Programmieren in Java

`http://proglang.informatik.uni-freiburg.de/teaching/java/2013/`

Java-Übung Blatt 2 (Einfache Klassen & Tests)

2013-04-23

Hinweise

- Schreiben Sie Identifier *genau so*, wie sie auf dem Blatt stehen (inklusive Groß- und Kleinschreibung), nicht nur ungefähr.
- Identifier und Kommentare bitte auf *englisch*!
- Schreiben Sie *sinnvolle* Kommentare
- Laden Sie Ihre Lösungen mit subversion (svn) ins Übungssystem hoch. Den entsprechenden Pfad finden Sie online.
- Das Übungssystem kann überprüfen, ob Sie Ihr Quelltext den Anforderungen genügt und ob Sie alle Klassen erstellt haben, etc. Nutzen Sie dies!
- Sollte das Übungssystem Ihre Lösungen ablehnen, dann werden sie *nicht* korrigiert! Akzeptanzkriterien:
  - Compiliert erfolgreich
  - Checkstyle bringt keine Fehler
  - Alle Packages, Klassen, Interfaces, Methoden, Typen, Argumente sind exakt wie auf dem Übungsblatt gefordert.
- Ihr korrigierender Tutor wird die Korrektur Ihrer Abgabe in Ihr svn-Repository unter dem Namen `Feedback-<login>-ex<XX>.txt` comitten. (`<login>` ist dabei Ihr myAccount name und `<XX>` die Kennziffern des Übungsblatts)
- Zusätzlich können Sie Ihre Gesamtpunktzahl im Übungsportal einsehen.

Abgabe: Freitag, 3. Mai 2013, um 23:59 Uhr.

Dieses Blatt ist das erste, welches Sie über das Daphne Übungsportal abgegen müssen. Eine Anleitung zur Abgabe und Validierung Ihrer Lösungen finden sich auf der Tools Webseite.

`http://proglang.informatik.uni-freiburg.de/teaching/java/2013/tools.html`

Aufgabe 1 (Validierung, 0 Punkte)
Projekt: `ex02_1`. Package: `numops`.
In dieser kurzen, nicht bewerteten Aufgabe haben Sie die Chance, sich mit den Fehlermeldungen des Jenkins Build-Servers vertraut zu machen. Ähnlich wie in Blatt `ex00` ist die folgende Aufgabe im Template schon beinahe gelöst; nur die formalen Vorgaben wurden nicht erfüllt und das Build-Script des Servers akzeptiert die Abgabe nicht. Sorgen Sie dafür, das die Abgabe akzeptiert wird.

Nun zur Aufgabe: Implementieren Sie folgende Klasse und Methoden:

```
1  package numops;
2  /**
3   * Some operations on integer arrays
4   */
5  class Operations {
6      /**
7       * The average of an array of integers
8       * @param is The array to average
9       */
10     public int average(int[] is) {
11         /* ... */
12     }
13     /**
14      * The sum of an array of integers
15      * @param is The array to sum up
16      */
17     public int sum(int[] is) {
```

```
18          /* ... */
19      }
20  }
```

Aufgabe 2 (Vektoren & Test, 11 Punkte)
Projekt: `ex02_2`. Package: `vector3`.
In der Klasse `Vector3` verstecken sich in mehreren Methoden Fehler, die es zu finden gilt.

Schreiben Sie für alle Methoden geeignete JUnit-Tests, die die korrekte Implementierung der Methoden in der Klasse `Vector3` gegenüber dem mittels Javadoc-Kommentaren spezifiziertem Verhalten testen. Korrigieren Sie die Fehler in der Implementierung der Klasse `Vector3`, so dass sich die Methoden danach entsprechend der Javadoc-Kommentare verhalten. **Die Struktur der Klasse, d.h. Methodennamen, Feldnamen, usw. dürfen nicht verändert werden und es dürfen keine neuen Methoden und Felder hinzugefügt werden.** Überprüfen Sie das Verhalten der geänderten `Vector3`-Klasse mit den von Ihnen geschriebenen Unit-Tests. Stellen Sie sicher, dass Sie alle Methoden aus `Vektor3` getestet haben.

Zum Datei-Layout: Im Skelett der Übungsaufgabe finden Sie parallel zum `src` Verzeichnis ein `test` Verzeichnis. Wenn Sie eine Klasse C testen wollen, dann legen Sie dazu eine Klasse C`Test` an und schreiben die Tests für C dort hinein. Die Klasse C`Test` soll sich im Verzeichnis `test` befinden, genau da wo sich C relativ zu `src` befindet; also in einem Package mit dem gleichen Namen. Zur Illustration:

```
+ src/
  + package1/
    | Class1.java <-- enthält Code von Class1
    | Class2.java
  + package2/
    | Class3.java
+ test/
  + package1/
    | Class1Test.java <-- enthält Test für Class1
    | Class2Test.java
  + package2/
    | Class3Test.java
```

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

```
1   double res = mySquareRoot(0.25);
2   //FALSCH: assertEquals(0.5, res); // scheitert an Rundungsfehlern
3   assertEquals( 0.5, res,
4   0.000001 ); // <-- akzeptiert Rundungsfehler bis 0.000001
```

Aufgabe 3 (Carsharing, 11.5 Punkte)
Projekt: `ex02_3`. Package: `carsharing`.

