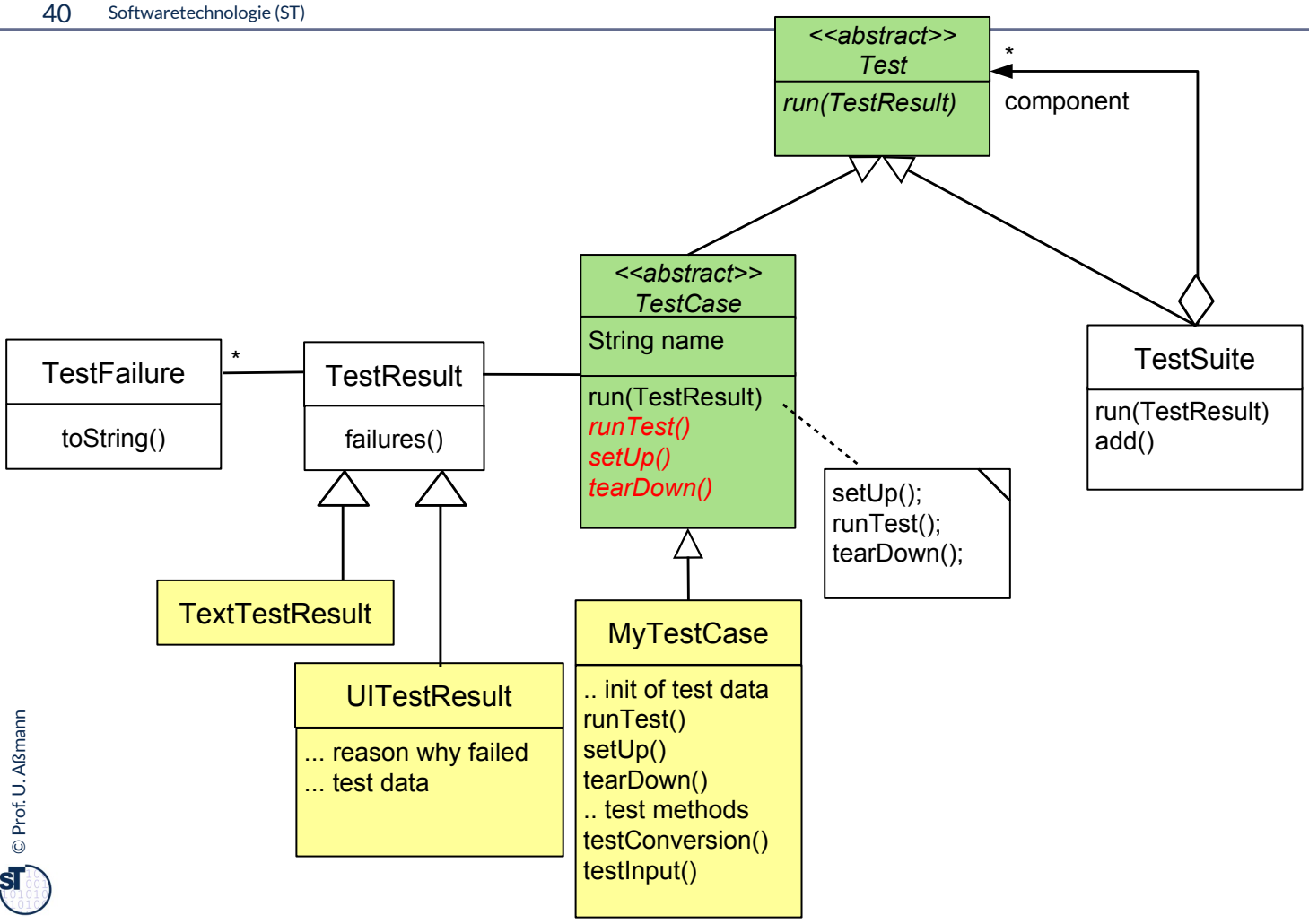
Das JUnit Regressionstest-Framework

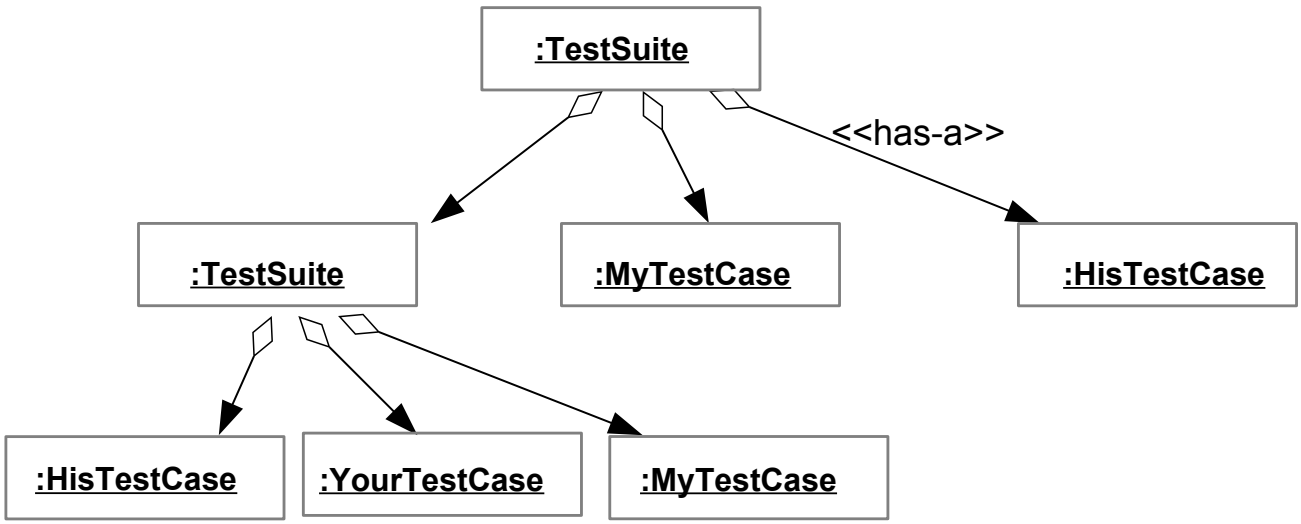
- Eine **Testsuite (Test-Ring)** werden heute aus Test-Frameworks gemacht
- **JUnit** www.junit.org ist ein technisches Java-Framework für Regressionstests, sowohl für einzelne Klassen (*unit test)*, als auch für Systeme
  - Durchführung von Testläufen mit Testsuiten automatisiert
  - Eclipse-Plugin erhältlich
  - Mittlerweile für viele Programmiersprachen nachgebaut
- Junit 3.8.1: Good old plain Java
  - 88 Klassen mit 7227 Zeilen
  - im Kern des Rahmenwerks: 10 Klassen (1101 Zeilen)
- Testresultate:
  - Failure (Zusicherung wird zur Laufzeit verletzt)
  - Error (Unvorhergesehenes Ereignis, z.B. Absturz)
  - Ok
- JUnit-4 versteckt mehr Funktionalität mit Metadaten (@Annotationen) und ist wesentlich komplexer. Empfehlung: Lernen Sie zuerst 3.8.1!
- Junit-5 ist zu komplex, nur für den Fortgeschrittenen geeignet!

