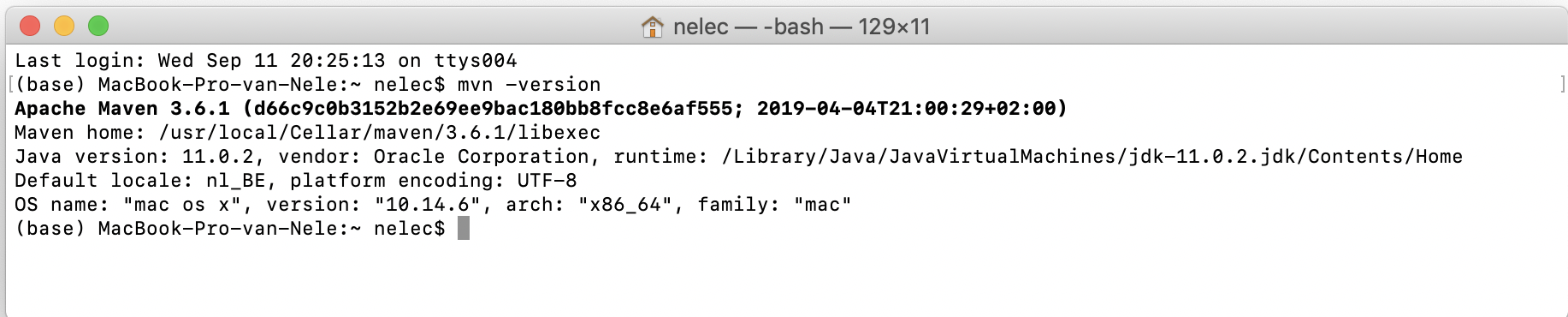
- Quiz
- Java cheat sheet (zie blackboard)

2. Wat is Maven?

- Een tool om het build proces voor java toepassingen te vereenvoudigen
- Geeft ontwikkelaars informatie over de kwaliteit van de java toepassing
- Richtlijnen voor best practises (code en testen gescheiden,...)
- Alternatieven: ant (with ivy), gradle

