Unit Test

Das Testen von Projekten

Unit Test

- Jede irgendwie interessante Software ist nicht fehlerfrei
- Ein guter Entwurf eines OO Softwaresystems ist eine gute Voraussetzung, ...
- ...verhindert aber nicht, dass Fehler auftreten.
- Wie findet man Fehler?

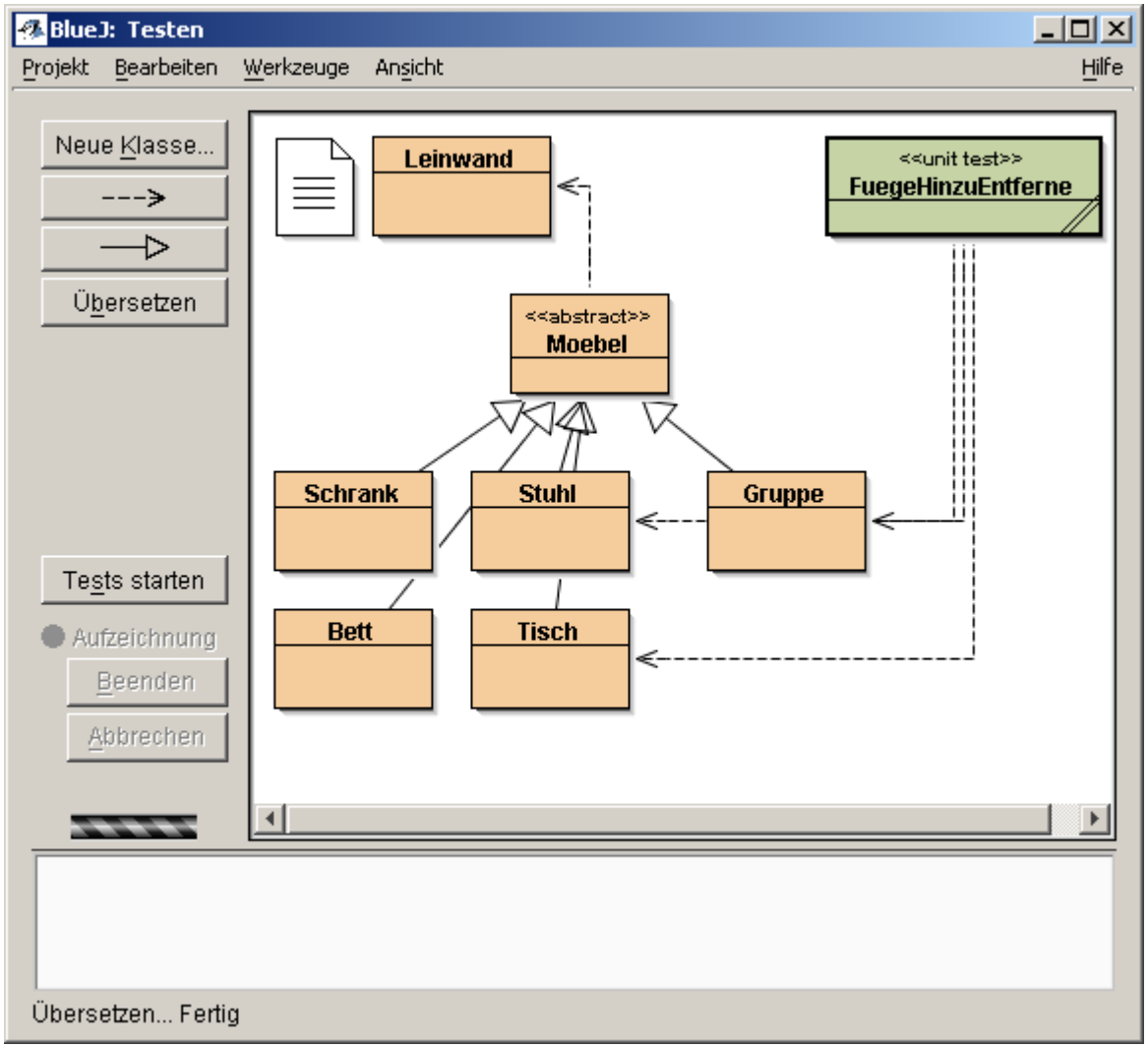
Unit Test

BlueJ nutzt ein „Fremdwerkzeug“: JUnit

The concepts of unit testing and regression testing are old, but their popularity was greatly increased recently with the publication of the eXtreme programming methodology1 and a unit testing tool for Java, *JUnit*.

