Kurzeinführung JUnit

Ullrich Köthe

Unit-Tests

- **Unit-Tests sind „ausführbare Spezifikationen“**
  - **Prüfen der semantischen Konformität einer Klasse, eines Moduls etc. zur Spezifikation**
  - **Dokumentieren der Spezifikation durch Testcode**
- **Tests und Anwendungscode parallel entwickeln**
  - **test a little, code a little, test a little, ...**
  - **Bugs durch Tests reproduzieren, dann korrigieren**
  - **wenn Tests nicht 100% erfolgreich => erst korrigieren**
- **Unit-Tests werden in TestSuites gesammelt und *häufig* ausgeführt**
  - **sämtliche Tests mindestens täglich**
  - **aktuell betroffene Tests im Minutenabstand**
  - **Tests niemals löschen (außer wenn veraltet / redundant)**

JUnit

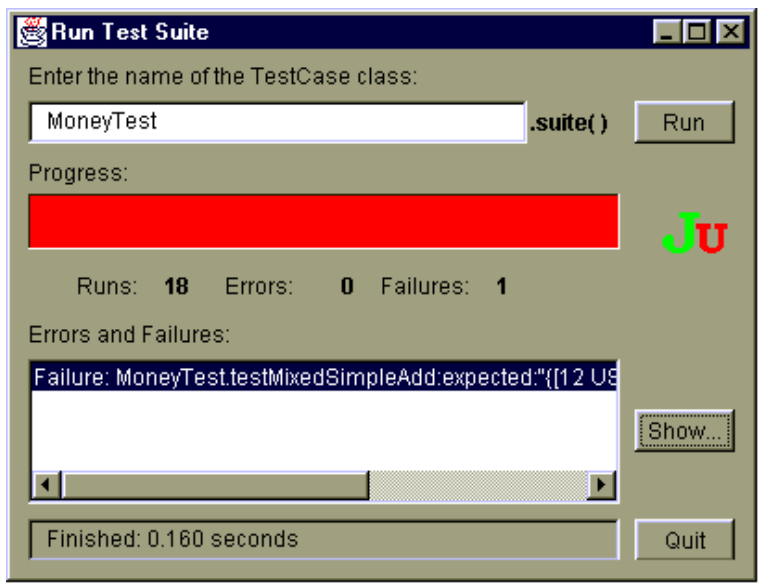
- **Java-Framework zur Vereinfachung von Unit-Tests (auch für andere Sprachen verfügbar)**
- **public domain, von Kent Beck und Erich Gamma**

  http://www.xprogramming.com/software.htm
- **Test-Klassen von Basisklassen aus JUnit ableiten**
- **Aufruf von der Console:**

  > java MyTest
- **GUI zum Aufruf der Tests;**

  > java junit.ui.LoadingTestRunner

  dann Namen der Testklasse
  oben eintragen,
  Run drücken,
  Fehler anschauen



Grundlegende Gestalt einer Testklasse

```
import junit.framework.*;

public class SimpleTest extends TestCase {

  public SimpleTest(String name) { super(name); }
  public static void main(String[] args) {
     junit.textui.TestRunner.run(suite()); }
  public static Test suite() {
      return new TestSuite(SimpleTest.class); }
                      //    ^^^^^^^^^^ name of test class

  protected void setUp() { ... }
  public void testThis() { ... }  // test functions
  public void testThat() { ... }  // named „test...()“
}
```

Prinzipien

- **Testobjekt wird für jeden Test neu initialisiert**
  ⇨ wohldefinierter Anfangszustand
- **wiederkehrende Initialisierungen („fixture“) in setUp()**
- **alle Testfunktionen beginnen mit „ void test...()“**
  ⇨ TestSuite findet Tests automatisch
- **Test können geschachtelt werden**

```
TestSuite combined = new TestSuite();
combined.addTest(SimpleTest.suite());
combined.addTest(new TestSuite(Test1.class));
combined.addTest(new Test2(„testSimpleAdd“));
TestResult result = combined.run();
```

Schreiben von Tests

```
public class RationalNumberTest extends TestCase {
  ...
  private RationalNumber oneHalf;
  private RationalNumber twoQuarters;

  protected void setUp() {
    oneHalf = new RationalNumber(1, 2);
    twoQuarters = new RationalNumber(2, 4); }

  public void testEqual() {
    assertEquals(oneHalf, oneHalf);
    assertEquals (oneHalf, twoQuarters);
    assert(!oneHalf.equals(new RationalNumber(1))); }

  public void testAdd() {
    RationalNumber expected = new RationalNumber(1);
    assertEquals(oneHalf.add(twoQuarters), expected); }
}
```

Assertions

- **werfen AssertionFailedException, wenn getestete Bedingung ‘false‘ ist**

```
assert(boolean condition)
assertEquals(double expected, double actual,
             double tolerance);
assertEquals(long expected, long actual);
assertEquals(Object expected, Object actual);
assertSame(Object expected, Object actual);
assertNull(Object o);
assertNotNull(Object o);
```

  - **sowie entsprechende Versionen mit spezifischer Meldung**
- **unbedingter Mißerfolg**

```
fail(String message)
```

Testen der Fehlerbehandlung

- **absichtliches Auslösen eines Fehlers**
- **Testen, ob erwartete Exception wirklich auftritt**

```
public class RationalNumberTest extends TestCase {
  ...
  public void testDivisionByZero() {
    zero = new RationalNumber(0);
    try {
        oneHalf.divide(zero);
        fail(“Failed to detect division by zero“);
    }
    catch(java.lang.ArithmeticException e) {}
  }
}
```