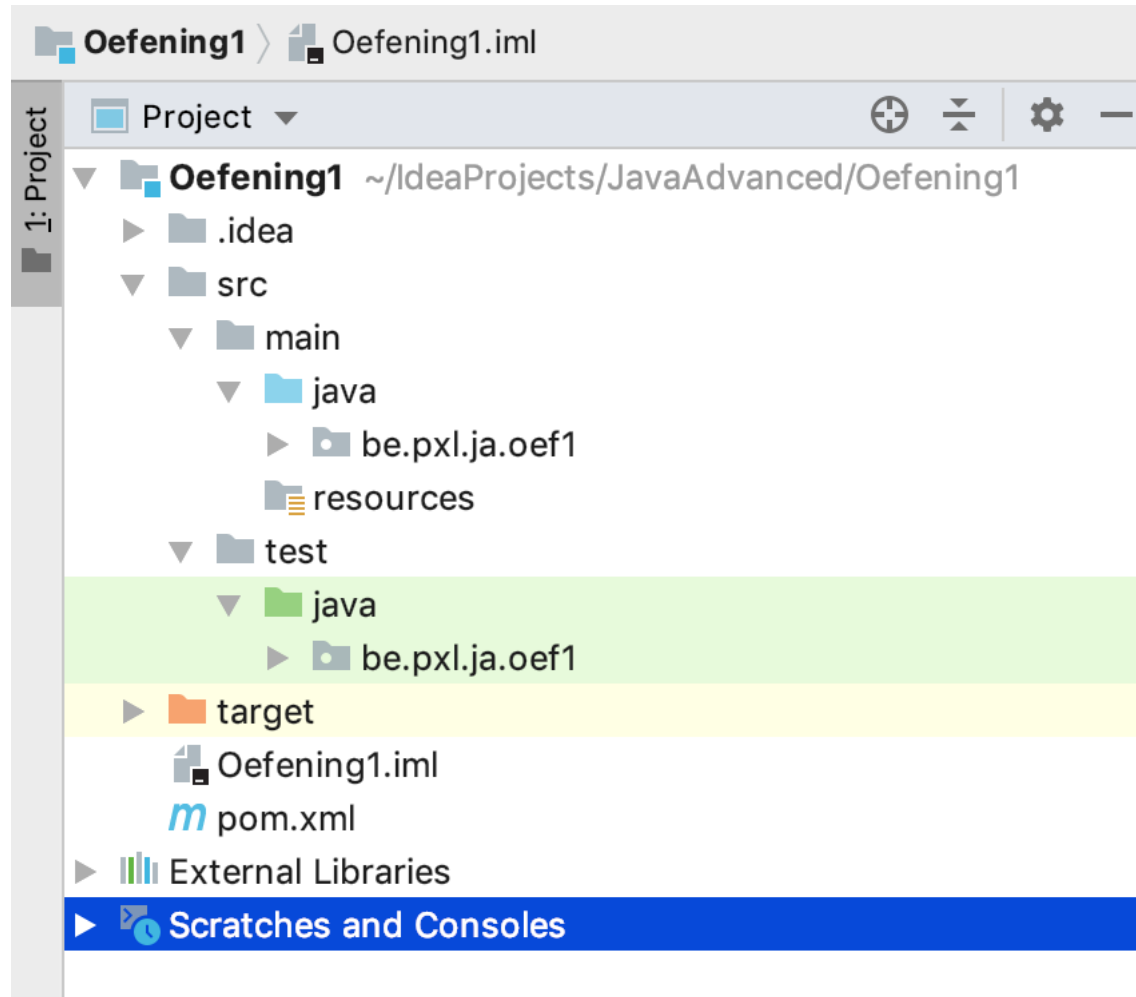
Installatie

- <https://maven.apache.org/download.cgi>
- <https://maven.apache.org/install.html>

A screenshot of a macOS terminal window. The title bar shows three colored window control buttons (red, yellow, green) on the left, a home icon and the text 'nelec — -bash — 129x11' in the center. The terminal content shows the output of the 'mvn -version' command. It starts with 'Last login: Wed Sep 11 20:25:13 on ttys004', followed by the prompt '[(base) MacBook-Pro-van-Nele:~ nelec\$ mvn -version]'. The output includes 'Apache Maven 3.6.1' with a long hash and timestamp, 'Maven home: /usr/local/Cellar/maven/3.6.1/libexec', 'Java version: 11.0.2' with vendor and runtime details, 'Default locale: nl_BE, platform encoding: UTF-8', and 'OS name: "mac os x", version: "10.14.6", arch: "x86_64", family: "mac"'. The prompt returns to '[(base) MacBook-Pro-van-Nele:~ nelec\$]' with a cursor.

```
Last login: Wed Sep 11 20:25:13 on ttys004
[(base) MacBook-Pro-van-Nele:~ nelec$ mvn -version
Apache Maven 3.6.1 (d66c9c0b3152b2e69ee9bac180bb8fcc8e6af555; 2019-04-04T21:00:29+02:00)
Maven home: /usr/local/Cellar/maven/3.6.1/libexec
Java version: 11.0.2, vendor: Oracle Corporation, runtime: /Library/Java/JavaVirtualMachines/jdk-11.0.2.jdk/Contents/Home
Default locale: nl_BE, platform encoding: UTF-8
OS name: "mac os x", version: "10.14.6", arch: "x86_64", family: "mac"
[(base) MacBook-Pro-van-Nele:~ nelec$ ]
```

Mappenstructuur



Project Object Model (POM)

