

**SEW3**

IT-Medientechnik

# Unittests mit JUnit

<https://junit.org/junit5/docs/current/user-guide/>

**TWS**



# Abhängigkeiten in pom.xml eintragen

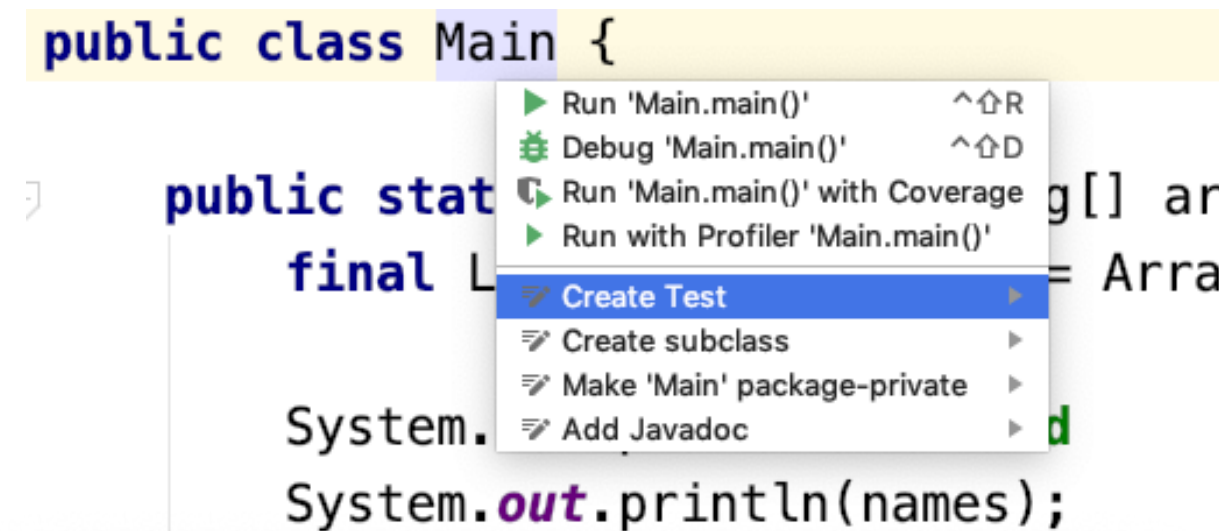
```
<properties>  
  <maven.compiler.source>11</maven.compiler.source>  
  <maven.compiler.target>11</maven.compiler.target>  
  <junit.jupiter.version>5.5.1</junit.jupiter.version>  
</properties>  
  
<dependencies>  
  <dependency>  
    <groupId>org.junit.jupiter</groupId>  
    <artifactId>junit-jupiter</artifactId>  
    <version>${junit.jupiter.version}</version>  
    <scope>test</scope>  
  </dependency>  
  <dependency>  
    <groupId>org.hamcrest</groupId>  
    <artifactId>hamcrest-all</artifactId>  
    <version>1.3</version>  
    <scope>test</scope>  
  </dependency>  
</dependencies>
```

