Kurzeinführung in JUnit

im Rahmen der Übungen zur Vorlesung
“Objektorientierte Programmierung mit Java“
im Sommersemester 2004

Johannes Mayer

jmayer@mathematik.uni-ulm.de

Abteilung Angewandte Informationsverarbeitung
Universität Ulm

Automatische Tests mit JUnit
JUnit: Was ist das?

- ***JUnit:*** *Framework* von *Java-Klassen und -Interfaces* zur einfachen Erstellung von *Klassen- und Interaktionstests* und Programmen zur *automatischen Durchführung* der Tests
- **Autoren:** Erich Gamma und Kent Beck
- **Homepage:** http://www.junit.org/
- *Freie Software* – Download unter obiger Homepage
- Einsatz heute schon *sehr verbreitet*
- *Im Folgenden verwendet* zur Implementierung von Tests

Automatische Tests mit JUnit

Ein erstes Beispiel

Beispiel: Gegeben ist eine „Klasse“ *Calc* zur *Addition von Integern* mit folgender *Schnittstelle*:

calc/Calc.java

```
public class Calc {
    public static int add(int a, int b);
}
```

Ziel: Entwicklung eines *Tests für die Methode add()* unter Verwendung von *JUnit*.

Automatische Tests mit JUnit

Ein erstes Beispiel (2)

Eigene von *TestCase abgeleitete Klasse* erzeugen mit einer *Methode test...()* für jeden *Testfall*. Ergebnisüberprüfung mit der Methode *assertTrue(...)* etc. durchführen.

calc/CalcTest.java

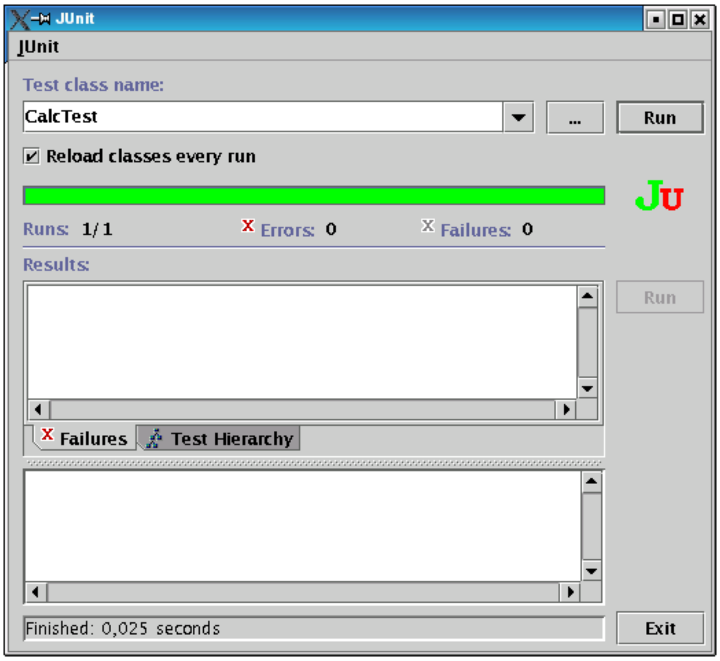
```
import junit.framework.*;

public class CalcTest extends TestCase {
    public void testAdd() {
        int result = Calc.add(1, 4);
        int expected = 5;
        assertEquals(expected, result);
    }
}
```

Bemerkung: Die Methode *assertEquals(..., ...)* ist besser als die Methode *assertTrue(...)* bei Vergleichen.
($\rightsquigarrow$ aussagekräftigere *Fehlermeldung*)

Automatische Tests mit JUnit
Ein erstes Beispiel (3)

Kommando: ***java -classpath junit.jar:. junit.swingui.TestRunner CalcTest***



Automatische Tests mit JUnit
JUnit-Framework: Die Klasse TestCase

- TestCase ist *Oberklasse aller Testklassen*
- *Testklasse* für die Klasse *MyClass* heißt per Konvention immer *MyClassTest*
- Jeder *Testfall* in einer Testklasse ist eine Methode *void test...()*, die keine Parameter erwartet, keinen Rückgabewert hat und deren Name mit „test“ beginnt
- *Unterklassen* von TestCase können einen *Konstruktor mit String-Parameter* haben, der mit diesem Parameter den entsprechenden Konstruktor der Oberklasse aufruft ($\leadsto$ Testen einzelner Fälle)
- *TestCase* erbt von *Assert* ($\leadsto$ auch für Java $\leq$ 1.4 und deutlich mächtiger als assert(...))

Automatische Tests mit JUnit
JUnit-Framework: Die Klasse Assert

- Definiert *Klassenmethoden assert...()*
- *assertTrue(...)* ist äquivalent zu *assert()* in Java 1.4
- *Vorteil von Assert:* Genauere Information über den Fehler durch spezielle assert()-Methoden

  Beispiele:
  - ⋄ *assertTrue(...)*, *assertFalse(...)*
  - ⋄ *assertEquals(..., ...)*
  - ⋄ *assertNull(...)*, *assertNotNull(...)*
  - ⋄ *assertSame(...)*, *assertNotSame(...)*
  - ⋄ *fail(...)*

Automatische Tests mit JUnit
JUnit-Framework: Die TestRunner-Klassen

- Drei verschiedene TestRunner-Klassen: *junit.swingui.TestRunner*, *junit.awtui.TestRunner* und *junit.textui.TestRunner*
- Was macht ein TestRunner mit der Testklasse? *Sammelt alle test...()-Methoden* der Klasse auf und führt diese Testfälle aus

Automatische Tests mit JUnit

Ein erstes Beispiel (4)

