Übungsbeispiele Programmierpraktikum

Hermann Schichl & Harald Schilly
hermann.schichl+pp@univie.ac.at & harald.schilly+pp@univie.ac.at
http://harald.schil.ly/teaching/ue-pp-14

Sommersemester 2014

Einleitung. Die in Teil 1 & 2 gegliederten Beispiele sollen in der Programmiersprache Java gelöst werden. Es darf entweder alleine oder pro Beispiel maximal zu zweit gearbeitet werden. Wichtig ist das weitgehend eigenständige Arbeiten und das tiefergehende Verständnis der einzelnen Beispiele. Generell herrscht Anwesenheitspflicht. Davon ausgenommen sind diejenigen, welche bereits vor dem Ende der Abgabefristen die restlichen Beispiele abgeben. Die Beispiele werden von den Übungsleitern und Tutoren kontrolliert. Die Zahl der Sterne ★★· ist ein Hinweis für den Schwierigkeitsgrad und ist gleichzeitig die Anzahl der maximal zu erreichenden Punkte. Die Gesamtpunktezahl ist 42. Quiz-Fragen regen das tiefere Verständnis an (sie sollen beantwortet werden, Antworten sind aber niemals falsch).
Die Ziele des Praktikums bestehen darin, sich mit dem Implementieren von Algorithmen vertraut zu machen, ein Entwicklungsumgebung kennenzulernen, Datenstrukturen zur Repräsentation von Information verwenden zu lernen, Programm-Code ausreichend verständlich zu dokumentieren und Techniken zur Verarbeitung von Daten kennenzulernen. Weitere Informationen, Hinweise und Literatur befinden sich im Appendix und auf der Webseite.
Der erste Teil behandelt Ausdrücke, Variablen, Kontrollstrukturen und Funktionen. Er muss – begleitet durch ein Abgabegespräch – bis spätestens 29. bzw. 30. April per E-Mail an den jeweiligen Übungsleiter abgeben werden. Die Abgabe besteht aus den gesammelten `*.java` Dateien, in denen jeweils Name und Matrikelnummer als Kommentar zwischen `/* ... */` am Anfang jeder Datei vermerkt sind. Bei Zweierteams muss außerdem der Name des jeweils anderen vermerkt werden.
Der zweite Teil, beschäftigt sich ausführlich mit Objektorientierung, Standardalgorithmen und zusätzlich benötigten Fähigkeiten wie Code-Dokumentation, Performance-Tests und dem Testen von Methoden und Algorithmen. Alle Beispiele sind bis spätestens 17. bzw. 18. Juni abzugeben. Am 24. bzw. 25. Juni gibt es ein Abschlussgespräch und die Benotung.
Ergänzend bzw. alternativ zu Beispielen aus Teil 1 und 2 kann können auch die schwierigeren, mit ⊕ gekennzeichneten Zusatzbeispiele am Ende des Übungsblattes, bearbeitet werden. Diese zählen jeweils 6 Punkte und setzen eventuell Programme aus dem zweiten Teil voraus. Für die Benotung zählt die Summe aller Punkte der korrekt gelösten Beispiele, egal aus welchem Teil.
Abmeldungen sind bis spätestens 31. März möglich, siehe §8(2) der “Satzung der Universität Wien”[1] – alle angemeldeten Studierenden werden ausnahmslos benotet.

[1] `http://www.univie.ac.at/satzung/studienrecht.html`

Teil 1

1. (Basics I, Codeanalyse I, ⋆) Starte die IDE, leg ein neues Projekt an, und führe dieses Programm aus:

```
int a = 21;
a /= 3;
int b = 12;
int c = a - b;
System.out.println(c);
c++;
double d = .001 * (b - c);
System.out.println(d);
```

   Gehe im Detail jede Zeile durch und kommentiere in der Zeile darüber was sie jeweils macht. Kommentare beginnen mit "`//`".

   Anschließend stell eine Vermutung auf, welchen Wert `a`, `b`, `c` und `d` am Ende des Programms jeweils haben – und überprüfe dies.

2. (Basics II, ⋆) Ein Eiskunstläufer tanzt komplizierte Figuren. Diese besteht aus folgenden Drehungen:

   - 13 mal 60° rechts,
   - 12 mal 45° links,
   - 7 mal 435° Schrauben rechts,
   - 2 mal 720° Schrauben links,
   - 4 mal 180° Bögen links und
   - 2 mal 270° Bögen rechts.