# Tests generieren

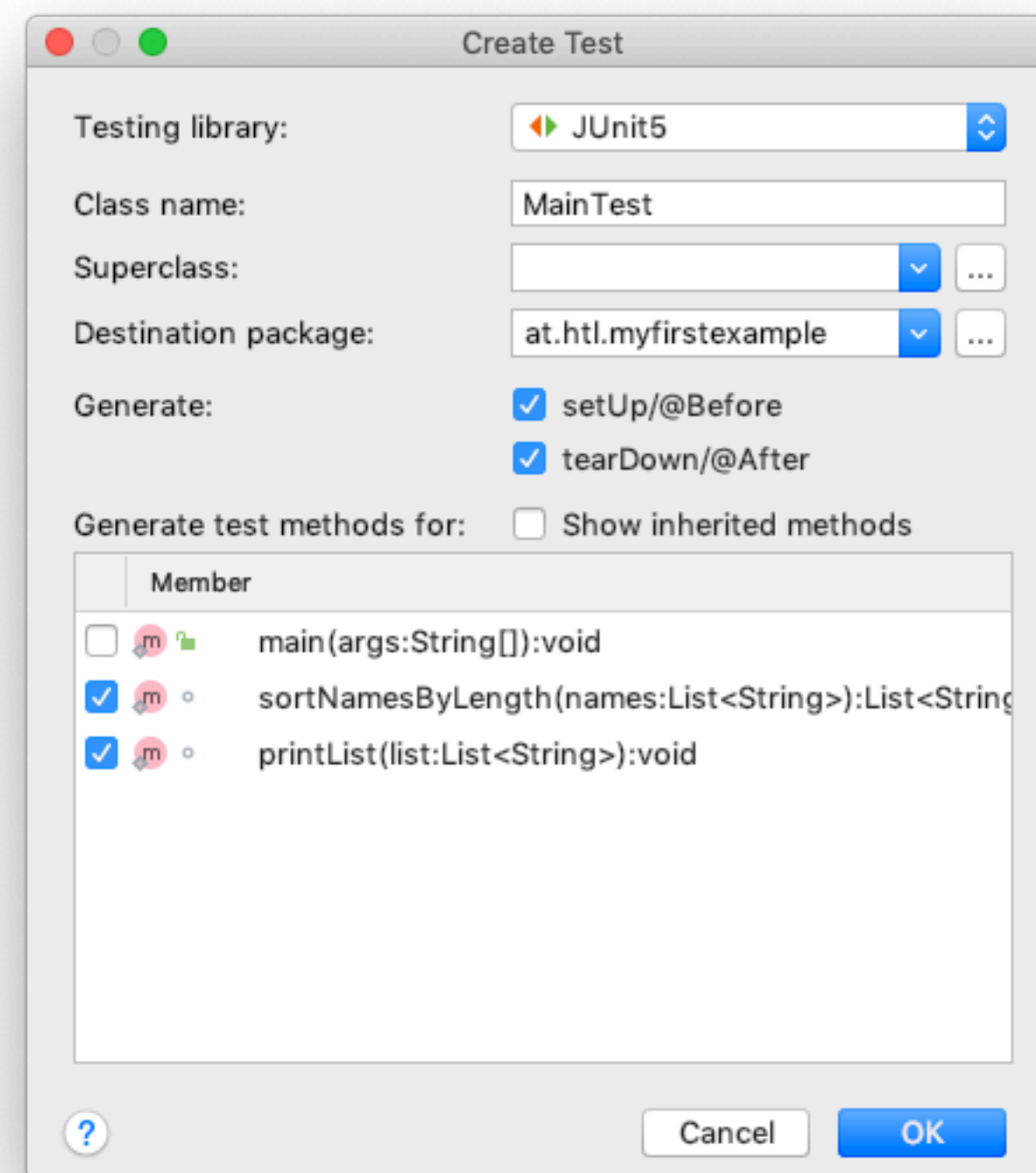
- 1. Cursor (Caret) auf den Klassennamen setzen

```
public class Main {
```