Kern von JUnit 3.8.1



Laufzeit-Snapshot von TestSuite

- JUnit baut zur Laufzeit eine hierarchisch geschachtelte Suite von Testfällen auf



Exkurs: Erkunde JUnit 3.8.x mit Browser und Javadoc

- Aufgabe:
  - laden Sie die API-Dokumentation von JUnit mit einem Brauser Ihrer Wahl
  - finden Sie die Aufgabe der Klassen TestResult, TestCase und TestSuite heraus
  - Welche Aufgabe hat die Klasse Assert?

/home/ua1/Courses/ST1/Material/junit3.8.1/javadoc/index.html

Gesetz 68 [PP]:
Bauen Sie die Dokumentation ein, anstatt sie dranzuschrauben

Testfall der Datumsklasse in JUnit 3.8.x

- TestCases sind Methoden, beginnend mit der Markierung `test`
- Initialisierung der Halterung mit `setUp`, Abbau mit `tearDown`
- Testfallklassen sind also "Kundenklassen" von zu testenden Klassen
- Test mit assertTrue, geerbt von TestCase

```
public class DateTestCase extends TestCase {
    Date d1, d2, d3;
  protected void setUp() {
    d1 = new Date(„1. Januar 2006“);
    d2 = new Date(„01/01/2006“);
    d3 = new Date(„January 1st, 2006“);
  }
  public void testDate1() {
    // Processing
    d1.parseDate(); d2.parseDate(); d3.parseDate();
    .... more to say here ....
  }
  public void testDate2() { .. more to say here .... }
  protected void tearDown() {
    // Checking
    assertTrue(d1.equals(d2)); assertTrue(d2.equals(d3));
    assertTrue(d3.equals(d1));
  }
}
```

Halterung (fixture)

Benutzung von TestCase

- Von einem Java-Programm startet man die Testfälle wie folgt:
  - Ein Testfall wird nun erzeugt durch einen Konstruktor der Testfallklasse
  - Der Konstruktor sucht die Methode des gegebenen Namens (“testDate1”) und bereitet sie zum Start vor
    - mit *Reflektion*, d.h. Suche nach dem Methode in dem Klassenprototyp
  - Die *run()* Methode startet den Testfall gegen die Halterung und gibt ein TestResult zurück

```
public class TestApplication {
   ...
   TestCase tc = new DateTestCase(„testDate1“);
   TestResult tr = tc.run();
}
```

Testsuiten

- Eine Testsuite ist eine Kollektion von Testfällen
- TestSuites sind komposit

```
public class TestApplication {
   ...
   TestCase tc = new DateTestCase(„testDate1“);
   TestCase tc2 = new DateTestCase(„testDate2“);
   TestSuite suite = new TestSuite();
   suite.addTest(tc);
   suite.addTest(tc2);
   TestResult tr = suite.run();

   // Nested test suites
   TestSuite subsuite = new TestSuite();
   ... fill subsuite ...
   suite.addTest(subsuite);
   TestResult tr = suite.run();
}
```

Regressionstest testet die Anforderungen aus dem Pflichtenheft mit Testsuiten

- Im Idealfall führt jede Anforderung des Pflichtenhefts zu einem oder mehreren Testfällen, die in Testsuiten gesammelt werden
  - Und automatisiert überprüft werden können (nächtlicher Test).

*Testsuiten werden im **Akzteptanztest** dem Kunden vorgeführt,*
um zu zeigen, dass
alle seine Anforderungen erfüllt worden sind.

Eine ausreichend große und gute grüne Testsuite macht
den Entwickler sicher, dass der Abnahmetest funktionieren wird.

junit 3.8.1 TestRunner GUI

- Die Klassen `junit.awtui.TestRunner, junit.swingui.TestRunner` bilden einfach GUIs, die Testresultate anzeigen
  - `sh$ java -jar unit3.8.1/junit.jar junit.awtui.TestRunner DateTestCase¶`
- Von Java-Anwendung aus:
  - Gibt man einem Konstruktor eines Testfalls eine Klasse mit, findet er die “test*”-Methoden (die Testfallmethoden) selbständig
  - Dies geschieht mittels *Reflektion*, d.h. Absuchen der Methodentabellen im Klassenprototypen und Methodenspeicher

```
public class TestApplication {
  public static Test doSuite() {
    // Abbreviation to create all TestCase objects
    // in a suite
    TestSuite suite = new TestSuite(DateTestCase.getClass());
  }
  // Starte the GUI with the doSuite suite
  public static main () {
    junit.awtui.TestRunner.run(doSuite());
  }
}
```

13.5.2) Testläufe in Junit 4.X

- https://www.admfactory.com/junit-3-vs-junit-4-comparison/

/Users/uweassmann/Courses/ST1/Material/junit4.4/javadoc/index.html
http://docs.oracle.com/javase/tutorial/java/annotations/index.html