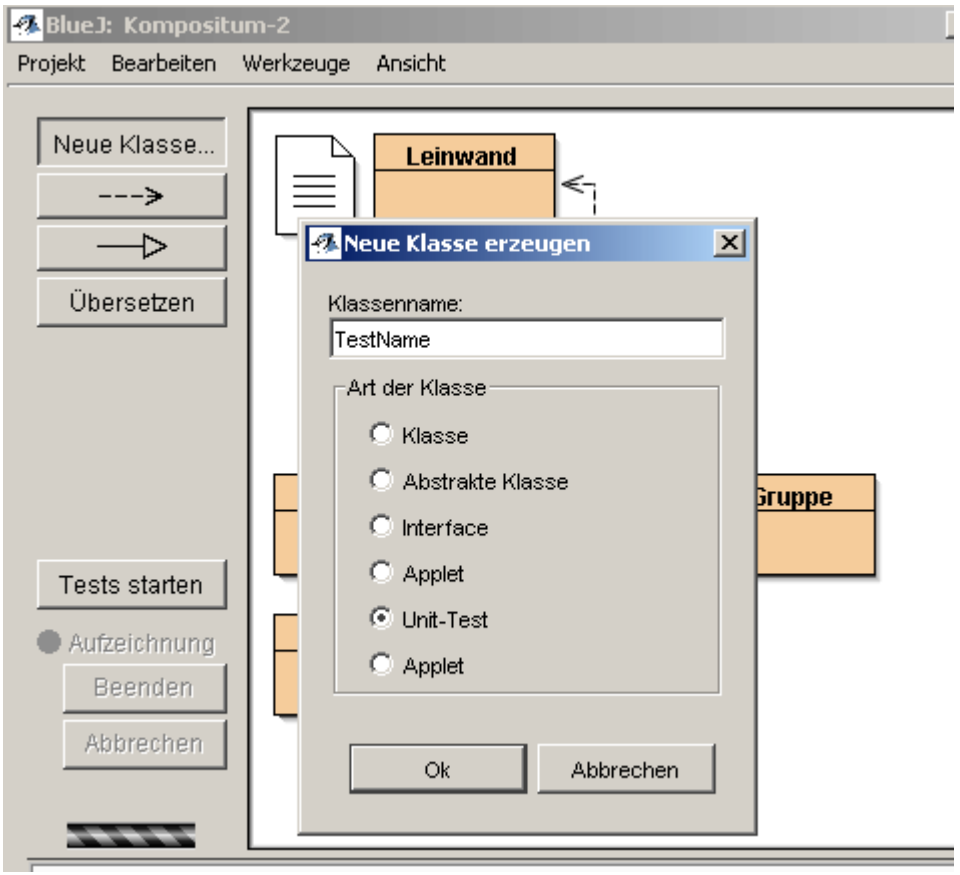
JUnit is a regression testing framework written by Erich Gamma and Kent Beck. You can find the software and a lot of information about it at *http://www.junit.org* .

Unit Test


BlueJ: Testen
Projekt
Bearbeiten
Werkzeuge
Ansicht
Hilfe
Neue Klasse...
Übersetzen
Tests starten
Aufzeichnung
Beenden
Abbrechen
Leinwand
<<unit test>>
FuegeHinzuEntferne
<<abstract>>
Moebel
Schrank
Stuhl
Gruppe
Bett
Tisch
Übersetzen... Fertig

