**Programmieren in Java**

http://proglang.informatik.uni-freiburg.de/teaching/java/2011/

# Java-Übung Blatt 2 (Einfache Klassen)

2011-05-09

## Hinweise

- Schreiben Sie Identifier *genau so*, wie sie auf dem Blatt stehen (inklusive Groß- und Kleinschreibung), nicht nur ungefähr.
- Exportieren Sie das Projekt als ZIP-Datei mit Namen *accountname-projectname*.zip. Beispiel: Peter Lustig (lustigp) gibt ex1b ab als Datei lustigp-ex1b.zip
- ZIP-Export in Eclipse: Rechtsklick auf die Wurzel des Projects, "Export...", "General/Archive File..."
- Mailen Sie die ZIP-Files an Ihren Tutor.

Abgabe: Montag, 16. Mai 2011, um 9.00 Uhr.

**Aufgabe 1** (Datum&Test, 4 Punkte)
Projekt: ex02_1. Package: date.
Laden Sie sich das Projekt ex02_1.zip von der Vorlesungswebsite herunter. Importieren Sie dieses Projekt nach Eclipse. Eine Anleitung dazu finden Sie ebenfalls auf der Vorlesungswebsite. In der Klasse Date verstecken sich in zwei Methoden Fehler, die es zu finden gilt.
Tipp: Die Fehler haben mit Schaltjahren und der korrekten zeitlichen Ordnung von Daten zu tun.

Schreiben Sie für alle Methoden geeinete JUnit-Tests, die die korrekte Implementierung der Methoden in der Klasse Date gegenüber dem mittels Javadoc-Kommentaren spezifiziertem Verhalten testen. Korrigieren Sie die Fehler in der Implementierung der Klasse Date, so dass sich die Methoden danach entsprechend der Javadoc-Kommentare verhalten. Überprüfen Sie das Verhalten der geänderten Date-Klasse mit den von Ihnen geschriebenen Unit-Tests.

**Aufgabe 2** (Telefontarife, 4 Punkte)
Projekt: ex02_2. Package: phone.
Zu Telefontarifen und Nutzerprofilen sei folgendes bekannt:

> Jeder Telefontarif hat einen Namen. Ein Telefontarif besteht aus einer monatlichen Grundgebühr, einem Minutenpreis und einem Preis pro SMS. In der Grundgebühr können Bonusminuten enthalten sein (0 oder mehr, aber nur ganze Minuten). Nutzerprofile werden beschrieben durch einen Namen (z.B. "Vieltelefonierer"), eine Anzahl monatlich telefonierter Minuten (ganzzahlig) und eine Anzahl monatlich verschickter SMS.

- Extrahieren Sie, wie aus der Vorlesung bekannt, Klassen aus dieser Beschreibung.
- Ihre Lösung soll für beliebige Nutzerprofile und beliebige Telefontarife berechnen können, wieviel der Nutzer im Monat zahlen muss. Diese Funktionalität soll in einer Methode calculateMonthlyPrice realisiert werden. Schreiben Sie diese Methode in diejenige Klasse, die am ehesten verantwortlich ist.
- Schreiben Sie mindestens vier Testfälle. Decken Sie dabei auch die Randfälle bei Bonusminuten ab.
- Achtung: speichern Sie Geldbeträge nie als Gleitkommazahlen (double oder float)! Gleitkommazahlen verursachen Rundungsfehler, und Rundungsfehler verursachen lange Nächte mit dem Kassenprüfer. Arbeiten Sie lieber mit ganzzahligen Centbeträgen.

- Die Telefongesellschaft ExampleCom hat zwei Tarife:

| Name | Grundgebühr | Bonusminuten | Minutenpreis | SMS-Preis |
|---|---|---|---|---|
| Smile | 20 EUR | 200 | 25 ct | 10 ct |
| Happy | 0 EUR | 0 | 39 ct | 19 ct |

  Schreiben Sie eine Klasse `ExampleComPrice`, deren `main`-Methode dem Anwender berechnet, was er jeweils in den beiden ExampleCom-Tarifen bezahlen müsste. Er soll als Kommandozeilenargumente (in dieser Reihenfolge) Nutzerprofilname, monatliche Minuten, monatliche SMS angeben und die Antwort dann auf der Konsole erfahren.

**Aufgabe 3** (Fahrrad, 4 Punkte)
Projekt: `ex02_3`. Package: `fahrrad`.
Wir modellieren den Antriebsaspekt von Fahrrädern mit Hilfe von zusammengesetzten Klassen, die klare Verantwortungen haben.

(a) Ein Schaltwerk (einer Kettenschaltung) hat einige ($\geq 1$) Ritzel und weiß, auf welchem Ritzel gerade die Kette liegt. Die Ritzel in einem Schaltwerk sind in einer Reihe von innen (zur Achsmitte hin) nach außen (zum Rand hin) angeordnet. Die Anzahl der Zähne der Ritzel wird bei der Herstellung in der Reihenfolge von innen nach außen angegeben.