m Oefening1 x

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
5   <modelVersion>4.0.0</modelVersion>
6
7   <groupId>be.pxl.ja</groupId>
8   <artifactId>Oefening1</artifactId>
9   <version>1.0-SNAPSHOT</version>
10
11   <properties>
12     <maven.compiler.target>11</maven.compiler.target>
13     <maven.compiler.source>11</maven.compiler.source>
14   </properties>
15
16   <dependencies>
17     <dependency>
18       <groupId>org.junit.jupiter</groupId>
19       <artifactId>junit-jupiter-engine</artifactId>
20       <version>5.5.2</version>
21       <scope>test</scope>
22     </dependency>
23   </dependencies>
24 </project>
```

Lifecycles en phases

- Clean lifecycle (project cleaning)
- Default lifecycle (project deployment)
 - `validate` - validate the project is correct and all necessary information is available
 - `compile` - compile the source code of the project
 - `test` - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
 - `package` - take the compiled code and package it in its distributable format, such as a JAR.
 - `verify` - run any checks on results of integration tests to ensure quality criteria are met
 - `install` - install the package into the local repository, for use as a dependency in other projects locally
 - `deploy` - done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.
- Site lifecycle (creation of project documentation)

Clean Lifecycle Bindings

<code>clean</code>	<code>clean:clean</code>
--------------------	--------------------------

Default Lifecycle Bindings - Packaging `ejb` / `ejb3` / `jar` / `par` / `rar` / `war`

<code>process-resources</code>	<code>resources:resources</code>
<code>compile</code>	<code>compiler:compile</code>
<code>process-test-resources</code>	<code>resources:testResources</code>
<code>test-compile</code>	<code>compiler:testCompile</code>
<code>test</code>	<code>surefire:test</code>
<code>package</code>	<code>ejb:ejb</code> or <code>ejb3:ejb3</code> or <code>jar:jar</code> or <code>par:par</code> or <code>rar:rar</code> or <code>war:war</code>
<code>install</code>	<code>install:install</code>
<code>deploy</code>	<code>deploy:deploy</code>

Bijv. `mvn clean package`

Uitvoerbare .jar

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <version>3.0.2</version>
      <configuration>
        <archive>
          <manifest>
            <addClasspath>true</addClasspath>
            <mainClass>
              be.px1.ja2.spel.App
            </mainClass>
          </manifest>
        </archive>
      </configuration>
    </plugin>
  </plugins>
</build>
```

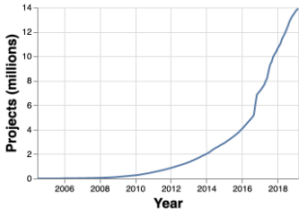
Dependencies bijv. JUnit

← → ↻ 🔒 mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-engine/5.5.2

MVNREPOSITORY

Search for groups, artifacts, categories

Indexed Artifacts (15.1M)



Popular Categories

Aspect Oriented

Actor Frameworks

Application Metrics

Build Tools

Bytecode Libraries

Command Line Parsers

Cache Implementations

Cloud Computing


Code Analyzers

Collections

Configuration Libraries

Core Utilities

Home » org.junit.jupiter » junit-jupiter-engine » 5.5.2



JUnit Jupiter Engine » 5.5.2

Module "junit-jupiter-engine" of JUnit 5.

License	EPL 2.0
Categories	Testing Frameworks
HomePage	https://junit.org/junit5/
Date	(Sep 08, 2019)
Files	jar (214 KB) View All
Repositories	Central
Used By	3,504 artifacts

Maven

Gradle

SBT

Ivy

Grape

Leiningen

Buildr

```
<!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-engine -->
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-engine</artifactId>
  <version>5.5.2</version>
  <scope>test</scope>
</dependency>
```

3. Wat is unittesten?

- Unittesten is een methode om softwaremodulen of stukjes broncode (**units**) afzonderlijk te **testen**. Bij unittesten zal voor iedere **unit** een of meerdere **tests** ontwikkeld worden.
- Een **unit** is voor ons een methode in een klasse.

Waarom schrijf ik unit testen?

- Minder fouten in je code
- Je durft je code aan te passen en beter leesbaar te maken.
- Je denkt meer na over je klasse en de implementatie van de methoden.
- Bron: <http://www.onjava.com/pub/a/onjava/2003/04/02/javaxpckbk.html>

Wat is JUnit?

- JUnit is een open source framework ontwikkeld om unit testen te schrijven en uit te voeren in Java.