Unit Test

Einfügen einer Testklasse



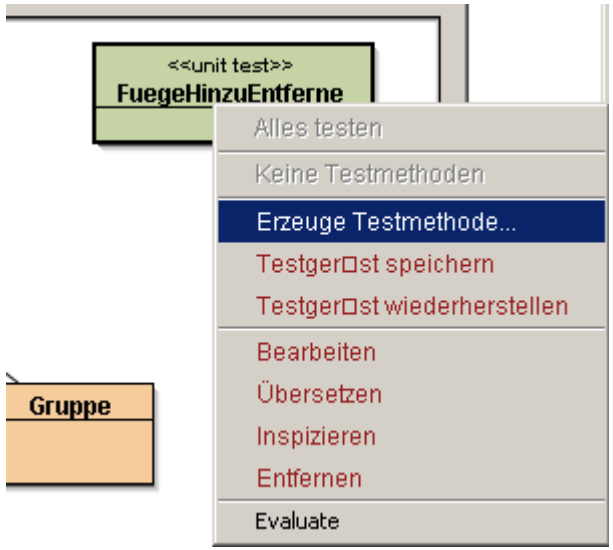
Unit Test

Ein Blick auf den Quelltext

```
9  public class FuegeHinzuEntferne extends junit.framework.TestCase
10 {
11     /**
12      * Konstruktor fuer die Test-Klasse FuegeHinzuEntferne
13      */
14     public FuegeHinzuEntferne()
15     {
16     }
17
18     /**
19      *  Setzt das Testgeruest fuer den Test.
20      *
21      * Wird vor jeder Testfall-Methode aufgerufen.
22      */
23     protected void setUp()
24     {
25     }
26
27     /**
28      * Gibt das Testgeruest wieder frei.
29      *
30      * Wird nach jeder Testfall-Methode aufgerufen.
31      */
32     protected void tearDown()
33     {
34     }
35 }
```

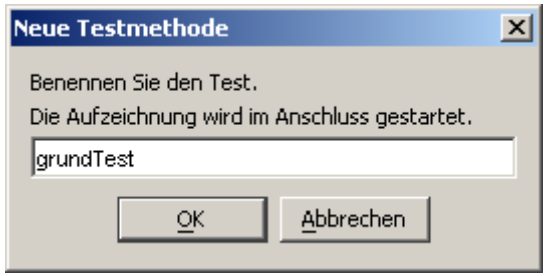
Unit Test

Übersetzen und
Erstellen einer Testmethode



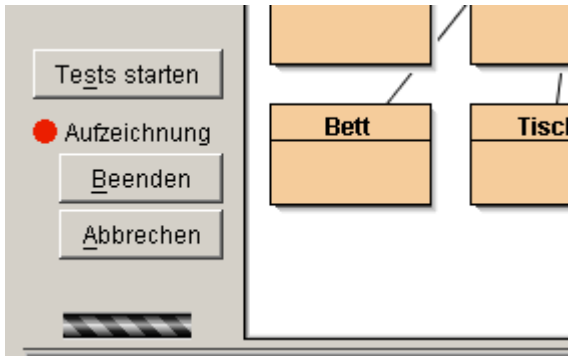
Unit Test

Erstellen einer Testmethode



Unit Test

Nun kann die Testmethode aufgezeichnet werden



Unit Test

Die Testmethode besteht aus:

- Erzeugen eines Objektes gruppe1
- Erzeugen eines Objektes stuhl1
- Erzeugen eines Objektes tisch1
- Aufruf der Methode fuegeHinzu(stuhl1) bei gruppe1
- Aufruf der Methode zeige() bei gruppe1
- Aufruf der Methode fuegeHinzu(tisch1) bei gruppe1
- Aufruf der Methode entferne(stuhl1) bei gruppe1

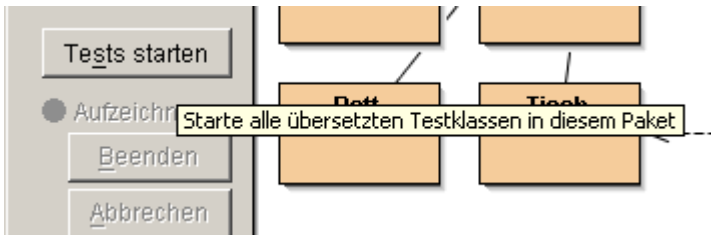
Unit Test

Der Quelltext der Testmethode enthält nun:

```
public void testGrundTest()
{
   Gruppe gruppe1 = new Gruppe();
   Stuhl stuhl1 = new Stuhl();
   Tisch tisch1 = new Tisch();
   gruppe1.fuegeHinzu(stuhl1);
   gruppe1.zeige();
   gruppe1.fuegeHinzu(tisch1);
   gruppe1.entferne(stuhl1);
}
```