Neuer Testfall in Junit 4.X mit Metadaten-Annotationen

- TestCase-Methoden werden mit **Metadaten-Annotationen** gekennzeichnet, Annotationen an Attribute und Methoden, die mit @ beginnen
- Vorteil: Testmethoden und ihre Testees können gemeinsam auftreten

```
import org.junit.*;        // Import of the annotations
public class DateTestCase /* no superclass is necessary */ {
   Date d1;
   Date d2;                          Halterung (fixture)
   Date d3;
  @Before protected int setUp() {
    d1 = new Date(„1. Januar 2006“);
    d2 = new Date(„01/01/2006“);
    d3 = new Date(„January 1st, 2006“);
  }
  @Test public int compareDate1() {
    // Processing
    d1.parseDate(); d2.parseDate(); d3.parseDate();
    // Checking
    assertTrue(d1.equals(d2)); assertTrue(d2.equals(d3));
    assertTrue(d3.equals(d1));
    .... more to say here ....
  }
  @Test public int compareDate2() {
    .... more to say here ....
  }
}
```

Benutzung von Testfall-Klasse in 4.x

- Von der Kommandozeile:
  - `$ java org.junit.runner.JUnitCore DateTestCase¶`
- Von Eclipse aus: In einer IDE wie Eclipse werden die Testfall-Prozeduren automatisch inspiziert und gestartet
- Von einem Java-Programm aus:
  - Ein Testfall wird erzeugt durch einen Konstruktor der Testfallklasse
  - Suche den Klassenprototyp der Testfallklasse
  - Die *run()* Methode von JUnitCore startet alle enthaltenen Testfälle über den Klassenprotoypen
    - Starten aller annotierten Initialisierungen, Testfallmethoden, Abräumer
  - und gibt ein "Result"-Objekt zurück

```
public class TestApplication {
   ...
   DateTestCase tc = new DateTestCase();
   // getClass() holt den Klassenprototypen
   Result tr = JUnitCore.run(tc.getClass());
}
```

Junit 4.X mit vielen weiteren Metadaten-Annotationen

- Viele weitere Test-Annotationstypen sind definiert

```
public class DateTestCase {
    Date d1;
  @BeforeClass protected int setUpAll() {
    // done before ALL tests in a class
  }
  @AfterClass protected int tearDownAll() {
    // done before ALL tests in a class
  }
  @Test(timeout=100,expected=IndexOutOfBoundException.class)
  public int compareDate2() {
    // test fails if takes longer than 50 msec
    // test fails if IndexOutOfBoundException is NOT thrown
    .... more to say here ....
  }
}
```

Für Interessierte: Was die Annotationen tun

- Junit 4 ist ein *Codegenerator-Framework (Metaprogramm-Framework*, siehe Vorlesung CBSE im SoSe), das sich im Java-Comiler registriert.
- Liest der Compiler die Annotationen, ruft er eine Prozedur aus dem jUnit-Codegenerator auf und übergibt die zu übersetzende Methode als Parameter:

```
@test →   call  Junit4.evaluateTestAnnotation(Method orig);
```

- Wobei der Codegenerator derart definiert ist:

```
// Reflective, metaprogramming code
public class Junit4 {
   Method evaluateTestAnnotation(Method orig) {
      Method testOrig = orig.clone();
      // somehow modify „testOrig“ to be a new
      // Test-Case method and put it into a TestCase
      // class
      return testOrig;
   }
}
```

13.6. Entwurfsmuster in JUnit

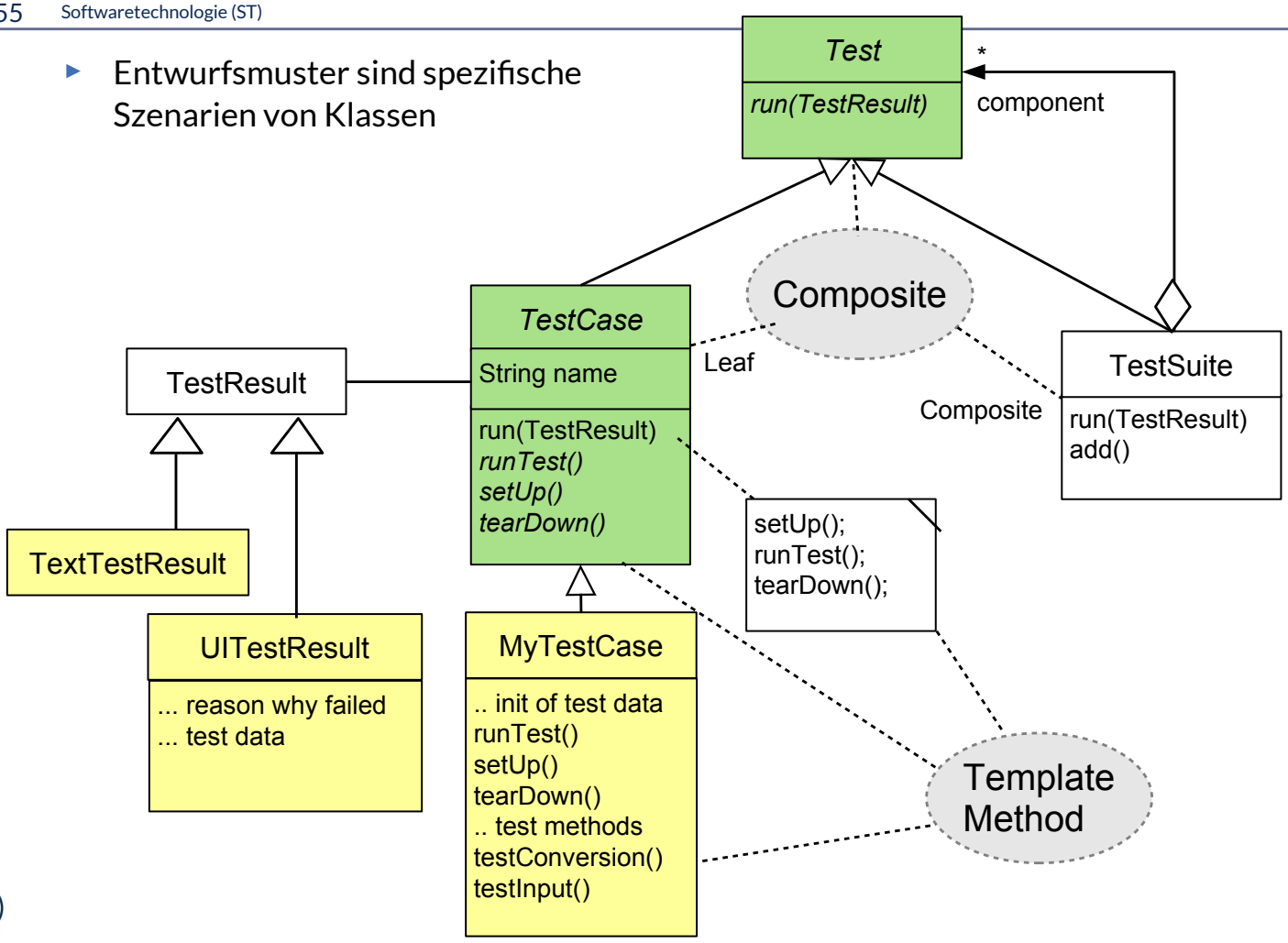
Was ist ein Entwurfsmuster?