Een voorbeeld

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

```
import static org.junit.jupiter.api.Assertions.assertEquals;
```

```
public class PersoonTest {
```

```
    @Test
```

```
    public void testGetVolledigeNaam( ) {
```

```
        Persoon p = new Persoon("Flater", "Guust");
```

```
        assertEquals("Guust Flater", p.getVolledigeNaam());
```

```
    }
```

```
    @Test
```

```
    public void testGetVolledigeNaamIndienNaamNull() {
```

```
        Persoon p = new Persoon(null, "Guust");
```

```
        assertEquals("Guust ?", p.getVolledigeNaam( ));
```

```
    }
```

```
    @Test
```

```
    public void testGetVolledigeNaamIndienVoornaamNull() {
```

```
        Persoon p = new Persoon("Flater", null);
```

```
        assertEquals("? Flater", p.getVolledigeNaam( ));
```

```
    }
```

```
}
```

Een voorbeeld

@Test

public void testGetVolledigeNaamIndienNaamNull() {

Persoon p = **new** Persoon(**null**, "**Guust**");

→ ARRANGE

String volledigeNaam = p.getVolledigeNaam();

→ ACT

assertEquals("Guust ?", volledigeNaam);

→ ASSERT

}

Testen uitvoeren

```
public class Persoon {  
    private String naam;  
    private String voornaam;  
  
    public Persoon(String naam, String voornaam) {  
        this.naam = naam;  
        this.voornaam = voornaam;  
    }  
  
    public String getVolledigeNaam() {  
        return null;  
    }  
}
```

Testen uitvoeren

Run: PersoonTest x

Tests failed: 3 of 3 tests – 31 ms

Test Results 31 ms

- ✗ PersoonTest 31 ms
 - ✗ testGetVolledigeNaamIndienNaam 24 ms
 - ✗ testGetVolledigeNaam() 2 ms
 - ✗ testGetVolledigeNaamIndienVoorn 5 ms

/Library/Java/JavaVirtualMachines/jdk-11.0.2.jdk/Contents/Home/bin/java ...

org.opentest4j.AssertionFailedError:
Expected :Guust ?
Actual :null
[<Click to see difference>](#)

<5 internal calls>

- at be.pxl.ja.demo.PersoonTest.testGetVolledigeNaamIndienNaamNull([PersoonTest.java:17](#)) <31 internal calls>
- at java.base/java.util.ArrayList.forEach([ArrayList.java:1540](#)) <9 internal calls>
- at java.base/java.util.ArrayList.forEach([ArrayList.java:1540](#)) <21 internal calls>

Testen uitvoeren

```
public class Persoon {  
    private String naam;  
    private String voornaam;  
  
    public Persoon(String naam, String voornaam) {  
        this.naam = naam;  
        this.voornaam = voornaam;  
    }  
  
    public String getVolledigeNaam() {  
        StringBuilder volledigeNaam = new StringBuilder(voornaam);  
        if (naam == null) {  
            volledigeNaam.append(" ?");  
        } else {  
            volledigeNaam.append(" ").append(naam);  
        }  
        return volledigeNaam.toString();  
    }  
}
```

Testen uitvoeren

✓

✗

↶

↷

≡

⚙️

Tests failed: 1, passed: 2 of 3 tests – 29 ms

▼ Test Results29 ms

▼ PersoonTest29 ms

- ✓ testGetVolledgeNaamIndienNaar21 ms
- ✓ testGetVolledgeNaam()1 ms
- ! testGetVolledgeNaamIndienVoorn7 ms

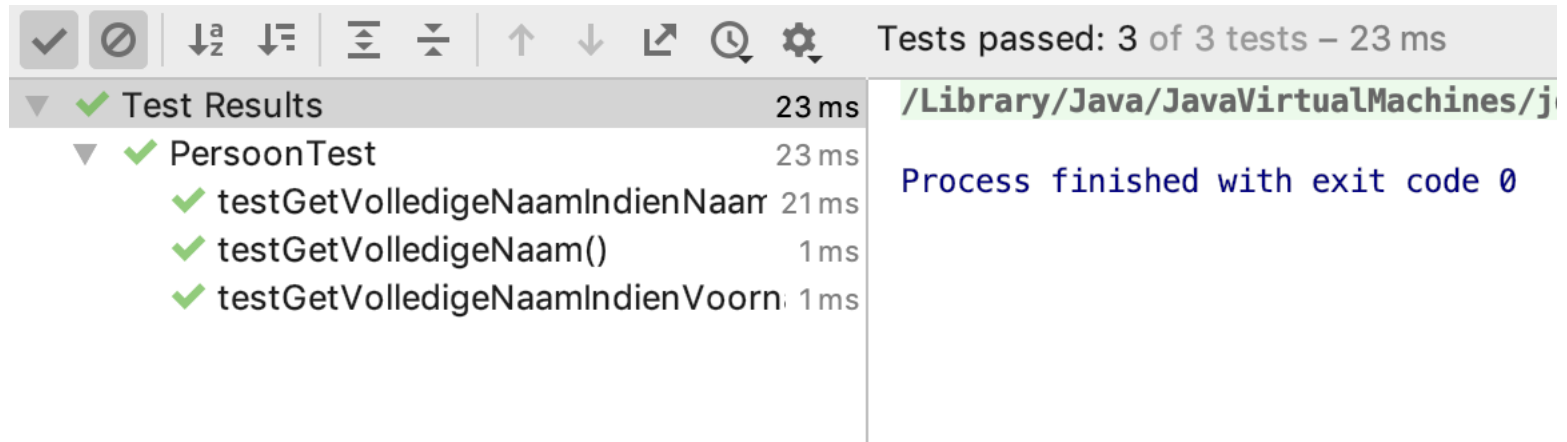
/Library/Java/JavaVirtualMachines/jdk-11.0.2.jdk/Contents/Home/bin/java ...