   Alles zusammengenommen, hat er sich nach links oder rechts gedreht? Wie groß ist seine Netto-Drehung (im Intervall von -360° bis +360°)?

3. (Kontrollstrukturen I, ⋆) Speichere in den Variablen `lb` und `ub` eine untere und obere positive `double`-Zahl als Grenzen. In einer weiteren `int`-Variablen `n` eine positive ganze Zahl ≥ 2. Teste explizit, dass $0 < lb < ub$ und $n \geq 2$ ist.

   Gefragt ist das Produkt von n gleich weit voneinander entfernten Zahlen, welche bei `lb` beginnen und bis einschließlich `ub` enden. Berechne dies explizit mittels einer `for`-Schleife.

   Beispiel: $lb = 1$, $ub = 5$, $n = 3$ liefert $1 * 3 * 5 = 15$.

   Quiz: Beschreibe bisherige Erfahrungen: Was hat gut funktioniert und worauf muss man Acht geben? Welche Schwierigkeiten gibt es und und was ist missverständlich?

4. (Kontrollstrukturen II, ⋆⋆) Die Volumsformel eines Kegels ist $V = \frac{1}{3} r^2 \pi h$.

   Implementiere diese mittels

   - einer Funktion mit der Signatur `static double kegel_volumen(double r, double h)`,
   - und kodiere global den Wert von π mit einer festen "Variablen" `final static double PI`.

   Berechne diese beiden Kegel:

- $r = 2.1$, $h = 22$
- $r = 3$, $h = 0.3$

Gib die Ergebnisse aus.

Quiz: Auf wie viele Stellen lässt sich *pi* angeben?

5. (Kontrollstrukturen III, ★) “Würfle” in einer `while`-Schleife mit einem “virtuellen” 6-seitigen Würfel, bis exakt 3-mal hintereinander die Augenzahl 6 geworfen wird. Wiederhole dieses “Experiment” 100-mal mittels einer äußeren `for`-Schleife und gib jeweils die Anzahl der dafür notwendigen Würfe aus.

   Wie viele Würfe sind durchschnittlich notwendig?

   Hinweis zur Modellierung eines Würfels: `Math.random()` liefert gleichverteilte Zufallszahlen im Intervall $[0, 1)$ und “Casting” via `(int)` rundet auf eine `int`-Ganzzahl ab.

   Quiz: Könnte dieser Algorithmus Probleme machen?

6. (ASCII I, ★★) Eine beliebige Funktion $f : \mathbb{R} \to \mathbb{R}$ soll in einer separaten Methode `static double f(double x)` angegeben werden. Gib eine Auswertungstabelle für diese Funktion aus. Es soll der Anfangswert, der Endwert und die Schrittweite in Variablen vorgegeben werden. Gib die Tabelle zeilenweise als formatierten `String` mit der Funktion `System.out.printf()` und der passenden Formatierungsanweisung in dem String auf die Art aus, dass die Zahlen am Komma untereinander ausgerichtet sind, zwei Nachkommastellen haben und es eine vertikale Trennlinie gibt. Beispiel:

```
         x |       f(x)
-----------+------------
      2.00 |       2.91
      4.50 |       1.21
      7.00 |       0.72
      9.50 |     -11.92
     12.00 |    -108.03
```

   Quiz: Welchen Unterschied machen “`f`”, “`g`” und “`e`” in den Formatierungsanweisungen?

7. (Codeanalyse II, ★★)

   Übertrage die beiden folgenden Programme und versuche sie auszuführen.

```java
int sum = 0;
for (int i = 0; i<=100; i++) {
    i = sum;
}
System.out.println(sum);
```

```java
static boolean is_prime(int x) {
    if (x % 2 == 0) return false;
    for(int i = 3; i >= Math.sqrt(x); i+=2) {
        if(x % i == 0) return false;
    }
    return true;
}
public static void main(String... args) {
    System.out.println(is_prime(15));
    System.out.println(is_prime(997));
    System.out.println(is_prime(2014));
}
```

Aufgabe: Dokumentiere jede relevante Zeile mittels eines Kommentars und beschreibe allgemein, was die Programme korrekter weise machen sollten. Anschließend korrigiere jeweils die Fehler.

