Seminar Simulation und Bildanalyse
mit Java

Thema: Testen von objektorientierter Software

Uta Dienst

1. Teil:
Einführung in den Software-Test

2. Teil:
JUnit-Einführung

1. Teil:
Einführung in den Software-Test

Inhaltsübersicht

- Motivation

  Softwarebedingtes Fehlverhalten, Software-Qualität

- Wie findet man Software-Fehler?
- Konventioneller Software-Test

  Definition, Blackbox- und Whitebox-Test, ...

- OO Software testen

  Unit Test, Dummy- u. Mock-Objekte

Software Fehler

- Trägerrakete Ariane 5
- Bestrahlungsgerät Therac 25
- Verkehrsflugzeug B373
- Defekter Geldautomat bei der Postbank
- Pentium Bug

Software-Qualitäts

Merkmale nach DIN ISO 9126:

- Funktionalität
- Zuverlässigkeit
- Benutzbarkeit
- Effizienz
- Änderbarkeit und Übertragbarkeit

Qualität als Unternehmensziel

- Erhöhung der Marktanteile und Kunden-Bindung
- Steigerung der Rentabilität und Produktivität
- Kostenreduktion
- Mitarbeiterzufriedenheit
- Vermeidung von Gewährleistungs- und Haftungs-Ansprüchen

Software-Fehler

- Wo?
  - ✓ Spezifikation
  - ✓ Design
  - ✓ Implementierung
- Wie entdecken?
  - ✓ Software-Test
  - ✓ Reviews/Inspektion
  - ✓ Dynamische Überprüfung (z.B. Debuggen)

Software-Test

- *Definition*: Testen von Software ist jede Ausführung eines Testobjekts, die zur Überprüfung dessen dient.

- Testen ist der Prozess, ein Programm mit der Absicht auszuführen, Fehler zu finden.

Konventionelle Test Methoden (1)

- Statische Analyse

  - Testobjekte kommen nicht zur Ausführung! -

  - ✓ Verifizierende Prüftechniken
    - Algebraische Techniken
    - Automatenbasierte Techniken

  - ✓ Analysierende Prüftechniken
    - Review- und Inspektions-Techniken
    - Metriken (Maße)
    - Grafiken und Tabellen

Konventionelle Test Methoden (2)

- Dynamische Programmanalyse

  - Klassifizierung von Testfall-Ermittlungsverfahren! -

  - ✓ White-Box vs. Black-Box-Test
  - ✓ Einordnung nach verwendeter Testreferenz

Konventionelle Test Methoden (3)

- Dynamische Programmanalyse
  - ✓ Strukturorientiere Test
    - Kontoll- oder Datenflussorientierte Tests
    - Bsp.: Anweisungs-, Zweigüberdeckungstest
  - ✓ Funktionsorientierte Test
    - Bsp.: Äquivalenzklassenbildung, Zustandsbasierter Test
  - ✓ Diversifizierender Test
    - Bsp.: Regressionstest

Testarten

- Modultest (Unit Test)

  Methodentest, Klassentest

- Integrationstest

  Bottom-up oder Top-down-Integrationstest

- Systemtest

  Funktionstest, Leistungstest, Stresstest

Besonderheiten OO-Systeme

- Botschaften zwischen Objekten
- Objektzustände
- Kapselung
- Polymorphie
- Dynamisches Binden
- Vererbung

Testen von OO-Software

- Kleinste testbare Einheiten sind Klassen.
- Getestet wird über die Schnittstelle bzw. über die Methoden der Klasse.
- Klassentest als OO Modultest (Unit Test).
- Testbarkeit wird durch möglichst kleine und möglichst unabhängige Einheiten gefördert.

Unit Tests

- Treiber

  Spez. Testteiber: Dummy- und Mock-Objekte

- Testrahmen

  Einbettung eines Testmoduls in eine künstliche Testumgebung.

- Automatisierte Unit Tests

  Weitverbreitetes Testframework: JUnit

2. Teil:
JUnit - Einführung

JUnit: Ein Open Source Software

- JUnit ist ein Java-Framework zum Schreiben und Ausführen automatischer Unit Test.
- JUnit erleichtert den Test-First-Ansatz.