java.lang.NullPointerException
at java.base/java.lang.StringBuilder.<init>(StringBuilder.java:124)
at be.px1.ja.demo.Persoon.getVolledgeNaam(Persoon.java:13)
at be.px1.ja.demo.PersoonTest.testGetVolledgeNaamIndienVoornaamNull(PersoonTest.java:23) <31 internal calls>
at java.base/java.util.ArrayList.forEach(ArrayList.java:1540) <9 internal calls>
at java.base/java.util.ArrayList.forEach(ArrayList.java:1540) <21 internal calls>

Testen uitvoeren

```
public String getVolledigeNaam() {  
    StringBuilder volledigeNaam = new StringBuilder();  
    if (voornaam == null) {  
        volledigeNaam.append("?");  
    } else {  
        volledigeNaam.append(voornaam);  
    }  
    if (naam == null) {  
        volledigeNaam.append(" ?");  
    } else {  
        volledigeNaam.append(" ").append(naam);  
    }  
    return volledigeNaam.toString();  
}
```

Testen uitvoeren



The screenshot displays a test runner interface with a toolbar at the top containing icons for pass, fail, sort, filter, expand, collapse, and search. The status bar indicates "Tests passed: 3 of 3 tests – 23 ms". The test results are listed in a tree view:

- Test Results (23 ms)
 - PersoonTest (23 ms)
 - testGetVolledigeNaamIndienNaam (21 ms)
 - testGetVolledigeNaam() (1 ms)
 - testGetVolledigeNaamIndienVoorn (1 ms)

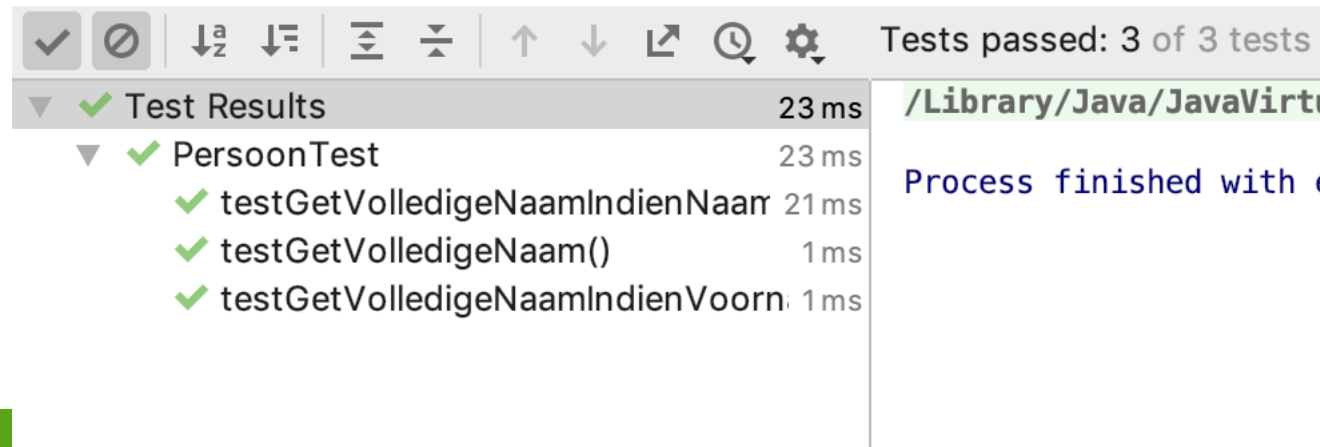
On the right side, the output pane shows the file path `/Library/Java/JavaVirtualMachines/j` and the message "Process finished with exit code 0".

