Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Programmieren

Prof. Dr. Larissa Putzar &
Thorben Ortmann

JUnit

Gliederung

- Motivation
- Test-Driven Development
- Teststufen
- Unit-Tests mit JUnit
  - AAA-Pattern
  - Assertions
- JUnit-Annotationen
- JUnit - Testen mit Exceptions
- JUnit-Test in Eclipse anlegen

Motivation

- Qualität – Sind die Anforderungen erfüllt?
  - Funktionalität
  - Sicherheit
  - Design
- Fehlervermeidung
- Schnelles Feedback bei der Entwicklung
- Software lässt sich anhand von Tests schneller verstehen

Test-Driven Development (TDD)

Red (Testfälle sind „rot“):

Schreibe einen oder mehrere Tests, die eine Funktionalität deines Programms testen sollen. Analysiere dabei anhand der Tests auch das Problem und mögliche Grenzfälle. Die Ausführung der Testfälle soll möglich sein, aber zunächst fehlschlagen.

Green (Testfälle sind „grün“):

Implementiere die Funktionalität deines Programms, sodass alle Testfälle nun erfolgreich sind. („Make it work“)

Refactor:

Räume deinen Code auf und optimiere ggf. die Struktur deines Codes („Make it right“) sowie die Performanz („Make it fast“).

Teststufen

Software-Tests können auf vielen verschiedenen Stufen/Ebenen stattfinden:

- Unit-Test (auch Modultest)
- Integrationstest
- System- und End-To-End-Tests
- Abnahmetest (User Acceptance Test (UAT))

Wie werden uns im Folgenden nur mit Unit-Tests befassen.

Unit-Tests mit JUnit

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import example.util.Calculator;
import org.junit.jupiter.api.Test;

class MyFirstJUnitJupiterTests {

	private Calculator calculator = new Calculator();

	@Test
	void addition() {
		assertEquals(2, calculator.add(1, 1));
	}
 }
```

AAA-Pattern

Arrange:

Schaffe die Voraussetzungen für die Ausführung der zu testenden Funktionalität. Initialisiere die benötigten Objekte uns setze die benötigten Werte.

Act:

Führe die zu testende Funktionalität mit den unter *Arrange* gesetzten Parametern aus.

Assert:

Prüfe, ob das in *Act* erzeugte Ergebnis (Ist-Wert) dem erwarteten Ergebnis (Soll-Wert) entspricht.

AAA-Pattern Beispiel

```
class MyFirstJUnitJupiterTests {

        @Test
        void addition() {
                // Arrange
                Calculator calculator = new Calculator();
                int firstSummand = 1;
                int secondSummand = 1;
                int expectedResult = 2;

                // Act
                int actualResult = calculator.add(firstSummand, secondSummand);

                // Assert
                assertEquals(expectedResult, actualResult);
        }
}
```

Assertions

- assertEquals(Soll-Wert, Ist-Wert)
- assertNotEquals(Soll-Wert, Ist-Wert)
- assertTrue(Ist-Wert)
- assertFalse(Ist-Wert)
- assertNull(Ist-Wert)
- assertNotNull(Ist-Wert)
- assertInstanceOf(Soll-Typ, Ist-Wert)
- assertThrows(Soll-Exception, Methodenaufruf(der die Exception werfen soll))
- …

Annotationen

- @Test
- @BeforeEach
- @AfterEach
- @BeforeAll
- @AfterAll
- @Disabled
- @ParameterizedTest

(für eine Beschreibung aller Annotationen siehe https://junit.org/junit5/docs/current/user-guide/#writing-tests-annotations)

Demo

- Anlegen eines Eclipse-Projektes
- mit JUnit-Testfällen für die Implementierung der Fakultätsfunktion
- im Stile des Test-Driven-Development
- Benutzung des AAA-Pattern
- Benutzung des Debuggers