Unit Test

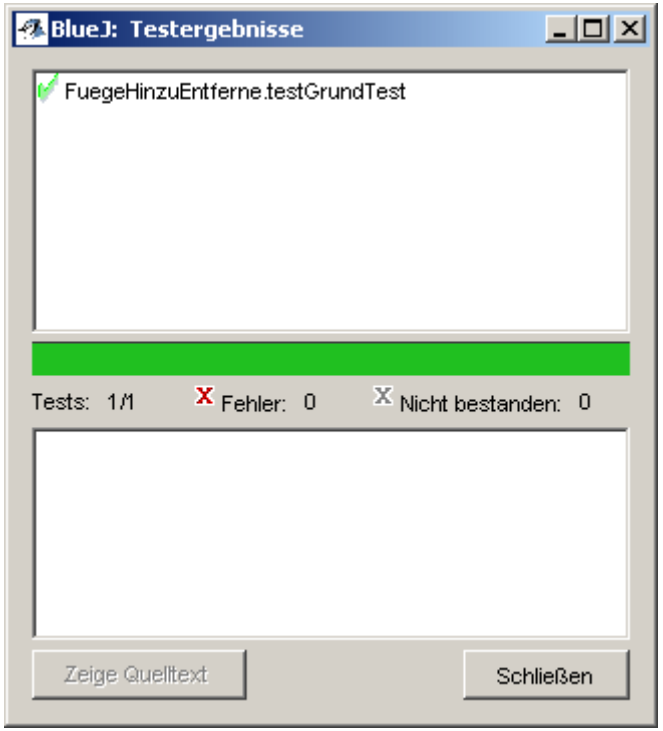
Nun kann die Testmethode durch einen Klick auf den Button „Tests starten“ gestartet werden



Unit Test

Der Test ist ziemlich uninteressant,
da eigentlich nur die Grundfunktionen
getestet werden

Das Ergebnis ist aber OK.



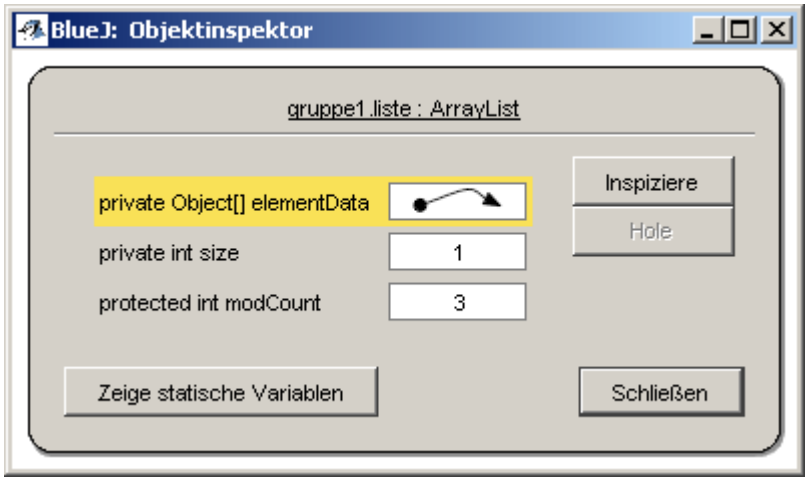
Unit Test

Wir fügen eine ernsthafteTestmethode hinzu:

- Erzeugen eines Objektes gruppe1
- Erzeugen eines Objektes stuhl1
- Aufruf der Methode fuegeHinzu(stuhl1) bei gruppe1
- Aufruf der Methode zeige() bei gruppe1
- Erneuter Aufruf der Methode fuegeHinzu(stuhl1) bei gruppe1
- Aufruf der Methode entferne(stuhl1) bei gruppe1

Unit Test

- Der Test verläuft erfolgreich
- Was heißt das aber in diesem Fall?
- Welchen Zustand hat gruppe1 nun?
- Egal, wie Sie diese Frage beantworten:
  Das Testergebnis ist unbefriedigend
- Warum?