Refactor

Kan je de leesbaarheid verbeteren?
Kan je dubbele code vermijden?

```
public String getVolledigeNaam() {  
    StringBuilder volledigeNaam = new StringBuilder();  
    volledigeNaam.append(vraagtekenIndienNull(voornaam));  
    volledigeNaam.append(" ");  
    volledigeNaam.append(vraagtekenIndienNull(naam));  
    return volledigeNaam.toString();  
}
```

```
private String vraagtekenIndienNull(String naam) {  
    if (naam == null) {  
        return "?";  
    }  
    return naam;  
}
```



The screenshot shows a test runner interface with a toolbar at the top containing icons for checkmark, close, expand, collapse, and other navigation functions. Below the toolbar, a table displays test results. The table has two main sections: 'Test Results' and 'PersonTest'. The 'Test Results' section shows a total of 23 ms. The 'PersonTest' section shows three individual tests, all of which passed successfully, with a total time of 23 ms. The tests are: 'testGetVolledigeNaamIndienNaam' (21 ms), 'testGetVolledigeNaam()' (1 ms), and 'testGetVolledigeNaamIndienVoorn' (1 ms). To the right of the table, a status bar indicates 'Tests passed: 3 of 3 tests' and shows the path '/Library/Java/JavaVirt'.

Test Results	23 ms
PersonTest	23 ms
testGetVolledigeNaamIndienNaam	21 ms
testGetVolledigeNaam()	1 ms
testGetVolledigeNaamIndienVoorn	1 ms

