Fragen, Antworten, Kommentare

Die Online-Befragung zur gewählten alternativen Veranstaltungsform ist online. Bitte ausfüllen: https://forms.gle/xLm21KRaTgs7En7G6. Danke an alle bisherigen teilnehmenden Personen, wer noch nicht mitgemacht hat, bitte mitmachen.

Die Erstellung von Tests ist eine zentrale Aufgabe während der Programmentwicklung. Tests können auch vor der Entwicklung geschrieben werden, dadurch besteht die Möglichkeit das zu programmierende Verhalten genau zu beschreiben. Wie immer bei Testfällen werden typische Fälle und jedwedes mögliche Randverhalten in Tests umgesetzt. Danach wird inkrementell die Software entwickelt und es sollten Schritt für Schritt mehr Tests grün werden. Da einige Tests laufen werden, auch wenn das zu entwickelnde Programm nichts macht, wird die Zahl der laufenden Tests nicht unbedingt kontinuierlich anwachsen. Das folgende Beispiel gibt einen kleinen unvollständigen Ausblick, was noch mit JUnit möglich ist. Beachten Sie, dass es unüblich ist Ausgaben in Tests zu programmieren, dies erfolgt hier zur Anschauung. Die zu testende Klasse hat eine Methode, die Zahlen addiert.

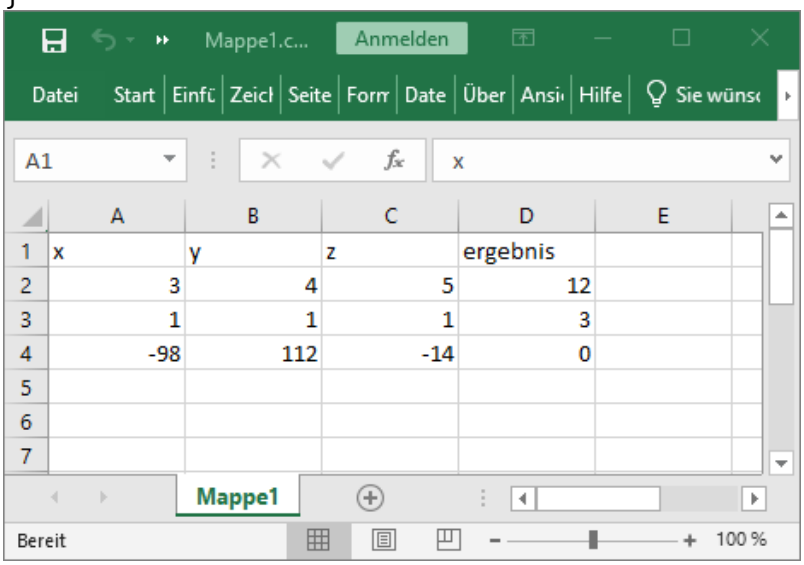
```
public class Adder {
  public int sum(int x, int y, int z) {
    return x+ y + z;
  }
}
```

Die Beispieltestklasse sieht wie folgt aus.

```
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Assumptions;

import java.util.List;
import java.util.stream.Stream;

import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.Arguments;
import org.junit.jupiter.params.provider.CsvFileSource;
import org.junit.jupiter.params.provider.MethodSource;

public class AdderTest {

  private Adder sut; // System under Test
  private Adder sut2; // kann beliebig viele Testobjekte geben

  @BeforeAll
  public static void setUpBeforeClass() {
    //Dies wird einmal am Start fuer alle Tests gemacht
  }
```

```java
@AfterAll
public static void tearDownAfterClass() {
  //Dies wird einmal am Ende fuer alle Tests gemacht
}

@BeforeEach
public void setUp() {
  //Dies wird einmal vor jedem Test gemacht
  this.sut = new Adder();
  this.sut2 = new Adder();
}

@AfterEach
public void tearDown() {
  //Dies wird einmal mach jedem Test gemacht
}

@Test
void testSumErwartet() {
  System.out.println("test1");
  int erg = this.sut.sum(21, 21, 0);
  Assertions.assertEquals(42, erg
      , "Erwartet wurde 42, gefunden: " + erg);
}

@Test
void testSumVertauschbar() {
  System.out.println("test2");
  Assertions.assertEquals(this.sut.sum(1, 2, -1)
      , this.sut2.sum(-1, 1, 2));
}

@Test
void testSumMitAnnahme() {
  System.out.println("test3");
  // Test nur ausfuehren, wenn Annahme erfuellt
  Assumptions.assumeTrue(1 + 2 + 3 == 6);
  int erg = this.sut.sum(1, 2, 3);
  Assertions.assertEquals(6, erg
      , "Erwartet wurde 42, gefunden: " + erg);
}

// Stellen Sie sich Stream einfach als List vor; der
// ebenfalls mit einem Iterator alle Elemente durchlaufen kann.
public static Stream<Arguments> daten(){
  List<Arguments> testdaten = List.of(
      Arguments.of(0, 0, 0, 0),
      Arguments.of(-2, -1, -3, -6),
      Arguments.of(6, 7, 29, 42)
  );
  return testdaten.stream();
}
```

```java
  @ParameterizedTest
  @MethodSource({"daten"})
  public void testSumParametrisiert(int a, int b, int c, int erg) {
    System.out.println("Testpar mit " + a + "," + b + "," + c);
    Assertions.assertEquals(erg, this.sut.sum(a, b, c));
  }

  @ParameterizedTest
  @CsvFileSource(resources ={"/Mappe1.csv"}, numLinesToSkip = 1)
  public void testSumMitExcelSheet(int a, int b, int c, int erg) {
    System.out.println("Testcsv mit " + a + "," + b + "," + c);
    Assertions.assertEquals(erg, this.sut.sum(a, b, c));
  }
}
```



Die resultierende Ausgabe sieht wie folgt aus.