Unit Test

- Der Test verläuft erfolgreich !!!
- Das hätte er aber nicht sollen !!!
- Unsere Klasse Gruppe enthält nämlich den Fehler, dass man ein Objekt erneut hinzufügen kann.
- *Hierin steckt übrigens ein durchgehaltenes Konzept, dass die Schülerinnen und Schüler keine perfekten Klassen vorgesetzt bekommen, sondern solche, an denen sie Fehler erkennen und selbst korrigieren können.*

Unit Test

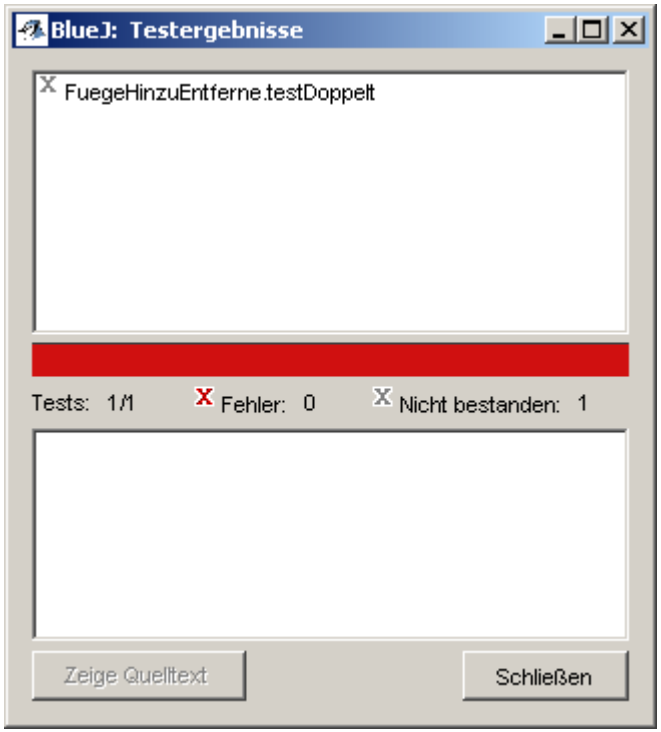
- Wir könnten zwar nun erst einmal unseren Fehler korrigieren, sollten aber besser zunächst unsere Testklasse verbessern.
- Um dann aber einfach testen zu können, führen wir bei fuegeHinzu(...) ein redesign durch:
- Wir lassen die Methode einen Rückgabewert geben vom Typ boolean, wenn sie das Objekt hinzugefügt hat.
- Beide Tests verlaufen natürlich weiterhin (leider) erfolgreich.

Unit Test

- Nun aber fügen wir dem Test Doppelt eine Testbedingung hinzu:
- assertTrue(gruppe1.fuegeHinzu(stuhl1));
- Damit wird eine Zusicherung (assertion) getestet, in diesem Fall, ob der Wert true war.
- Wieder ist der Test erfolgreich.
- Wir haben fuegeHinzu(...) ja auch noch nicht korrigiert.

Unit Test

- Viel interessanter ist es, wenn wir dem Löschen eine Zusicherung hinzufügen:
- Wir fordern, dass man ein Objekt nicht zweimal nacheinander entfernen kann.
- Nun erzeugt der Test einen Fehler!



Unit Test

Wir schreiben eine Methode enthalten ...

```
private boolean enthalten(Object gesucht)
  {
    for (Iterator it = liste.iterator();
            it.hasNext();){
       Object o = it.next();
       if (o.equals(gesucht))
          {return true;};
       }
    return false;
  }
```

... die prüft, ob ein Objekt schon enthalten ist.

Unit Test

Nun sollten wir nach einer Korrektur von

```
public boolean fuegeHinzu(Moebel moebel)
   {
      if (! enthalten(moebel)){
         ... ;
         return true;
      }
      return false;
  }
```

korrekt testen können.

Unit Test

Die Testmethode enthält nun den Code

```
public void testDoppelt(){
   Gruppe gruppe1 = new Gruppe();
   Stuhl stuhl1 = new Stuhl();
   assertTrue(gruppe1.fuegeHinzu(stuhl1));
   gruppe1.zeige();
   assertTrue(! gruppe1.fuegeHinzu(stuhl1));
   assertTrue(gruppe1.entferne(stuhl1));
   assertTrue(! gruppe1.entferne(stuhl1));
}
```

Unit Test

- Die Testmethoden kann man natürlich viel besser machen.
- Dies soll nur ein Anreiz sein, damit zu arbeiten.
- Beachten Sie dazu folgende weitergehende Konzeption (Quelle BlueJ Test-Tutorial):
  - Writing tests first
  - The eXtreme Programming methodology suggests that tests should be written before the implementation of any method.
  - Using BlueJ's unit test integration this can be done in two different ways.