- 2. <Alt> <Enter> drücken



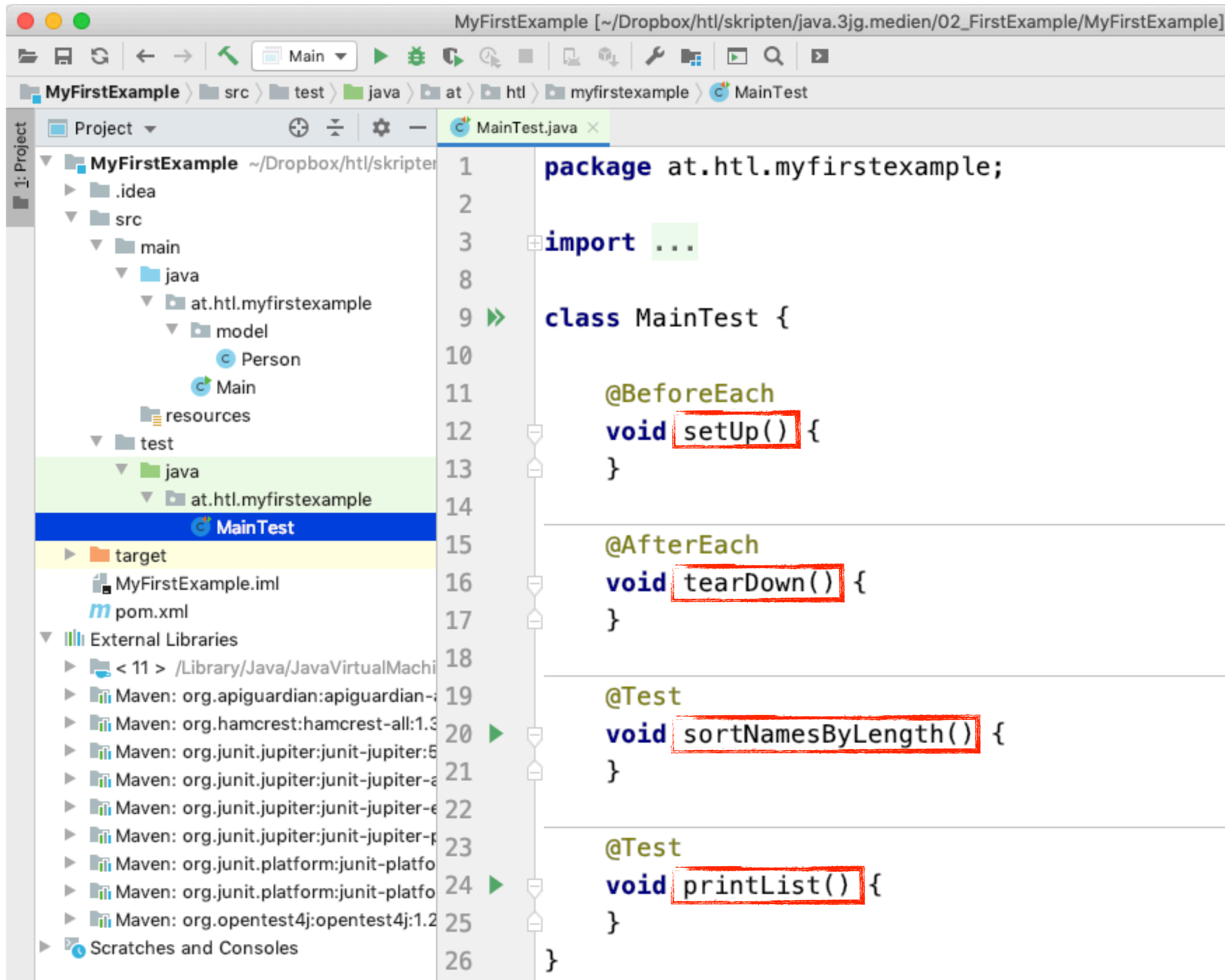
- 3. Create Test



- 4. Ok

Verwendet man Maven ist die Directory-Struktur definiert (Maven Standard Directory Layout). Man muss kein Root-Testverzeichnis oder sonstiges definieren

# Testgerüst wird generiert



The screenshot shows an IDE window titled "MyFirstExample" with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with a "test" directory containing a "MainTest" class. The code editor displays the following Java code:

