Unittests mit Java

Florian Pix

25. Januar 2019

Java-Kurs

Gliederung

1. Unittests
2. Oft genutzte Funktionen
3. Annotations
4. Testrunner
5. Demo und Aufgabe

Unittests

Unittests

Unittests sind Tests, die Einzelteile eures Programms überprüfen sollen. Sie können Teil ganzer *Integrationstests* sein die das Zusammenspiel der einzelnen Einheiten (Units) untersuchen.

Unittests

JUnit ist ein Framework für Java, dass wir zum Testen nutzen können.

Unittests

JUnit ist ein Framework für Java, dass wir zum Testen nutzen können.

Stand 20.Januar 2019, ist JUnit5 die aktuellste Version,
die auch im SWT Praktikum (3.Semester) genutzt wurde.

`https://junit.org/junit5/`

Unittests

JUnit ist ein Framework für Java, dass wir zum Testen nutzen können.

Stand 20.Januar 2019, ist JUnit5 die aktuellste Version,
die auch im SWT Praktikum (3.Semester) genutzt wurde.

`https://junit.org/junit5/`

Um es in euer Projekt einzubinden könnt ihr nach dem Download:

Eclipse: Project Properties$\rightarrow$ *JavaBuildPath*
Libraries $\rightarrow$ *AddExternalJARs...*

IntelliJ: File$\rightarrow$ *ProjectStructure* $\rightarrow$ *Modules*
Dependencies $\rightarrow +\rightarrow$ *JARsordirectories*

Oft genutzte Funktionen

Oft genutzte Funktionen

assertEquals()
assertTrue()
assertThrows()
fail()

Javadoc Assertions

Oft genutzte Funktionen

assertEquals()

```
1 public class AClass{
2     AClass(){}
3 }
4 void testCalculate(){
5 int expected = 5;
6 int actual = AClass.add(2, 3);
7 assertEquals(expected, actual);
8 }
```

Unser erwarteter Wert entspricht dem tatsächlichen,
also hat diese Einheit diesen Test bestanden.

Oft genutzte Funktionen

assertTrue()

```
1  public class AClass{
2      boolean noProblems = True;
3      AClass(){}
4      ...
5      public importantFunction(...);
6  }
7  ...
8  AClass aClass = new AClass();
9  ...
10 void testCalculate(){
11     AClass.importantFunction(...);
12     assertTrue(AClass.getNoProblems());
13 }
```

Wenn importantFunction() unsere Flag nicht ändert, besteht die Unit den Test.

Oft genutzte Funktionen

assertThrows()

Wir wollen testen ob eine Funktion unter bestimmten Bedingungen eine Exception wirft.

```
1 Executable codeToTest = () -> new Person("","");
2 assertThrows(IllegalArgumentException.class, codeToTest,
3         "The constructor of the person class did not throw an
      IllegalArgumentException,"
4         + "when given an empty string as forename parameter.");
```

Wenn codeToTest eine Exception wirft, besteht die Einheit den Test.

Oft genutzte Funktionen

fail()

```
1 void testSomething(){
2     int result = AClass.aFunction();
3     if(result < 4.0){
4         fail("Result should be above 4.0");
5     }
6 }
```

```
1 void testSomething(){
2     try{
3         AClass.bFunction();
4     } catch (SomethingWentTerriblyWrongException() e) {
5         fail("Something went terribly wrong");
6     }
7 }
```

Wenn man beim Durchlaufen des Tests auf ein fail() trifft, schlägt er fehl.

Annotations

Annotations

@Test

@BeforeAll, @AfterAll

@BeforeEach, @AfterEach

Annotations

@Test

```
1 @Test
2 public void test1() {
3     list.add("test");
4     assertFalse(list.isEmpty());
5     assertEquals(1, list.size());
6 }
```

Annotations

@BeforeAll, @AfterAll

```
1  @BeforeAll
2  public static void setUpAll() {
3      System.out.println
4      ("@BeforeAll , wird vor allen TestCases einmal ausgefuehrt.");

5  }
6
7  @AfterAll
8  public static void cleanUpAll() {
9      System.out.println
10     ("@AfterAll ,wird nach allen TestCases einmal ausgefuehrt.");

11 }
```

Annotations

@BeforeEach, @AfterEach

```
1
2  @BeforeEach
3  public void setUpEach() {
4      list = new ArrayList<String>();
5      System.out.println
6          ("@BeforeEach ,wird vor jedem einzelnen TestCase ausgefuehrt
       .");
7  }
8
9  @AfterEach
10 public void cleanUpAfterEach() {
11     list.clear();
12     System.out.println
13         ("@AfterEach, wird nach jedem einzelnen TestCase ausgefuehrt
       .");
14 }
```

Annotations

@ParameterizedTest,@RepeatedTest

Tests können wiederholt ausgeführt werden, sogar mit unterschiedlichen Parametern.

@DisplayName

Testklassen und -methoden können eigene Namen mit Leerzeichen, Sonderzeichen und sogar Emojis haben.

Weitere tolle Annotations

Testrunner

Testrunner

Entweder nutzt ihr den in eurer IDE eingebauten Testrunner,
oder schreibt euch einen eigenen.

```
1  import org.junit.runner.JUnitCore;
2  import org.junit.runner.Result;
3  import org.junit.runner.notification.Failure;
4  public class TestRunner {
5      public static void main(String[] args) {
6          Result result = JUnitCore.runClasses
7                                  (JunitAnnotationsExample.class);
8          for (Failure failure : result.getFailures()) {
9               System.out.println(failure.toString());
10         }
11         System.out.println("Result = " + result.wasSuccessful());
12     }
13 }
```

Demo und Aufgabe