Unit Test

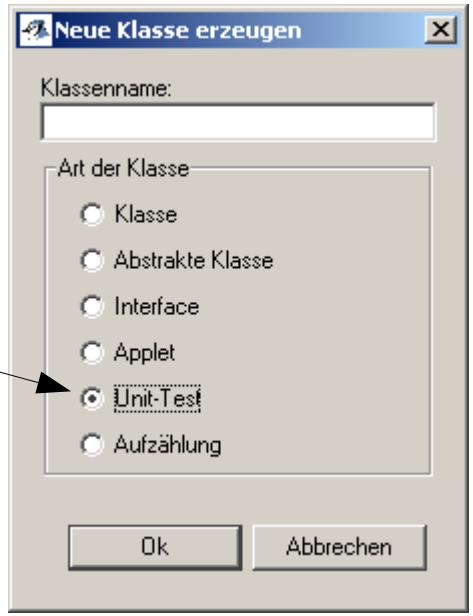
- Firstly, tests can be written by hand, as explained in the previous section. Test writing then works the same way as in non-BlueJ implementations of JUnit.
- Secondly, we can create method stubs in the reference class, returning dummy values for methods with non-void return types. After doing this, we can create tests using the interactive recording facility, and write assertions according to our expectations of the finished implementation.

Aufgabe zum Testen

- Erstellen Sie mit BlueJ ein neues Projekt mit dem Namen Zaehler
- Erstellen Sie eine neue Klasse und nennen Sie diese ebenfalls Zaehler
- Lassen Sie die Klasse übersetzen
- Diese Standardklasse hat – von den BlueJ – Entwicklern vorgegeben – schon eine Funktionalität
- Erzeugen Sie ein Objekt und testen Sie es

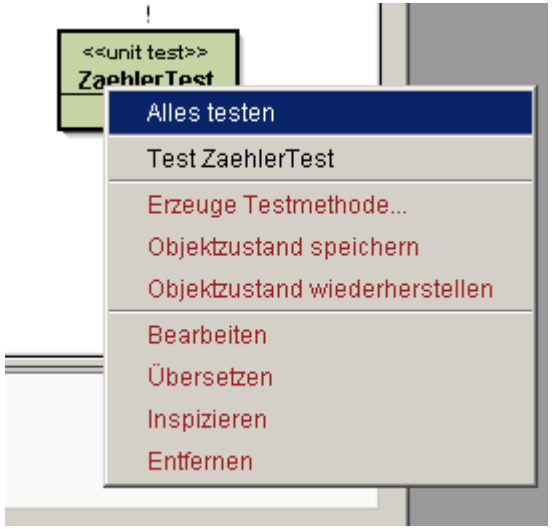
Aufgabe zum Testen

- Löschen Sie die in der Standardklasse vorgegebene Methode „beispielMethode“
- Erzeugen Sie eine neue Klasse
- Bei der Auswahl wählen Sie Unit-Test
- Nennen Sie diese z.B. ZaehlerTest



Aufgabe zum Testen

- Welche Eigenschaften soll die Klasse Zaehler erfüllen?
- Definieren Sie die Eigenschaften
- Erzeugen Sie eine Testmethode
- Formulieren Sie die Eigenschaften in der Testmethode aus



Aufgabe zum Testen

- Wenn ich ein Zählerobjekt erzeuge, soll es den Zählerstand 0 haben. ===> ???

```
assertEquals(zaehler1.zeigeStand(), 0);
```

- Passen Sie die Klassendefinition von Zaehler an! Was ist zu tun?

```
public int zeigeStand(){ return x; }
```

- Wenn ich dem Zähler „die message zaehle() schicke“,
  soll er den Zählerstand 1 haben. ===> ???

Aufgabe zum Testen

- Zunächst muss Zaehler die Methode zaehle() implementieren

```
public void zaehle() {
  }
```

- Der Test führt nun natürlich zu einem Fehler
- Vervollständigen Sie die Methode
- Erweitern Sie die Funktionalität:
  - Zähler zurücksetzen
  - Neu: Zyklischer Zähler