Hinweis: Wenn ein Programm nicht von sich aus stoppt, muss es explizit beendet werden. Beschreibe, woran sich diese Fälle erkennen lassen und wie man derer handhab wird.

Quiz: Lässt sich immer im Voraus sagen, ob ein Programm endlos laufen wird?

8. (Kontrollstrukturen IV, ⋆⋆) Es soll die *Fläche* unter einer Funktion $f(x)$ in einem beliebigen Intervall $[a, b]$ mit $a < b$ berechnet werden. $f(x)$ soll in einer separaten Methode `static double f(double x)` definiert werden.

   Benütze die zusammengesetzte Trapezregel[2] und `Math.abs()` für den Absolutbetrag. Vergleiche die Güte der berechneten Annäherung für drei verschiedene Schrittweiten (z. B. 0.1, 0.01 und 0.001) mit dem tatsächlichen Wert.

   Beispiel: Die Fläche unter $f(x) = sin(x)$ im Intervall $[0, 3.14]$ ist ungefähr 2 und im Intervall $[0, 6.28]$ ungefähr 4.

   Quiz: Kann die Schrittweite beliebig klein gemacht werden?

9. (Strings, ⋆) In diesem Beispiel beschäftigen wir uns mit den Eigenheiten von Zeichenketten (Strings). Gehe zuerst am Papier theoretisch durch, welches Ergebnis zu erwarten ist. Führe dann das Programm aus und erkläre das Verhalten schriftlich!

```
1  public static void main(String... args) {
2    String a1 = "foo";
3    String a2 = a1;
4    testStrings("a1, a2", a1, a2);
5    String a3 = "foo";
6    testStrings("a1, a3", a1, a3);
7    String a4 = new String("fo") + "o";
8    testStrings("a1, a4", a1, a4);
9    String a5 = a4.intern();
10   testStrings("a1 a5", a1, a5);
11 }
12
13 static void testStrings(String info, String a, String b) {
14     System.out.println(info);
15     System.out.printf("<%s>.equals(<%s>) -> %s\n", a, b, a.equals(b));
16     System.out.printf("<%s>   ==   <%s>  -> %s\n\n", a, b, a == b);
17 }
```

10. (API, ⋆) Lerne die Java Plattform Dokumentation und insbesondere die Java-API kennen (siehe Literaturangaben). Wo ist sie zu finden, wie ist sie aufgebaut, welche Arten von Dokumentationen gibt es? Beantworte folgende Fragen:

    a) Welche Besonderheit hat das Paket `java.lang` im Unterschied zu allen anderen Paketen der API?

    b) Was ist ein “Java HotSpot Compiler” und was macht er?

    c) Später lernen wir, dass Java eine objektorientierte Sprache ist, wobei (fast) alles von einer ganz allgemeinen Klasse “Object” abgeleitet wird. Finde dieses `Object` in der API, notiere die URL und alle Methodensignaturen dieser Klasse.

    d) Suche nun die Klasse `java.lang.Math` und erkläre, was die Methoden `hypot` und `toDegrees` machen.

    e) Finde die Klasse `HashSet` und dessen Interface `Set`. Erkläre anhand der Beschreibungen, was sie leistet und beschreibe einen möglichen Anwendungsfall.

[2] `http://de.wikipedia.org/wiki/Trapezregel`

11. (ASCII II, Analysis I, ★★★) Abermals begeistert der Eiskunstläufer die MathematikerInnen! Eine besonders schwierige Figur lässt sich mit folgender nach t parametrisierten Kurve zeichnen:

$$f(t) = \begin{pmatrix} sin(-1 + log(1 + 4 * t)) \\ cos(sqrt(4 * (1 + t))) \end{pmatrix} \qquad t \in [0,\ 10\pi]$$