Def.: Ein **Entwurfsmuster** beschreibt eine Standardlösung für
- ein Standardentwurfsproblem
- in einem gewissen Kontext

- Ein Entwurfsmuster wiederverwendet bewährte Entwurfsinformation
  - Ein Entwurfsmuster darf nicht *neu*, sondern muss wohlbewährt sein
- Ein Entwurfsmuster enthält mindestens:
  - Klassendiagramm der beteiligten Klassen
  - Objektdiagramm der beteiligten Objekte
  - Interaktionsdiagramm (Sequenzdiagramm, Kommunikationsdiagramm)
- Entwurfsmuster sind ein wesentiches Entwurfshilfsmittel aller Ingenieure
  - Maschinenbau – Elektrotechnik - Architektur
- Entwurfsmuster treten auch in Frameworks wie JUnit auf

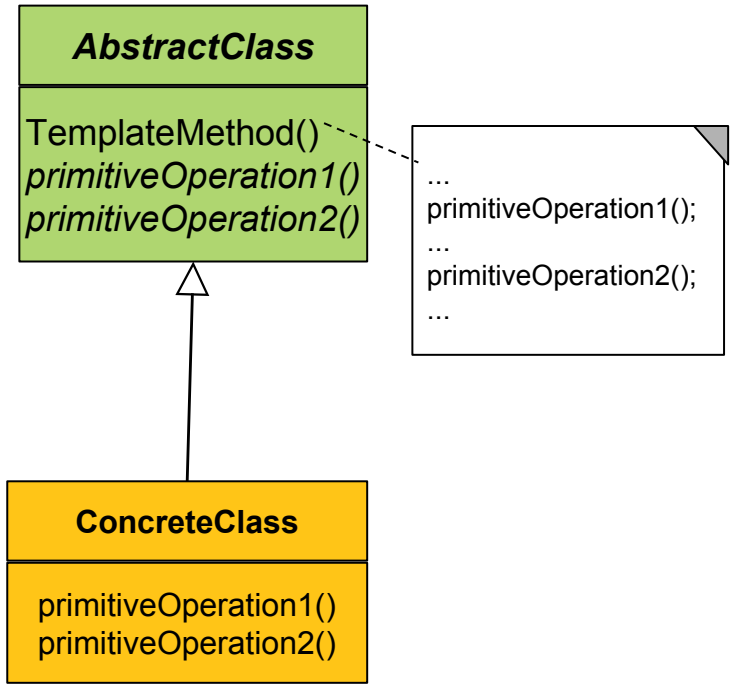
Beispiel: Entwurfsmuster in Junit 3.x

- Entwurfsmuster sind spezifische Szenarien von Klassen



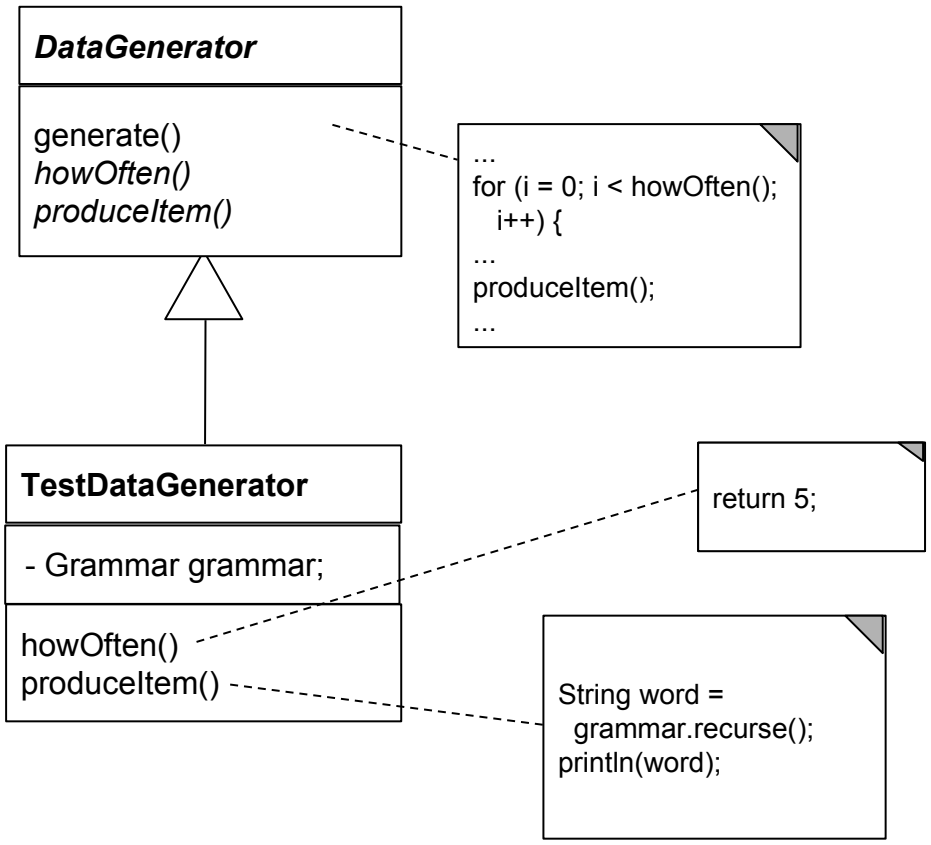
Entwurfsmuster TemplateMethod

- Definiert das Skelett eines Algorithmusses in einer *Schablonenmethode (template method)*
  - Die Schablonenmethode ist konkret
- Delegiere Teile zu abstrakten *Hakenmethoden (hook methods)*
  - die von Unterklassen konkretisiert werden müssen
- Variiere Verhalten der abstrakten Klasse durch verschiedene Unterklassen
  - Separation des “fixen” vom “variablen” Teil eines Algorithmus