- *http://www.junit.org*
- *http://www.frankwestphal.de/UnitTestingmitJUnit.html*

Kleine Begriffslehre

- Testfälle
- Testmethoden
- Fixture
- Methode setUp() und tearDown()
- Testsuiten

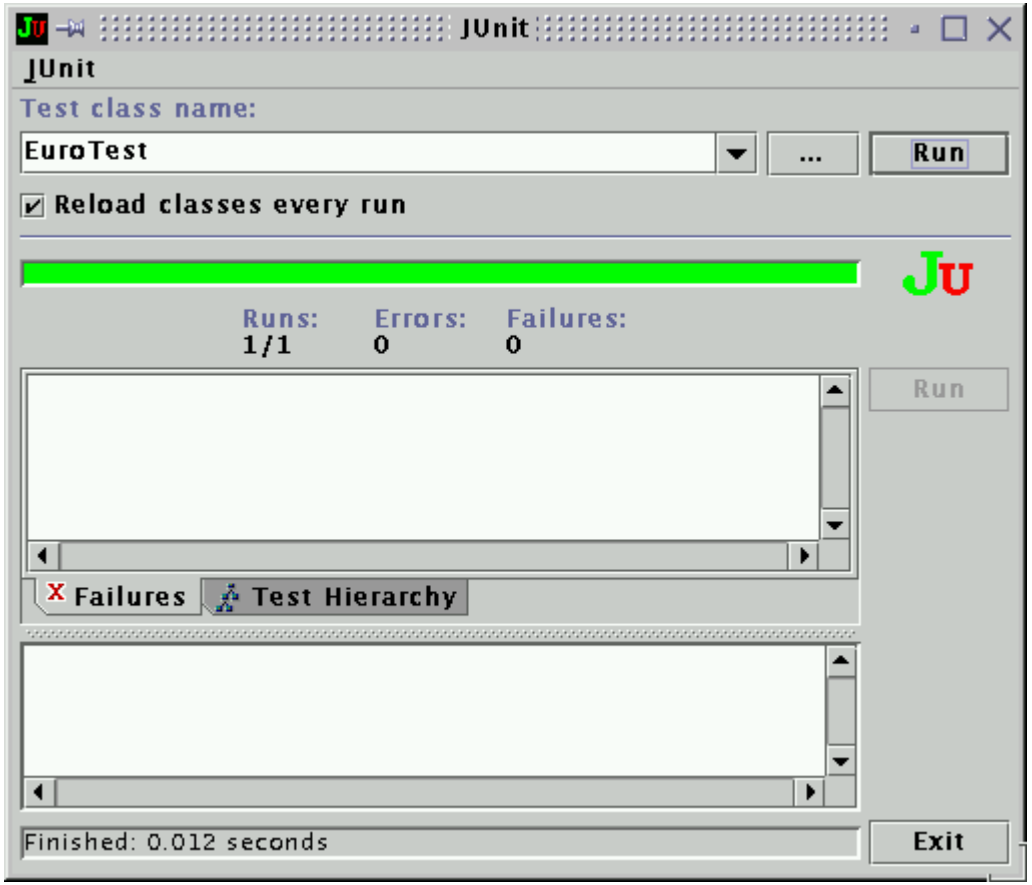
Ein erstes Beispiel

```
import junit.framework.*;
public class EuroTest extends TestCase {
    public EuroTest( String name ) { super( name ); }
    public void testAmount( ) {
        Euro two = new Euro( 2.00 );
        assertEquals( "two", 2.00, two.getAmount( ) ); }
    public static void main( String[ ] args ) {
        junit.swingui.TestRunner.run( EuroTest.class ); }
}
```

Erst testen, dann programmieren!

```
public class Euro {
    private double amount;

    public Euro( double amount ) {
        this.amount = amount; }
    public double getAmount( ) {
        return this.amount; }
}
```