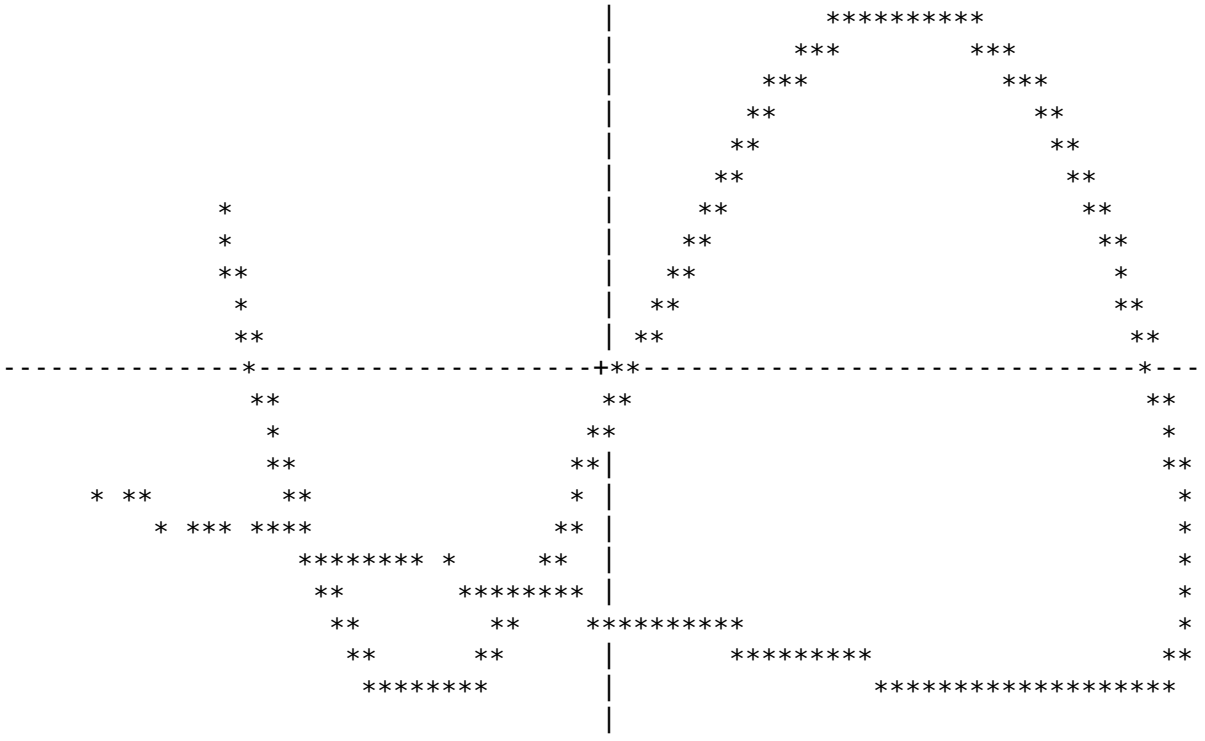
Definiere hierzu diese Funktion $f(t)$ mit der Signatur `static double[] f(double t)`.

Benütze ein 2D-Array um die Ausgabe zwischenzuspeichern. Definiere dessen Größe mittels statischer Variablen `COLS` und `ROWS`, welche die Anzahl der Spalten (ca. 80 bis 100) und Zeilen (ca. 40 bis 60), die zur Darstellung der Funktion verwendet werden sollen, angeben.

Dann werte die Funktion an mindestens 1000 Stellen im angegebenen Intervall aus und "zeichne" zeilenweise das Array. *Quiz*: Warum so viele Funktionsauswertungen?

Dabei soll alles so skaliert werden, dass der dargestellte Bereich für x und y jeweils von $[-1,\ 1]$ geht und der Ursprung in der Mitte liegt. Vergiss nicht auf das Koordinatenkreuz und die korrekte Orientierung der x und y Achsen!

Die folgende Abbildung zeigt das ungefähre Ergebnis an. Detailverbesserungen sind natürlich erwünscht :-)



Teil 2

12. (Rekursion I, Benchmark, ★★)

    Die "Schröder–Hipparchus"-Zahlen[3] lauten 1, 1, 3, 11, 45, 197, 903, 4279, 20793, ... und kommen in Zahlentheorie und Kombinatorik vor. Sie geben z. B. an, auf wie viele Arten ein konvexes Polygon mittels Diagonalschnitten zerlegt werden kann.