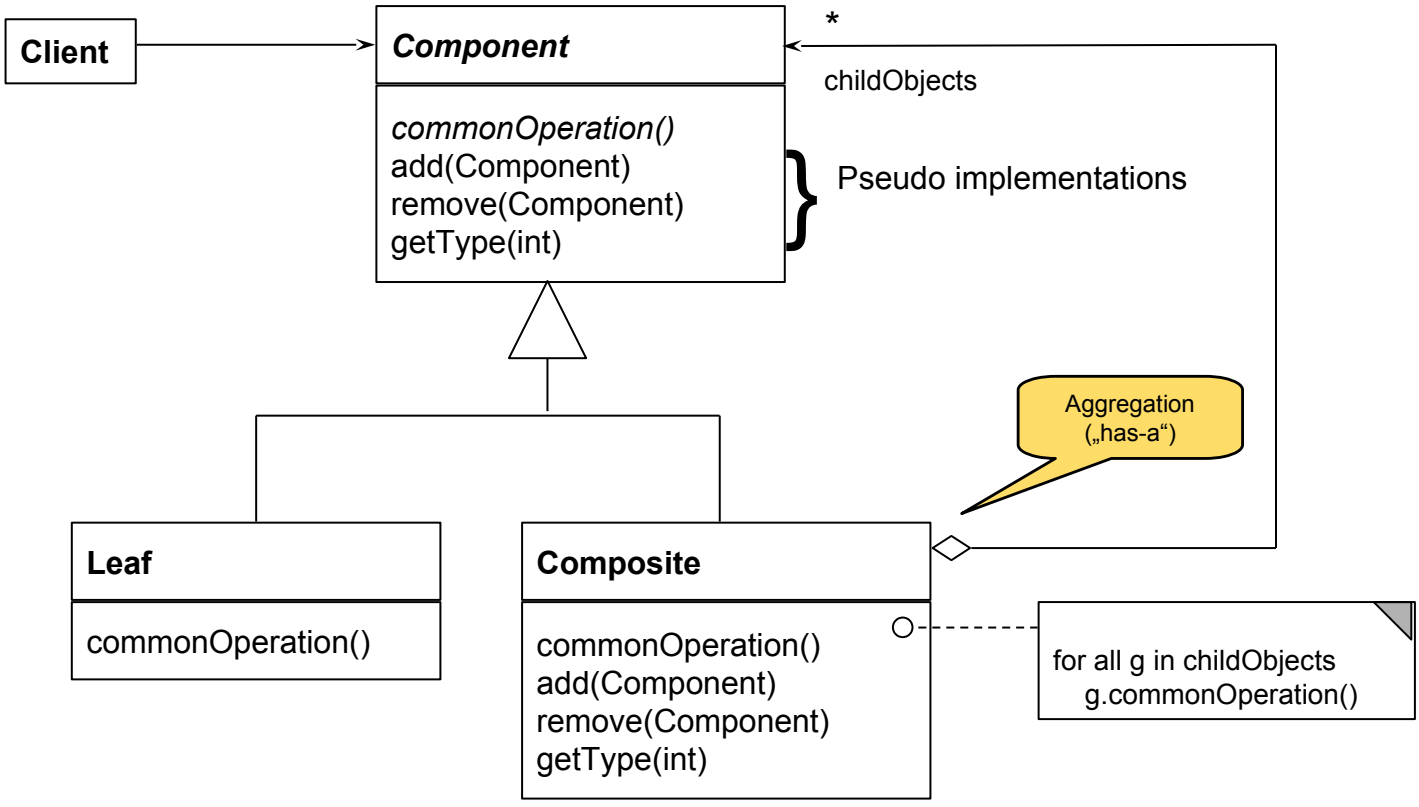
Beispiel TemplateMethod: Ein Datengenerator

- Parameterisierung eines Generators mit Anzahl und Produktion
  - (Vergleiche mit `TestCase` aus JUnit)



Entwurfsmuster Composite

- Composite besitzt eine rekursive n-Aggregation zur Oberklasse

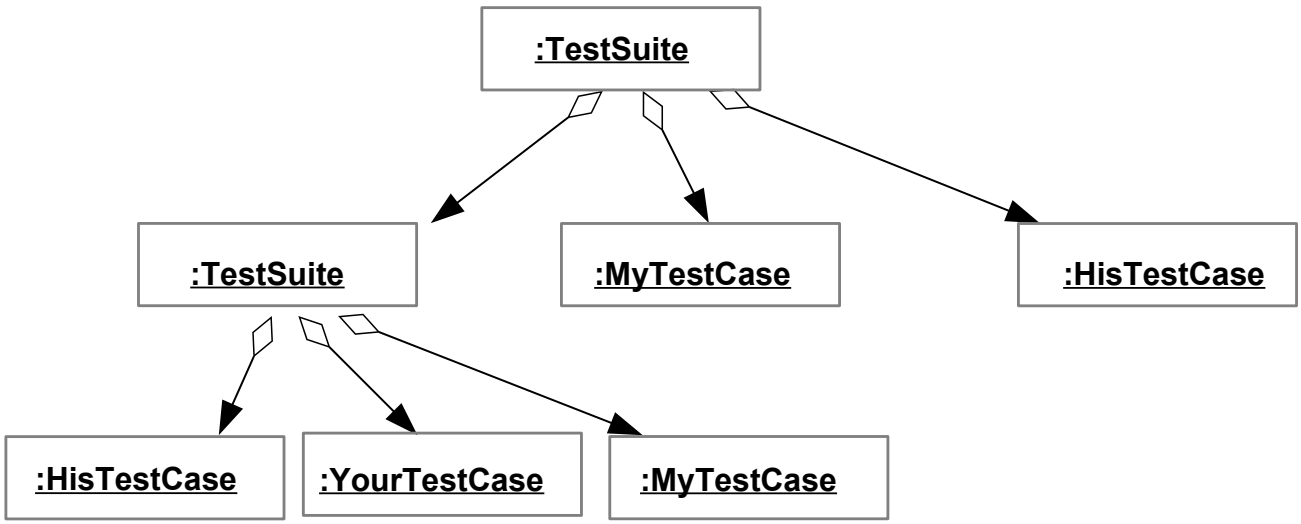


Composite in Junit 3.x

- Mehrere Methoden von *Test* sind komposit strukturiert
  - run()
  - countTestCases()
  - tests()
  - toString()