Eine zweite Testklasse für die Klasse Calc:

calc/CalcAnotherTest.java

```
import junit.framework.*;

public class CalcAnotherTest extends TestCase {
    public CalcAnotherTest(String s) {
        super(s);
    }
    public void testAddZero() {
        int result = Calc.add(1, 0);
        int expected = 1;
        assertEquals(expected, result);
    }
    public void testAddNegative() {
        int result = Calc.add(1, -1);
        int expected = 0;
        assertEquals(expected, result);
    }
}
```


Automatische Tests mit JUnit

Noch ein Beispiel: Brüche

Beispiel: Gegeben ist eine Klasse *Fraction* zur *Bruchrechnung*, deren Objekte nicht veränderbar sind (siehe folgende Schnittstelle).

fraction/Fraction.java

```
public class Fraction {
    public Fraction();
    public Fraction(int num);
    public Fraction(int num, int denom);
    public String toString();
    public boolean equals(Object o);
    public Fraction add(Fraction x);
    public double getValue();
}
```

Automatische Tests mit JUnit

Brüche: Ein paar Testfälle

fraction/FractionTest.java

```
import junit.framework.*;

public class FractionTest extends TestCase {
    public void testAdd() {
        Fraction a = new Fraction(3, 5);
        Fraction b = new Fraction(2, 7);
        Fraction expected = new Fraction(31, 35);
        assertEquals(expected, a.add(b));
    }
    public void testAddZero() {
        Fraction a = new Fraction(3, 5);
        Fraction zero = new Fraction();
        Fraction expected = new Fraction(3, 5);
        assertEquals(expected, a.add(zero));
    }
}
```


Automatische Tests mit JUnit

Das Inventar

- *Mehrere Testfälle* arbeiten mit *denselben Objekten*
- *Code nicht duplizieren*
- **Also:** Objekte in das *Inventar* auslagern
- Inventarobjekte werden mit *setUp() erzeugt* und mit *tearDown()* werden *permanente Ressourcen* (z.B. Datei- bzw. Netzwerkverbindungen) wieder *freigegeben*.
- Inventarobjekte werden *vor jedem einzelnen Testfall neu erzeugt*, d. h. die Methode *setUp*() wird vor und die Methode *tearDown*() wird nach *jedem Testlauf* (= Aufruf einer Methode test...()) aufgerufen.

Automatische Tests mit JUnit
Das Inventar (2)

Das vorige Beispiel *mit einem Inventar*:

fraction/FractionInventarTest.java

```java
import junit.framework.*;

public class FractionInventarTest extends TestCase {
    private Fraction a;
    protected void setUp() {
        a = new Fraction(3, 5);
    }
    public void testAdd() {
        Fraction b = new Fraction(2, 7);
        Fraction expected = new Fraction(31, 35);
        assertEquals(expected, a.add(b));
    }
    public void testAddZero() {
        Fraction zero = new Fraction();
        Fraction expected = new Fraction(3, 5);
        assertEquals(expected, a.add(zero));
    }
}
```


Automatische Tests mit JUnit
Testen von Exceptions

- Test auch von *Ausnahmesituationen*, sprich *Exceptions*
- *Erwartete Exceptions* abfangen (try-catch-Anweisung) und ggf. mit *fail(...)* bei *unerwarteter Situation* abbrechen

fraction/FractionExceptionTest.java

```java
import junit.framework.*;

public class FractionExceptionTest extends TestCase {
    public void testDenominatorZero() {
        try {
            Fraction a = new Fraction(1, 0);
            fail ("Should raise an IllegalArgumentException");
        }
        catch (IllegalArgumentException ex) {
        }
    }
}
```

Automatische Tests mit JUnit
Vergleich von Gleitkommazahlen

Wie lässt sich die *Methode getValue()* von *Fraction* testen? Mögliche *Schwierigkeiten*?

fraction/FractionDoubleTest.java

```
import junit.framework.*;

public class FractionDoubleTest extends TestCase {
    private static final double eps = 1e-12;
    public void testZero() {
        Fraction a = new Fraction();
        assertEquals(0.0, a.getValue(), eps);
    }
    public void testPositive() {
        Fraction a = new Fraction(17, 2);
        assertEquals(8.5, a.getValue(), eps);
    }
    public void testNegative() {
        Fraction a = new Fraction(7, -8);
        assertEquals(-7.0/8.0, a.getValue(), eps);
    }
}
```

Anmerkung: *Vergleich von Gleitkommazahlen* nicht via *x == y*, sondern via *Math.abs(x − y) < ...*!

Automatische Tests mit JUnit
Fazit

- *einfach* zu bedienen
- *frei bzw. kostenlos*
- *automatische* Tests einfach möglich
- *Regressions-Tests*, d. h. Testwiederholung nach jeder Erweiterung oder Änderung, dadurch ohne großen Aufwand möglich
- Tests in *Java*
- *Vergleichbare Test-Frameworks* auch für andere Sprachen vorhanden (z. B. für C++, C#, Visual Basic, Delphi, JavaScript, Php, Perl, HTML, XML, Flash, . . .) ⇝ http://www.xprogramming.com/software.htm

Automatische Tests mit JUnit
Literatur

- *Johannes Link*: *Unit Tests mit Java*. dpunkt-Verlag, Heidelberg, 2002.
  Homepage zum Buch
- *Vincent Massol, Ted Husted*: *JUnit in Action*. Manning, Greenwich, 2004.
- *Frank Westphal*: *Testgetriebene Entwicklung mit JUnit und FIT*. dpunkt-Verlag, Heidelberg, 2004.
  Homepage zum Buch