    Aufgabe dieses und des folgenden Beispiels ist, die dazugehörige rekursive Relation auf verschiedene Arten zu programmieren. Die Formel lautet:

$$S_n = \frac{(6n-3)\cdot S_{n-1} - (n-2)\cdot S_{n-2}}{n+1} \qquad \text{mit} \qquad S_0 = S_1 = 1$$

    Es sollen die Folgenglieder dieser Rekursion sowohl *iterativ*[4] als auch *rekursiv*[5] berechnet werden. Berechne die ersten ca. 20 Zahlen, benütze durchwegs den Datentyp `Long` und überlege, welches Verhalten diese Zahlenfolge zeigt!

    Vergleiche in einem *Benchmark*, ob die rekursive oder die iterativen Methode schneller ist – und um wie viel. (Um auf aussagekräftige Zahlen zu kommen, sollte dieselbe Berechnung genügend oft wiederholt werden; siehe Glossar "Benchmark").

    Quiz: Der verwendete Datentyp `Long` hat Grenzen. Wann sind die erreicht? Was passiert wenn sie überschritten werden und kann der Datentyp `BigInteger` helfen?

13. (Rekursion II, Benchmark, ★★) Verbessere die rekursive Berechnung der Folgenglieder des vorhergehenden Beispiel derart[6], dass Zwischenergebnisse für S_i in einem Array an der Position i gespeichert werden. Ist das Zwischenergebnis bereits bekannt, gib es zurück – wenn nicht, führe die Berechnung aus und speichere das Ergebnis. Dadurch werden unnötige Rekursionen eingespart. Mache erneut ein aussagekräftiges Benchmark (siehe Glossar).

    Hinweis: Verwende `null` als Marker für "unbekannte" Werte in einem Array des Typs `Long`.

    Quiz: Wieso ist z. B. "0" oder "1" kein geeigneter Marker?

14. (Klassen I, Matrix I, ★★★) Schreibe eine Klasse `Matrix`, welche die wichtigsten Matrix-Operationen implementiert. Die Klasse soll allgemeine $m \times n$ Matrizen mit `Double` Einträgen als Elemente ermöglichen. Die Indizes laufen dabei jeweils von $0 \ldots m-1$ und $0 \ldots n-1$. Ein Beispielprogramm soll jede Methode aufrufen und die Ergebnisse in der Konsole ausgeben.

    Folgende Methoden soll die Klasse `Matrix` beherrschen:

```
1  // Konstruktor für eine n x n Matrix
2  public Matrix(int n)
3  // Konstruktor für eine rows x cols Matrix
4  public Matrix(int rows, int cols)
5  // Zufallsmatrix, n x n Matrix mit zufälligen Einträgen
6  public static Matrix random(int n)
7  // Größe der Matrix, [rows, cols]
8  public int[] size()
9  // gib das Element an der Position (r,c) zurück
```

[3] http://en.wikipedia.org/wiki/SchrÃűder-Hipparchus_number
[4] d.h. mit einer `for`-Schleife und mehreren Variablen
[5] d.h. mittels Aufruf der `f(n)` Funktion innerhalb ihrer selbst!
[6] Fachbegriff: "dynamisches Programmieren"

```
10 public Double get(int r, int c)
11 // setze Element (r,c) auf den Wert v
12 public void set(int r, int c, Double v)
13 // addiere elementweise eine Matrix a (neues Objekt zurückgeben!)
14 public Matrix add(Matrix a)
15 // multipliziere mit einer Matrix a (neues Objekt zurückgeben!)
16 public Matrix mult(Matrix a)
17 // berechne die Summe einer Zeile
18 public Double rowsum(int r)
19 // berechne die Summe einer Spalte
20 public Double colsum(int c)
21 // multipliziere die Matrix mit dem Skalar a  (neues Objekt zurückgeben!)
22 public Matrix mult(double a)
23 // "wahr", wenn quadratisch
24 public boolean isSquare()
25 // String-Darstellung
26 public String toString()
```

Bemerkung: Um Missverständnissen vorzubeugen, die Matrix-Matrix Multiplikation für quadratische Matrizen A und B selber Größe n für unsere Zwecke so definiert:

$$(A \cdot B)_{j,k} = \sum_{i=0}^{n-1} A_{j,i} \cdot B_{i,k} \quad \text{für} \quad (j, k) \in \{(i_1, i_2) \mid 0 \leq i_1 \leq n-1;\ 0 \leq i_2 \leq n-1\}$$

wobei der Index links die Position des jeweiligen Elements an der Stelle j, k der Ergebnismatrix angibt.