Laufzeit-Snapshot von Composite

- Composite beschreibt Ganz/Teile-Hierarchien von Laufzeit-Objekten, z.B. geschachtelte Testsuiten und -fälle



Bsp.: Zählen von Testfällen in JUnit


```
abstract class Test {
  abstract int countTestCases();
}
class TestSuite extends Test {;
  Test [20] children;// here is the n-recursion
  int countTestCases() { // common operation
    for (i = 0; i <= children.length; i++) {
      curNr += children[i].countTestCases();
    }
    return curNr;
  }
  void add(Test c) {
     children[children.length++] = c;
  }
```

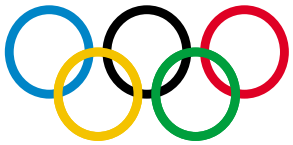
```
class TestCase extends Test {
   private int myTestCaseCount = 10;
   int countTestCases() { // common operation
      return  myTestCaseCount;
   }
   void add(Test c) {
       /// impossible, dont do anything
   }
}

// application
main () { int nr = test.countTestCases(); }
```

Funktionales Programmieren:
- Iteratoralgorithmen (map)
- Faltungsalgorithmen (folding)

Praktikum Wintersemester

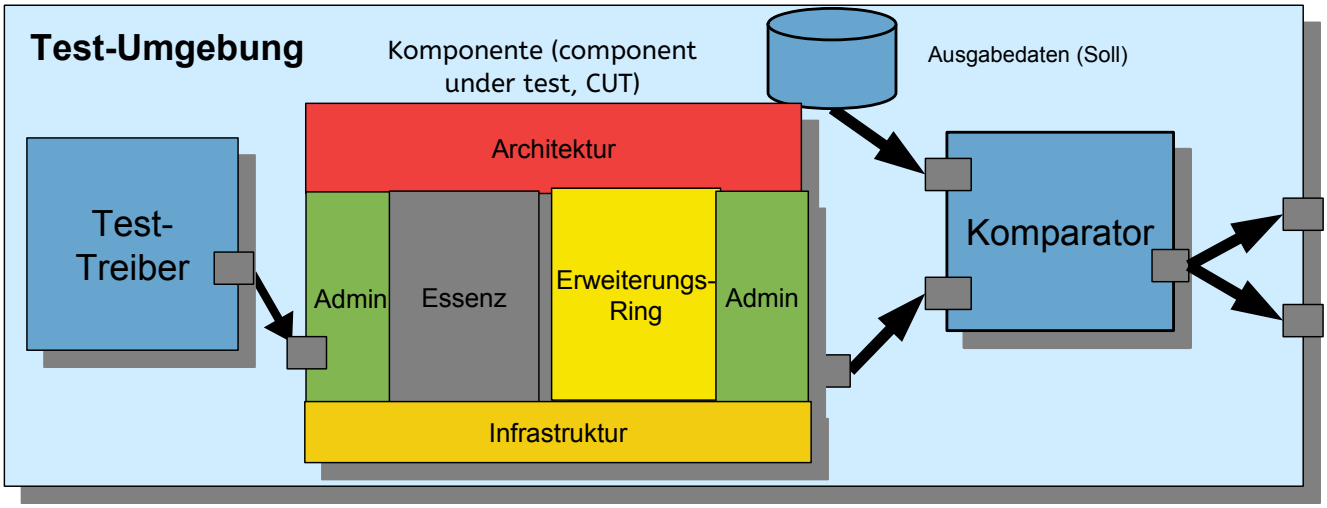
- Erstellung eines Akzeptanztestbeschreibung im Vertrag (Pflichtenheft)
  - Ohne Erfüllung kein Bestehen des Praktikums!
  - Eine Iteration: Kunde stellt einen Zusatzwunsch: Wie reagiert man auf die Veränderung?
- **Tip**: Erstellen Sie sich von Anfang an einen Regressionstest!
  - Und lassen sie diesen bei jeder Veränderung laufen, um zu überprüfen, ob Sie wesentliche Eigenschaften des Systems verändert haben

Biologische Software: Erweiterbare Komponenten (Frameworks)



Def.: Software-Frameworks (-Rahmenwerke) sind *erweiterbare* Komponenten, d.h. erweiterbare und wiederverwendbare Programmeinheiten. Sie sind Ergebnis „biologischer" Softwareentwicklung. Im einfachsten Fall sind sie Klassenpakete mit „„olympischen Ringen":

- klaren Schnittstellen, Administrationsring und Infrastruktur,
- gut abdeckender Testtreiber-Hülle und
- **Erweiterungsring**, an den Erweiterungen eingebaut werden können.



Was haben wir gelernt?

- **Software ohne Tests ist keine Software**
- **Programme ohne Administration sind nicht nutzbar**
- **Ringe** sind querschneidende Schichten des Programms, die es für die Zukunft vorbereiten
- Achten Sie auf das Management Ihres Projekts im Praktikum
  - Planen Sie hinreichend
  - Testen Sie sorgfältig und von Anfang an (*test-driven development, TDD)*
  - Entwerfen Sie eine Testarchitektur, Akzeptanztestsuite, Regressionstest
- Erste Entwurfsmuster TemplateMethod, Composite
- Lernen Sie, Java zu programmieren:
  - Ohne ausreichende Java-Kenntnisse weder Bestehen der Klausur noch des Praktikums
  - Nutzen Sie fleissig das Java-INLOOP-System!