Sie möchten sich als Student beim Carsharing anmelden, wissen aber noch nicht so genau für welchen der vielen Anbieter sie sich entscheiden möchten.

Zu Carsharingangeboten und Fahrprofilen sei folgendes bekannt:

> Jedes Carsharingangebot (`Plan`) hat einen Namen (`name`). Ein Carsharingangebot besteht aus einer monatlichen Grundgebühr (`baseFee`), dem Preis pro gefahrene Kilometer (`pricePerKilometer`) und eine Gebühr pro Nutzung (`feePerUse`). In der Grundgebühr können Inklusivkilometer (`kilometerIncluded`) und eine bestimmte Anzahl freie Nutzungen (`usesIncluded`) enthalten sein.
>
> Fahrprofile (`Profile`) werden beschrieben durch einen Namen (`name`, z.B. "Vielfahrer", "Urlaubsfahrer"), eine monatlich zurückgelegte Strecke (`distance`, km, ganzzahlig) und eine Anzahl monatlich Nutzungen (`uses`).

- Extrahieren Sie, wie aus der Vorlesung bekannt, Klassen aus dieser Beschreibung.
- Ihre Lösung soll für beliebige Fahrprofile und beliebige Carsharingangebote 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.
- Überprüfen Sie die Korrektheit Ihrer Implementierung durch Testfälle. Decken Sie dabei auch die Randfälle bei Inklusivkilometer und freie Nutzungen 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 Carsharinggesellschaft SuperCar hat zwei Tarife:

Name	Grundgebühr	Preis/km	Preis/Nutzung	Inklusivkm	Freie Nutzungen
ShortRunner	10 EUR	10 ct	150 ct	0km	0
LongRunner	25 EUR	5 ct	120 ct	50km	5

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

Aufgabe 4 (Fahrrad, 11 Punkte)
Projekt: `ex02_4`. Package: `fahrrad`.
Wir modellieren den Antriebsaspekt von Fahrrädern mit Hilfe von zusammengesetzten Klassen, die klare Verantwortungen haben.

(a) Ein Schaltwerk einer Kettenschaltung (engl. derailleur) hat einige (≥ 1) Ritzel (engl. sprocket) 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 (engl. teeth) 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 (upwards) oder nach innen (downwards) 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  package bike;
2  /**
3   * A derailleur consists of a number of sprockets. Each of them has a fixed number
4   * of teeth. The current sprocket is stored and can be changed.
5   *
6   * @author You
7   */
8  class Derailleur {Derailleur(int[] numberOfTeethPerSprocket) {
9          /* ... */
10     }
11     public int currentNoOfTeeth() {
12         /* ... */
13     }
14     public void downwards() {
15         /* ... */
16     }
17     public void upwards() {
```

```
18          /* ... */
19      }
20  }
```

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 (engl. coarse) den Gang vor, das hintere fein (engl. fine). 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 `Derailleur` korrekt ist).

```
1  package bike;
2  /**
3   * Gear encapsulates two derailleurs and calculates the total gear ratio
4   * with respect to the number of teeth of the currently selected sprocket
5   * of the derailleurs.
6   *
7   * @author You
8   */
9  class Gear {Gear(Derailleur coarse, Derailleur fine) {
10          /* ... */
11      }
12      public double currentGearRatio() {
13          /* ... */
14      }
15      public void coarseUpwards() {
16          /* ... */
17      }
18      public void coarseDownwards() {
19          /* ... */
20      }
21      public void fineUpwards() {
22          /* ... */
23      }
24      public void fineDownwards() {
25          /* ... */
26      }
27 }
```

Tipps:

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

```
1      int distance = ..., speed = ...;
2      int time1 = distance / speed; // Ganzzahldivision
3      double time2 = distance / speed; // immer noch Ganzzahldivision!
4      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:

```
1    double res = mySquareRoot(0.25);
2    //FALSCH: assertEquals(0.5, res); // scheitert an Rundungsfehlern
3    assertEquals( 0.5, res,
4        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 vorne und hinten 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 `Gear` korrekt ist.

```
1  package bike;
2  /**
3   * A bike, consisting of a wheel and a gear drive.
4   *
5   * @author You
6   */
7  class Bike {Bike(Gear gear, double wheelDiameterInMeter) {
8          /* ... */
9      }
10     public double speed() {
11         /* ... */
12     }
13     public double circumference() {
14         /* ... */
15     }
16     public double setRotations(double rotationsPerSecond) {
17         /* ... */
18     }
19     public void leftUpwards() {
20         /* ... */
21     }
22     public void leftDownwards() {
23         /* ... */
24     }
25     public void rightUpwards() {
26         /* ... */
27     }
28     public void rightDownwards() {
29         /* ... */
30     }
31 }
```