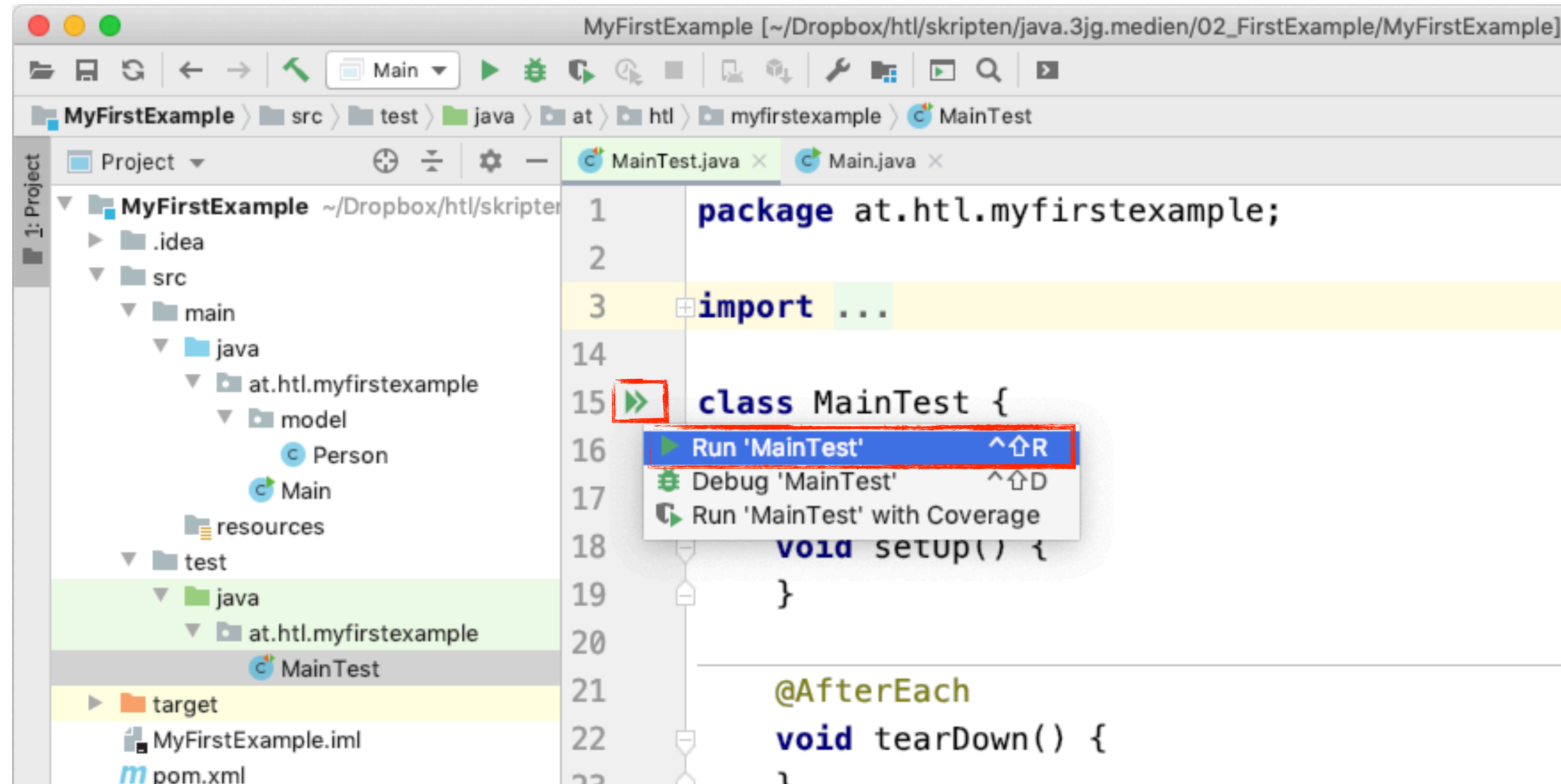
```
1 package at.htl.myfirstexample;
2
3 import ...
8
9 class MainTest {
10
11     @BeforeEach
12     void setUp() {
13     }
14
15     @AfterEach
16     void tearDown() {
17     }
18
19     @Test
20     void sortNamesByLength() {
21     }
22
23     @Test
24     void printList() {
25     }
26 }
```

## JUnit-Annotations

@BeforeEach	Denotes that the annotated method should be executed <i>before each</i> @Test, @RepeatedTest, @ParameterizedTest, or @TestFactory method in the current class; analogous to JUnit 4's @Before. Such methods are <i>inherited</i> unless they are <i>overridden</i> .
@AfterEach	Denotes that the annotated method should be executed <i>after each</i> @Test, @RepeatedTest, @ParameterizedTest, or @TestFactory method in the current class; analogous to JUnit 4's @After. Such methods are <i>inherited</i> unless they are <i>overridden</i> .
@BeforeAll	Denotes that the annotated method should be executed <i>before all</i> @Test, @RepeatedTest, @ParameterizedTest, and @TestFactory methods in the current class; analogous to JUnit 4's @BeforeClass. Such methods are <i>inherited</i> (unless they are <i>hidden</i> or <i>overridden</i> ) and must be <i>static</i> (unless the "per-class" <a href="#">test instance lifecycle</a> is used).
@AfterAll	Denotes that the annotated method should be executed <i>after all</i> @Test, @RepeatedTest, @ParameterizedTest, and @TestFactory methods in the current class; analogous to JUnit 4's @AfterClass. Such methods are <i>inherited</i> (unless they are <i>hidden</i> or <i>overridden</i> ) and must be <i>static</i> (unless the "per-class" <a href="#">test instance lifecycle</a> is used).