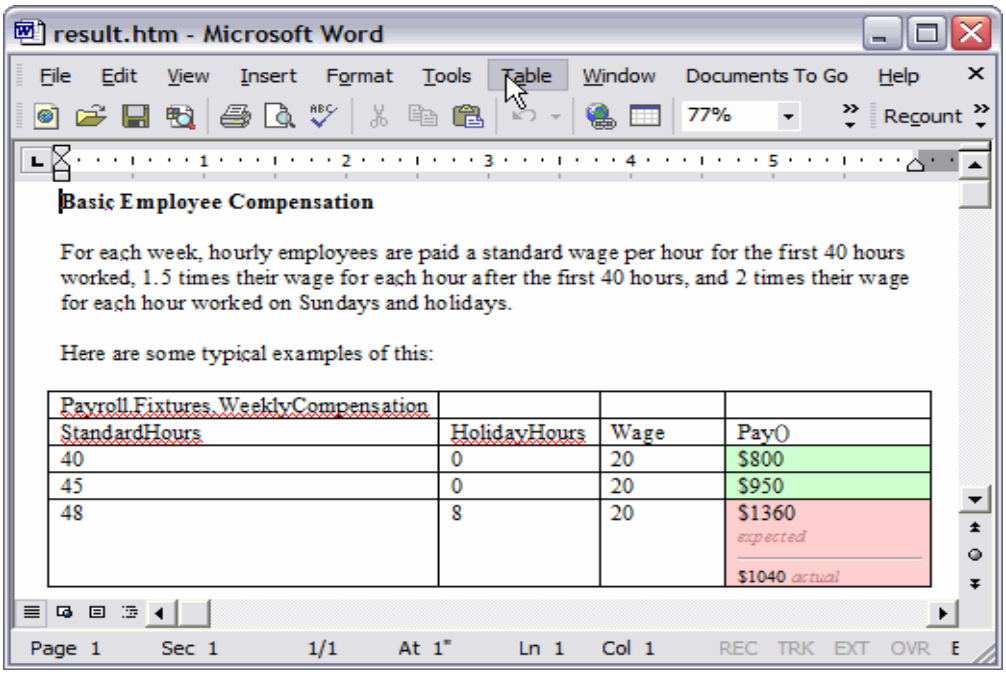
Verständnisfragen

- Erklären Sie den “Ring der Administration”.
- Welche Teil-Ringe besitzt der Ring der Administration?
- Wieso ist der Test-Ring für Wiederverwendung so wichtig?
- Was ist der Unterschied zwischen einer Klasse und einer Komponente?
- Was unterscheidet den Ring der funktionalen Essenz von den anderen Ringen?

Anhang

FIT Testfalltabellen Framework http://fit.c2.com

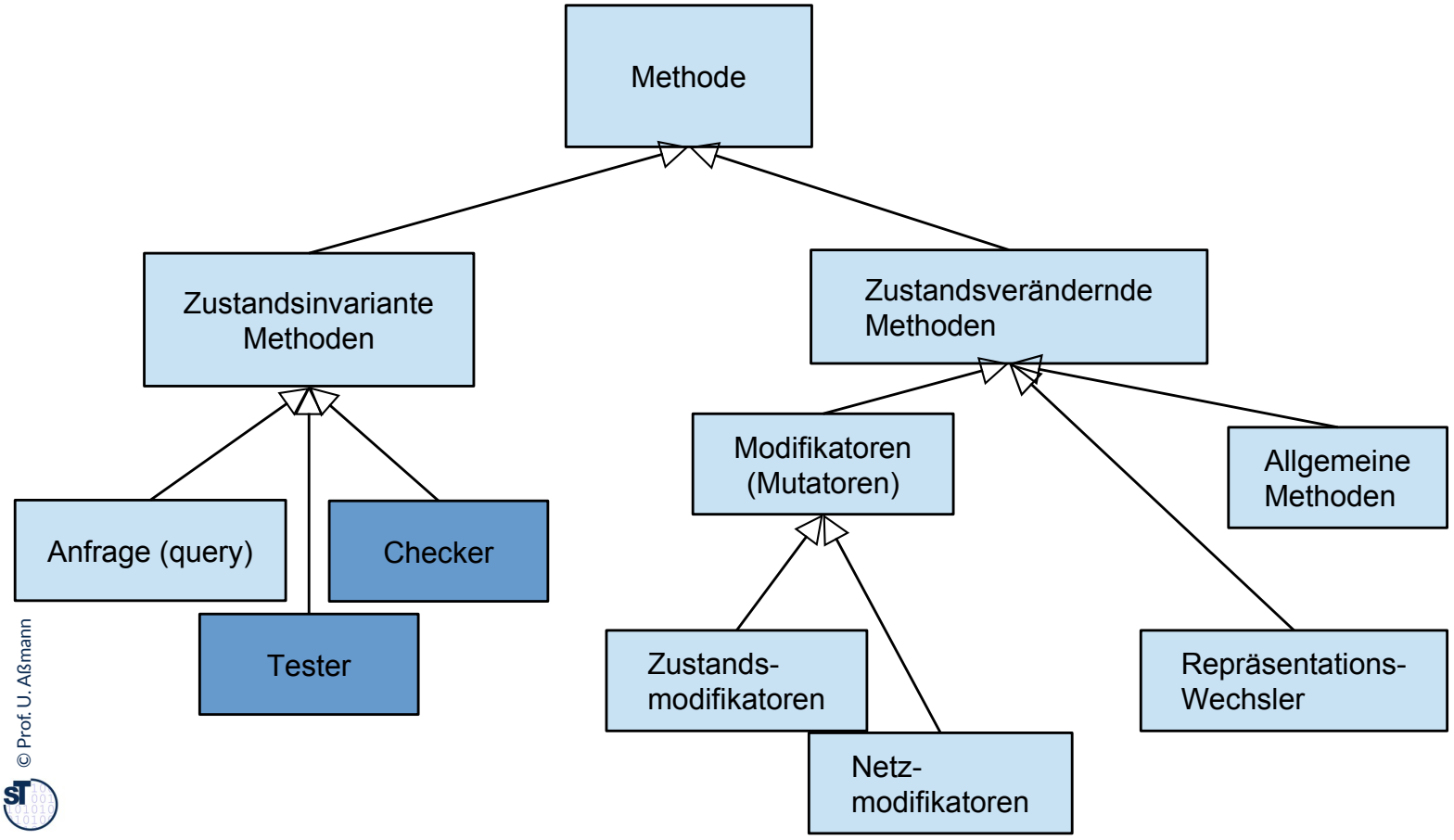
- FIT bietet eine Spezifikation der Testfälle in Word oder Excel
  - Automatische Generierung von Junit-Testfällen
  - Automatischer Feedback
- siehe Softwaretechnologie-II, WS