Quiz: Wieso ist es wichtig, in den angegebenen Fällen neue Objekte zurückzugeben?

15. (Collections, ★★) Üblicherweise wird in Programmen nicht nur ein einzelnes, isoliertes Objekt benötigt, sondern eine ganze "Sammlung" von ähnlichen Objekten bzw. werden Assoziationen zwischen ihnen hergestellt.

    Erstelle ein kurzes Programm um diese Techniken kennenzulernen. Es soll folgende Aufgaben bewerkstelligen:

    - Speichere 100 zufällig gewählte Zahlen im Bereich 0 bis 100 in
      - einem `Integer[]`–Array;
      - in einer `ArrayList<Integer>` [7];
      - in einer Menge, also der Klasse `HashSet<Integer>`;
      - in einem `ArrayDeque<Integer>`.

      Gib für *jeden* dieser Typen alle Methoden an, durch die Elemente *hinzugefügt* oder *entfernt* werden können.
    - Anschließend gib jeweils folgende Informationen aus:
      - Wie viele Elemente befinden sich jeweils in den Containern? Fällt etwas auf?
      - Gib alle "Sammel"-Objekte mittels `System.out.println()` aus. Tipp: Für die korrekte Ausgabe des Arrays ist die Hilfsfunktion `Arrays.toString(Variable)` hilfreich. Was fällt auf?
      - Berechne jeweils die Summe aller Elemente.

    Weiterführende Informationen hier[8] und in der API hier[9].

[7] Dabei gibt der Klassenname in den spitzen Klammern an, welchen Typ diese `ArrayList` beinhaltet; Stichwort: "*Generics*"

[8] `http://download.oracle.com/javase/tutorial/collections/TOC.html`

[9] `http://download.oracle.com/javase/7/docs/technotes/guides/collections/index.html`

16. (Klassen II, ★★) In diesem Beispiel geht es um Klassen und Vererbungen. Es sollen folgende Klassen (konkret oder abstrakt) mit den jeweils angegeben Konstruktoren, Methoden und Feldern implementiert werden. Das Interface `ITrainInfo` stellt dabei sicher, dass alle Klassen untereinander kompatible Methoden haben!

    `interface ITrainInfo`:

    - `Double getLength()` – Länge
    - `Double getMass()` – Masse
    - `Integer getPassengerNumber()` – Anzahl Passagiere
    - `Double getMaximumSpeed()` – Maximalgeschwindigkeit

    `class Waggon implements ITrainInfo`:

    - Felder:
        - `private String id` – eindeutige ID
        - `private Double length` – Länge
        - `private Double mass` – Masse
        - `private Integer nbPassengers` – Anzahl Passagiere
        - `private Double maxSpeed` – Maximalgeschwindigkeit
    - Methoden:
        - `public Waggon(String id)` – Konstruktor für einen Wagen mit einer eindeutigen ID
        - die Methoden des Interfaces
        - `setPassengerNumber(Integer)` – setze den Wert für die Anzahl der Passagiere
        - `public String toString()` – generiert eine für den Menschen lesbare Ausgabe (z. B. “Wagon 5142”)[10].

    Instanziiere mindestens fünf Objekte dieser `Waggon` Klasse, z. B.:

    - `travelWaggon` – Reisewagen: eher leicht, große Anzahl Personen, ...
    - `freightWaggon` – Güterwagen: schwer, keine Personen, mittlere Geschwindigkeit
    - `diningWaggon` – Speisewagen: leicht, wenige Personen.

    Jetzt benötigen wir noch eine Lok, welche sich von `Waggon` ableitet:

    `class Locomotive extends Waggon` mit den zusätzlichen Methoden **`start`** und **`stop`** sowie einer passenden Zustandsvariablen **`Boolean isDriving`**.

    Schreibe eine weitere Klasse **`Run`**, welche die Objekte instanziiert und alle Methoden durchprobiert.

    Quiz: Können interfaces auch Felder beinhalten?

17. (Klassen III, ★★)

    Nun bau dies nach belieben zu einem vollständigen Zug zusammen. Dabei sollen Methoden wie `getMass()` und `getLenght()` die Werte automatisch anhand der gegebenen Waggone berechnen.