<https://junit.org/junit5/docs/current/user-guide/#writing-tests-annotations>

# Starten des Testlaufs



# Testergebnis

The screenshot displays an IDE interface with the following components:

- Project Structure:** MyFirstExample.iml, pom.xml, External Libraries (including Maven dependencies for hamcrest, junit-jupiter, and opentest4j), and Scratches and Consoles.
- Code Editor:** Shows the implementation of two test methods:

```
@Test
void sortNamesByLength() {
    final List<String> names = Arrays.asList("Berta", "Maximillian", "Tim", "Susi");
    Main.sortNamesByLength(names);
    assertThat(names, contains("Tim", "Susi", "Berta", "Maximillian"));
}

@Test
void printList() {
    fail("Not yet implemented");
}
```
- Run Console:** Shows the execution of the tests. The status is "Tests failed: 1, passed: 1 of 2 tests - 47 ms".
- Test Results:** A tree view showing the results for the MainTest class:
  - MainTest (47 ms)
    - printList() (33 ms) - Failed
    - sortNamesByLength() (14 ms) - Passed
- Stack Trace:** The failure for printList() is detailed as:

```
org.opentest4j.AssertionFailedError: Not yet implemented
<2 internal calls>
    at at.htl.myfirstexample.MainTest.printList(MainTest.java:34) <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>
```

# Tests fehlschlagen lassen

The screenshot shows an IDE window with a code editor and a run console. The code editor displays a Java test method:

```
@Test
void sortNamesByLength() {
    final List<String> names = Arrays.asList("Berta", "Maximillian", "Tim", "Susi");
    Main.sortNamesByLength(names);
    assertThat(names, contains("Susi", "Berta", "Maximillian", "Tim"));
}
```

The run console shows the following error message:

```
Tests failed: 2 of 2 tests - 46 ms
Test Results
  MainTest
    printList() 35 ms
    sortNamesByLength() 11 ms
java.lang.AssertionError:
Expected: iterable containing ["Susi", "Berta", "Maximillian", "Tim"]
but: item 0: was "Tim"
at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:20)
at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:8)
at at.htl.myfirstexample.MainTest.sortNamesByLength(MainTest.java:30) <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>
```

The IDE interface includes a sidebar with 'Structure' and 'Favorites' views, a 'Run' toolbar, and a status bar at the bottom showing 'Tests failed: 2, passed: 0 (moments ago)' and system information like '30:73 LF UTF-8 4 spaces' and '266 of 725M'.