http://fit.c2.com/files/WelcomeVisitors/example.gif

Erweiterung: Begriffshierarchie der Methodenarten

- **Wiederholung**: Welche Arten von Methoden gibt es in einer Klasse?



Wie wähle ich Testdaten für Testfälle aus?

- Bestimme die **Extremwerte** der Parameter der zu testenden Methode
  - Nullwerte immer testen, z.B. 0 oder null
  - Randwerte, z.B. 1.1., 31.12
- Bestimme **Bereichseinschränkungen**
  - Werte ausserhalb eines Zahlenbereichs
  - negative Werte, wenn natürliche Zahlen im Spiel sind
- Bestimme **Zustände**, in denen sich ein Objekt nach einer Anweisung befinden muss
- Bestimme **Äquivalenzklassen** von Testdaten und teste nur die Repräsentanten
- Bestimme alle Werte aller **boolschen Bedingungen** in der Methode
  - Raum aller Steuerflußbedingungen

Even Worms are Tested

- StuxNet Tests in Israel
  - http://catless.ncl.ac.uk/Risks/26.31.html#subj3.1
- LAUSD payroll fiasco
  - http://catless.ncl.ac.uk/Risks/24.84.html
- Surprising reimplementation of system with good new tests:
  - http://catless.ncl.ac.uk/Risks/24.85.html#subj6.1

Bekannte Pannen

- Hamburg-Altona Bahnhof 1995
  - http://catless.ncl.ac.uk/Risks/16.93.html#subj1.1
  - http://catless.ncl.ac.uk/Risks/16.94.html#subj1.1
  - http://catless.ncl.ac.uk/Risks/17.02.html#subj3.1
- Toll Collect Krise 2004
  - http://catless.ncl.ac.uk/Risks/23.21.html#subj6.1
- Velaro-D-Züge von Siemens
  - http://www.sueddeutsche.de/wirtschaft/verspaetete-lieferung-von-ice-zuegen-eine-halbe-milliarde-euro-auf-dem-abstellgleis-1.1655927
  - http://www.nwzonline.de/wirtschaft/bericht-neue-siemens-ice-der-bahn-erhalten-zulassung_a_11,5,196943309.html

Edison, der Erfinder der Glühbirne

"If I find 10,000 ways something won't work, I haven't failed. I am not discouraged, because every wrong attempt discarded is another step forward."

Thomas A. Edison

"Müsste Edison eine Nadel im Heuhaufen finden, würde er einer fleißigen Biene gleich Strohhalm um Strohhalm untersuchen, bis er das Gesuchte gefunden hat."

- Nikola Tesla, New York Times, 19. Oktober 1931

Aber: Ein Wort der Warnung

> [Edison] had no hobby, cared for no sort of amusement of any kind and lived in utter disregard of the most elementary rules of hygiene. [...] His method was inefficient in the extreme, for an immense ground had to be covered to get anything at all unless blind chance intervened and, at first, I was almost a sorry witness of his doings, **knowing that just a little theory and calculation would have saved him 90% of the labour.**
>
> But he had a **veritable contempt for book learning and mathematical knowledge,** trusting himself entirely to his inventor's instinct and practical American sense.
>
> Nikola Tesla

Definition neuer Ausnahmen

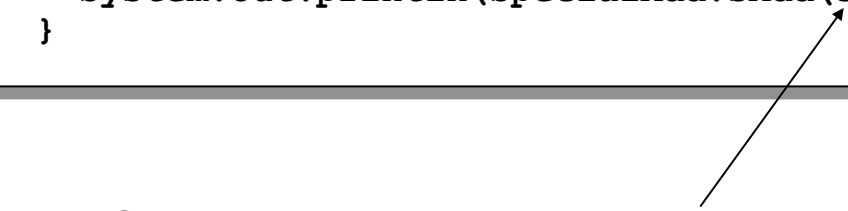
Benutzung von benutzerdefinierten Ausnahmen möglich und empfehlenswert !

```
class TestException extends Exception {
  public TestException () {
    super();
  }
}
class SpecialAdd {
  public static int sAdd (int x, int y)
                              throws TestException {
    if (y == 0)
      throw new TestException();
    else
      return x + y;
  }
}
```

Deklaration und Propagation von Ausnahmen

- Wer eine Methode aufruft, die eine Ausnahme auslösen kann, muß
  - entweder die Ausnahme abfangen
  - oder die Ausnahme weitergeben (***propagieren***)
- Propagation in Java: Deklarationspflicht mittels **throws**
  (außer bei Error und RunTimeException)

```
public static void main (String[] argv){
  System.out.println(SpecialAdd.sAdd(3,0));
}
```



Java-Compiler: Exception TestException must be caught,
or it must be declared in the throws clause of this method.

Bruch von Verträgen und Ausnahmen

- Man kann Verträge auch mit Ausnahmetests prüfen
- Vorteil: kontrollierte Reaktion auf Vertragsbrüche

```
class ContractViolation extends Exception {..};
class ParameterContractViolation extends ContractViolation
{..};
class FigureEditor{
  draw (Figure figure) throws ContractViolation {
    if (figure == null)
        throw new ParameterContractViolation();
  }
}
```

- im Aufrufer:

```
try {
   editor.draw(fig);
} catch (ParameterContractViolation) {
   fig = new Figure();
   editor.draw(fig);
}
```