    `class Train implements ITrainInfo`:

    - Felder:
        - `private String id` – eindeutige ID

[10] welche dann z. B. in `System.out.println(...)` automatisch für Instanzen dieser Klasse aufgerufen wird

- `private Locomotive locomotive` – die Lok
- `private Deque<Waggon> waggons = new LinkedList<>()` – angehängten Waggone[11]

- Methoden:
  - `public Train(String id, Locomotive locomotive)` – Konstruktor für einen Wagen mit einer eindeutigen ID und Lok.
  - `public void addWagon(Wagon w)` – kuppelt den Wagon `w` mittels `push(w)` an den Zug an.
  - `public Wagon removeWagon()` – kuppelt den letzten Wagon mittels `pop()` ab und gibt ihn zurück.
  - die Methoden des Interfaces
  - sowie `start()` und `stop()` – rufen die entsprechenden Methoden der Lok auf.
  - `public String toString()` – generiert eine für den Menschen lesbare Ausgabe (z. B. “Zug E414”).

Instanziiere mindestens zwei unterschiedliche Züge, gib Länge, Masse, Maximalgeschwindigkeit und Gesamtzahl der Passagiere aus, und lass sie für ein paar Sekunden fahren.

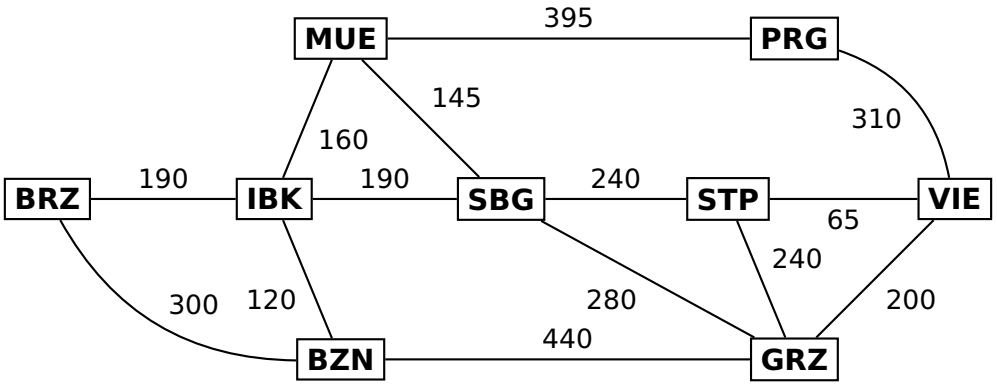
```
1 for (int t = 0; t < 5; t++) {
2   Thread.sleep(1000);
3   System.out.println("Fahrzeit: " + t + " [s].");
4 }
```

Dann stoppe die Züge und kopple jeweils einen Wagon ab.

Quiz: Wie müsste man das Rangieren von einem Waggon in der Mitte eines Zuges zu einem anderen Zug programmieren?

18. (Klassen IV, Graph I, ★★★)

Im vorhergehenden Beispielen sind ganze Züge konstruiert worden, jetzt dreht sich alles um das dazugehörige Schinennetz. Der Graph in der Abbildung zeigt, wie ein solches aussehen könnte.



Implementiere folgende Klassen und instanziiere dann das angegebene Netz.

`class Stop`: eine Haltestelle

- Felder:
  - `private String name` – eindeutiger Name

[11] ein sogenannter LIFO-(“Last-In-First-Out”)-Stack

- Methoden:
  - `public Stop(String name)` – Konstruktor mit einem eindeutigen Namen.
  - `public String toString()` – generiert eine für den Menschen lesbare Ausgabe (z. B. “Haltestelle VIE”).

`class Link`: Verbindung von einer Haltestelle zu einer anderen

- Felder:
  - `private Stop origin` – die eine Haltestelle
  - `private Stop destination` – die andere Haltestelle
  - `private Double distance` – Distanz zwischen den Haltestellen
- Methoden:
  - `public Link(Stop origin, Stop destination, Double distance)` – Konstruktor
  - `public String toString()` – generiert eine für den Menschen lesbare Ausgabe (z. B. “Entfernung von VIE nach PRG ist 195km”).