Man kann das Schaltwerk auffordern, die Kette um ein Ritzel nach außen oder nach innen zu verschieben; wenn die Kette da schon am entsprechenden Ende ist, bleibt sie dort. Zu Beginn ist die Kette auf dem innersten Ritzel.

Vervollständigen Sie die folgende Schaltwerk-Klasse und testen Sie sie.

```
1  public class Schaltwerk {
2      ...
3      public Schaltwerk(int[] ritzelZaehne) {...}
4      /** @return Anzahl Zaehne des aktuellen Ritzels */
5      public int aktuelleZaehne() {...}
6      public void nachInnen() {...}
7      public void nachAussen() {...}
8  }
```

Tipps für den Test:

- Ein `int`-Array mit festen Werten können Sie durch Array-Literal-Ausdrücke wie `new int[]{17,21,28}` herstellen.

(b) Eine Gangschaltung besteht aus einem vorderen und einem hinteren Schaltwerk. Das vordere Schaltwerk gibt grob den Gang vor, das hintere fein. Beim vorderen Schaltwerk ist das kleinste Ritzel innen, beim hinteren außen (nur für solche Schaltwerke ist das Verhalten der Gangschaltung spezifiziert).

Man kann die Gangschaltung anweisen, in groben oder feinen Schritten rauf oder runter zu schalten ("rauf" bedeutet jeweils in Richtung des größeren Übersetzungsverhältnisses). Das soll intern auf geeignete Weise in eine Außen- oder Innenbewegungen des entsprechenden Schaltwerks umgesetzt werden.

Man kann die Gangschaltung fragen, welche Übersetzung sie gerade realisiert (also das Verhältnis $\frac{\text{Drehzahl Radwelle}}{\text{Tretdrehzahl}} = \frac{\text{Zähne vorn}}{\text{Zähne hinten}}$).

Vervollständigen Sie die folgende Gangschaltung-Klasse und testen Sie sie (unter der Annahme, dass `Schaltwerk` korrekt ist).

```
1  public class Gangschaltung {
2      ...
3      public Gangschaltung(Schaltwerk vorderesSW, Schaltwerk hinteresSW) {...}
4      void grobRunter() {...}
5      void grobRauf() {...}
```

```
    void feinRunter() {...}
    void feinRauf() {...}
    double uebersetzung() {...}
}
```

Tipps:

- Beachten Sie bei der Division von `int`-Ausdrücken:

```
int distance = ..., speed = ...;
int time1 = distance / speed; // Ganzzahldivision
double time2 = distance / speed; // immer noch Ganzzahldivision!
double time3 = ((double)distance) / speed; // Gleitkommadivision!
```

  Wenn Sie in Java zwei `int`-Ausdrücke dividieren, bekommen Sie einen gerundeten `int`-Wert zurück. Um einen `double`-Wert zu erhalten, können Sie Dividend oder Divisor mit `((double)derAusdruck)` in einen `double`-Wert umwandeln, dann wird in `double` dividiert.

- `double`- und `float`-Werte sind nur Näherungen. Wenn Sie in Tests ein `double`-Ergebnis prüfen wollen, verwenden Sie Toleranzangaben wie hier:

```
double res = mySquareRoot(0.25);
//FALSCH: assertEquals(0.5, res); // scheitert an Rundungsfehlern
assertEquals( 0.5, res,
      0.000001 ); // <-- akzeptiert Rundungsfehler bis 0.000001
```

(c) Ein Fahrrad hat einen Hinterradumfang (in $m$) und eine Gangschaltung.

Das Fahrrad hat am Lenker vier Tasten für die Gangschaltung: links und rechts je eine zum Rauf- und Runterschalten. Mit der linken Hand bedient man den groben Teil der Gangschaltung und mit der rechten den feinen. Das Fahrrad macht keine Annahmen, ob die Gangschaltung als Kettenschaltung oder wie sonst funktioniert; es weiß nur, dass die Gangschaltung grob und fein einstellbar ist und eine Übersetzung realisiert.

Das Fahrrad hat eine momentane Tretdrehzahl (Umdrehungen pro Sekunde der Tretwelle), die man verändern kann (anfangs 0). Man kann das Fahrrad fragen, wie schnell es momentan fährt (in $\frac{m}{s}$).

Vervollständigen Sie die folgende Fahrrad-Klasse und testen Sie sie unter der Annahme, dass `Gangschaltung` korrekt ist.

```
public class Fahrrad {
  ...
  public Fahrrad(Gangschaltung gs, double radUmfangMeter) {...}
  void linksRauf() {...}
  void linksRunter() {...}
  void rechtsRauf() {...}
  void rechtsRunter() {...}
  void tretdrehzahlAendern(double tretUmdrProSec) {...}
  double geschwindigkeit() {...}
}
```