JUnits graphische Oberfäche



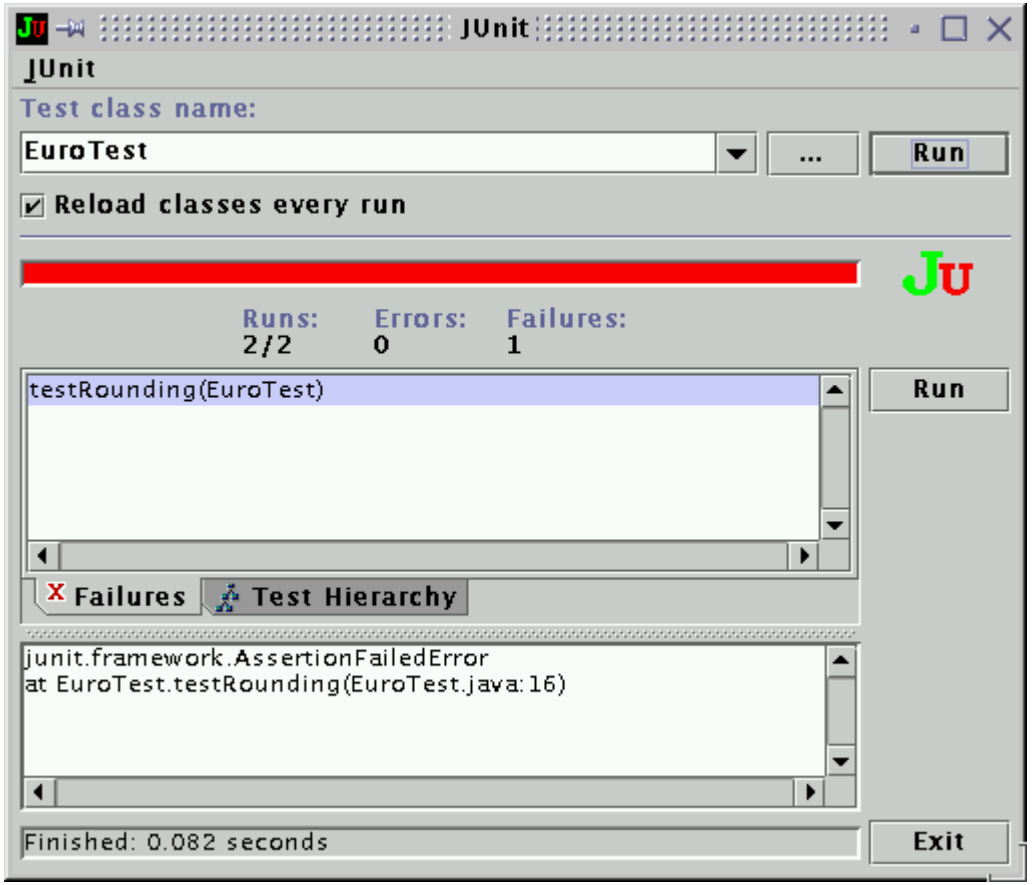
Testen mit JUnit

- assertTrue(theJungleBook.isChildrensMovie( ));
- assertEquals("foobar", "foo" + "bar");
- assertEquals(9, customer.getFrequentRenterPoints( ));
- assertEquals(3.1415, Math.pi( ), 1e-4);
- assertNull(hashMap.get(key));
- assertNotNull(httpRequest.getParameter("action"));
- assertSame(bar, hashMap.put("foo", bar).get("foo"));

Fehlschlagen eines Tests (1)

```
public class EuroTest extends TestCase {

    ...

    public void testRounding( ) {

        Euro roundedTwo = new Euro( 1.995 );

        assertEquals( "two", 2.00, two.getAmount( ) );

    }

}
```

Fehlschlagen eines Tests (2)



Fehlschlagen eines Tests (3)

```
public class Euro {

    private long cents;


    public Euro( double euro ) {

        cents = Math.round( euro * 100.0 ); }

    public double getAmount( ) {

        return cents / 100.0; }

}
```

Ein Testfall in JUnit (1)

```
public class EuroTest extends TestCase {

    ...

    public void testAdding( ) {

        Euro two = new Euro( 2.00 );

        Euro three = two.add( new Euro( 1.00 ) );

        assertEquals( "sum", 3.00, three.getAmount( ), 0.001 );

    }

}
```

Ein Testfall in JUnit (2)

```
public class Euro {

    ...

    private Euro( long cents ) { this.cents = cents; }

    public Euro add( Euro other ) {

        return new Euro( this.cents + other.cents );

    }

}
```

Ein Testfall in JUnit (3)

```
public class EuroTest extends TestCase {
    private Euro two;
    public EuroTest( String name ) { super( name ); }
    protected void setUp( ) { two = new Euro( 2.00 ); }
    protected void tearDown( ) { }
    public void testAmount( ) {
        assertEquals( "two", 2.00, two.getAmount( ) ); }
    ...
}
```

Lebenszyklus eines Testfalls

1. new EuroTest("testAdding")
2. setUp( )
3. testAdding( )
4. tearDown( )
5. new EuroTest("testAmount")
6. setUp( )
7. testAmount( )
8. tearDown( )
9. new EuroTest("testRounding")
10. setUp( )
11. testRounding( )
12. tearDown( )

Testen von Exceptions (1)

```
public class EuroTestextends TestCase {
    ...
    public void testNegativeAmount( ) {
        try {
            final double NEGATIVE_AMOUNT = - 2.00;
            new Euro( NEGATIVE_AMOUNT );
            fail( "Should have raised an IllegalArgumentException" ); }
        catch ( IllegalArgumentException expected ) { }  }
}
```

Testen von Exceptions (2)

```
public class Euro {

    ...

    public Euro( double euro ) {

        if ( euro < 0.0 ) {

            throw new IllegalArgumentException( "Negative amount" );

        }

        cents = Math.round( euro * 100.0 ); }

}
```

Fazit

- Testing can show the presence of errors,

  but never their absence.

  (Edsger W. Dijkstra)

Danke für die Aufmerksamkeit!