Tests passed: 3 of 3 tests

/Library/Java/JavaVirt

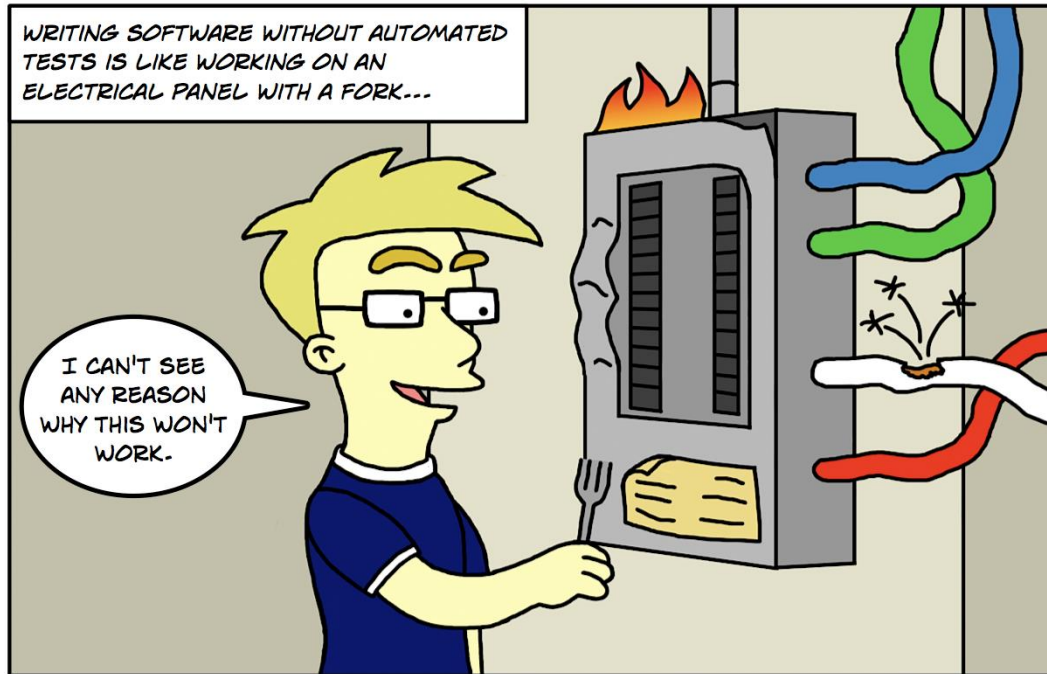
Process finished with

Methoden in org.junit.jupiter.api.Assertions

Methode	Betekenis
assertEquals()	Evalueert de gelijkheid van 2 waarden. De test slaagt als beide waarden gelijk (equal) zijn.
assertFalse()	Evaluatie van een booleaanse uitdrukking. De test slaagt indien de uitdrukking false is.
assertTrue()	Evaluatie van een booleaanse uitdrukking. De test slaagt indien de uitdrukking true is.
assertNotNull()	Vergelijkt een object referentie met null. De test slaagt indien de object referentie niet null is.
assertNull()	Vergelijkt een object referentie met null. De test slaagt indien de object referentie null is.
assertSame()	Vergelijkt het geheugenadres van twee object referenties (gebruik maken van == operator). De test slaagt indien beide object referenties naar hetzelfde object verwijzen.
fail()	Zorgt ervoor de huidige test zal falen. Wordt regelmatig gebruikt bij het testen van exception handling.

WRITING SOFTWARE WITHOUT AUTOMATED TESTS IS LIKE WORKING ON AN ELECTRICAL PANEL WITH A FORK...

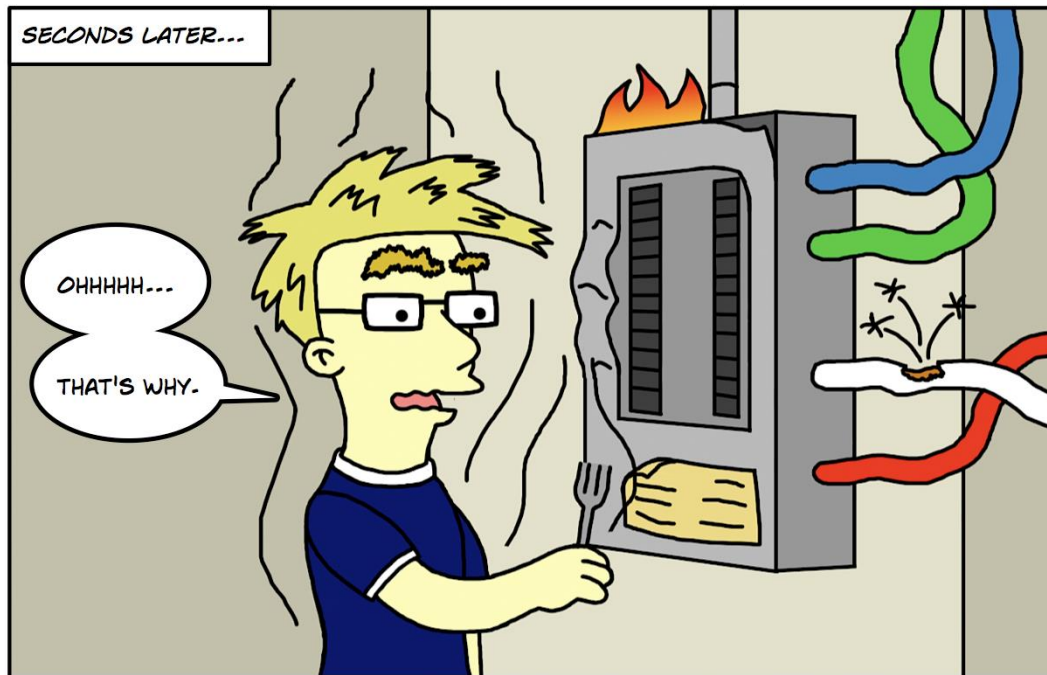
I CAN'T SEE ANY REASON WHY THIS WON'T WORK.



SECONDS LATER...

OHHHHH...

THAT'S WHY.



Handboek / pluralsight

- Handboek “Hoofdstuk 11: Junit”
(behalve 11.10)
- Pluralsight “Getting Started Unit Testing with JUnit 5” by Jim Weaver
 - Writing you first test
 - Writing more complex unit tests