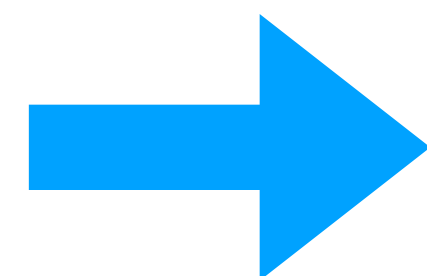
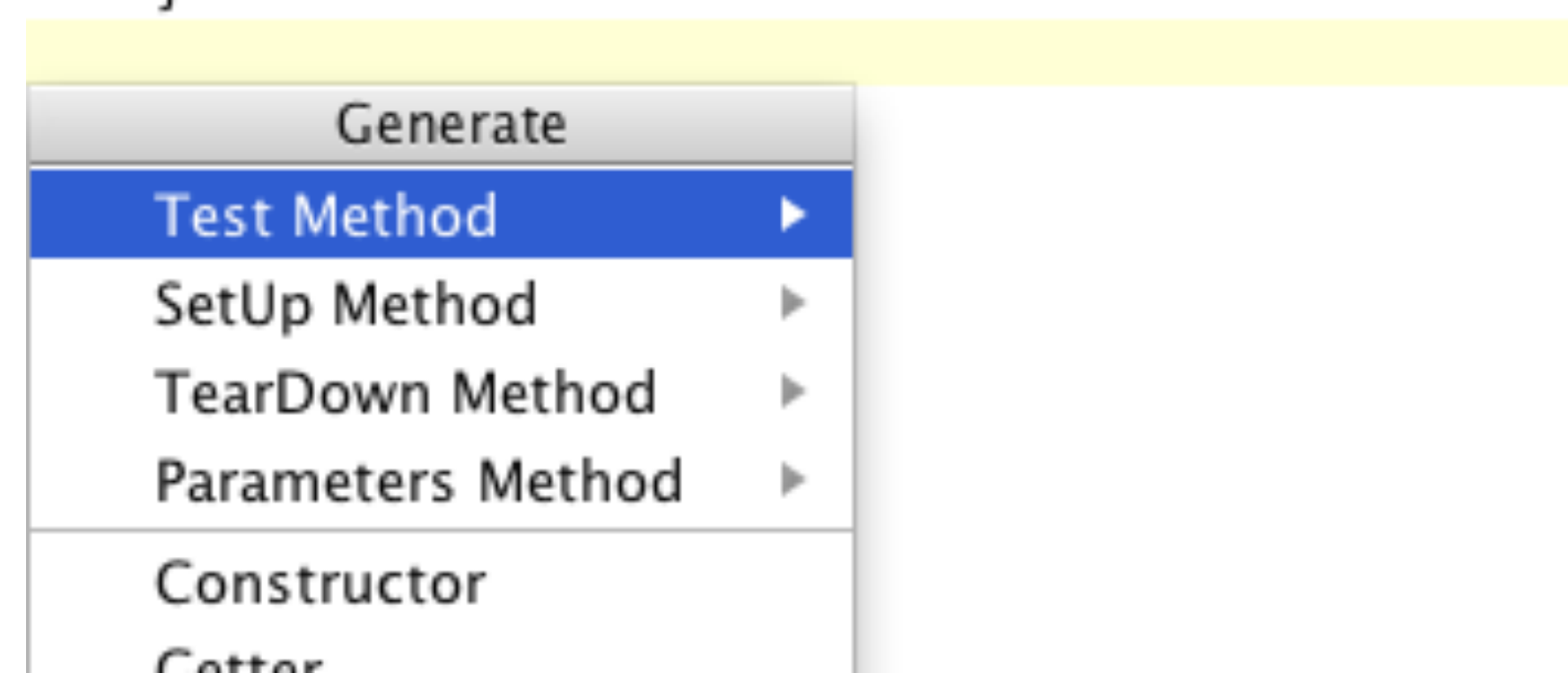
Während des Entwickelns soll man die einzelnen Tests auch einmal fehlschlagen lassen, um Ihre Funktionsfähigkeit zu überprüfen

# Anlegen einer neuen Testmethode

Cursor in leere Zeile innerhalb der Klasse stellen:

⌘N + ↵

```
@Test  
public void printList() throws Exception {  
}
```



```
@Test  
void name() {  
}
```



# Reihenfolge der Testdurchführung

```
import org.junit.jupiter.api.MethodOrderer.Alphanumeric;  
  
@TestMethodOrder(Alphanumeric.class)  
class MainTest {
```

JUnit Jupiter provides the following built-in `MethodOrderer` implementations.

- `MethodOrderer.Alphanumeric`
- `MethodOrderer.OrderAnnotation`
- `MethodOrderer.Random`

<https://blog.codeleak.pl/2019/03/test-execution-order-in-junit-5.html>

# Bibliotheken für Tests

- SystemRules (leider derzeit nur für jUnit4)  
<https://stefanbirkner.github.io/system-rules/>
- Java-Faker  
<https://github.com/DiUS/java-faker>
- Log Collectors  
<https://github.com/haasted/TestLogCollectors>
- Awaitility (zum Testen asynchroner Methodenaufrufe - wartet bis eine Bedingung erreicht ist)  
<https://github.com/awaitility/awaitility>
- EqualsVerifier  
<https://jqno.nl/equalsverifier/>
- Make-it-easy  
<https://github.com/npryce/make-it-easy>



Noch  
Fragen?

HTL LE  NDING

**Schön, hier zu lernen**