`class RailNetwork`: das gesamte Schienennetzwerk

- Felder:
  - `private Set<Link> links` – Menge aller Verbindungen
- Methoden:
  - `public void addLink(Link link)` – fügt Verbindung hinzu
  - `public boolean isConnected(Stop[] path)` – Gibt `True` zurück, wenn der angegebene Pfad zusammenhängend ist. d. h. Iteriere über alle aufeinander folgenden Paare von `Stop`-Objekten in der Liste und stell sicher, dass alle Teilstücke existieren und abgefahren werden können[12].
  - `public Double totalLength(Stop[] path)` – Gesamtlänge des Pfades.
  - `public String toString()` – generiert eine für den Menschen lesbare Ausgabe

Aufgabe:

- Beschreibe Fahrstrecken (mindestens 2) durch ein Array von `Stop` Objekten. Überprüfe, ob diese Fahrstrecken jeweils zusammenhängend sind (mindestens eine soll es sein, mindestens eine nicht).
- Berechne die gesamte Länge der jeweiligen Fahrstrecken.
- Berechne mittels gesamter Fahrstrecke, der Anzahl Zwischenstopps (feste Standzeit pro Aufenthalt plus geschätzter Zeit für Verzögerung & Beschleunigung) und der maximalen Geschwindigkeit, wie lange ein Zug unterwegs sein wird.

Quiz: Ist ein mehrmaliges hinzufügen der selben Links ein Problem? Unter welchen Umständen könnte es das werden?

19. (Tests, ⋆⋆) Schreibe für die Klassen `Waggon`, `Train`, `Stop`, `Link` und `RailNetwork` aus dem vorhergehenden Beispielen *Tests*. Das heißt, eine weitere Klassen `TrainTest` und `RailwayTest`, welche zuerst einige fix vorgegebene Klassen instanzieren, dann alle Methoden für jede Berechnung aufruft und jeweils explizit überprüft, ob jedes Ergebnis korrekt ist. D. h. jedes berechnete Ergebnis jeweils mit einem explizit eingegebenen Wert verglichen wird. *Alle* Methoden sollen getestet werden!

[12] Ein `Link` lässt sich in beide Richtungen abfahren.

Hinweis: Verwende das Schlüsselwort `assert`[13] und füge zu den Klassen jeweils eine Methode `public boolean equals(Object other)` hinzu, welche auf Äquivalenz testet. Das folgende Programmlisting zeigt exemplarisch, wie so ein `equals`-Test korrekt implementiert wird. `ExpectedClassName` ist durch den zu erwartenden Klassennamen zu ersetzen, um problemlos casten zu können. Weiters muss der Hashwert der Objekte immer dann übereinstimmen, wenn dieser Vergleichstest wahr ist.

```
1  @Override
2  public boolean equals(Object other) {
3      if (this == other) {
4          return true;
5      }
6      if (other instanceof ExpectedClassName){
7          ExpectedClassName o = (ExpectedClassName) other;
8          boolen isSame = member1.equals(o.member1);
9          isSame &= member2.equals(o.member2);
10         [...]
11         return isSame;
12     }
13     return false;
14 }
15
16 @Override
17 public int hashCode() {
18     int h1 = Double.valueOf(member1).hashCode();
19     int h2 = member2.hashCode();
20     [int h3 = ...]
21     return h1 + 31*h2 + [31*31*h3 +] ...;
22 }
```

Quiz: Welche Grenzen haben solche Tests?

20. (Javadoc, Sprachelemente, Code Conventions, ★★) Formatiere den Code aller Klassen entsprechend der "Java Code Conventions" (siehe Literaturangaben). Insbesondere korrigiere Zeilenumbrüche, Einrückungen und beachte die korrekte Groß- und Kleinschreibung und identifiziere die unterschiedlichen Sprachelemente.

    Lies die notwendigen Kapitel der Dokumentation für "javadoc" durch. Formuliere passende Beschreibungen für die Klassen selbst, für *alle* Methoden (und Parameter) und *alle* Felder, trage den eigenen Namen als Autor in den Headern ein und mach einen erklärenden Querverweis (Schlüsselwort "`@link`") innerhalb des Beschreibungstextes der Methode `Train.getLength()` auf die Methode `Waggon.getLength()`.

    Generiere anschließend die "javadoc"-Dokumentation als Sammlung von HTML Dateien mit dem Programm *javadoc* bzw. mittels der IDE.

    Quiz: Warum ist eine einheitliche Formatierung und genaue Dokumentation so wichtig?

21. (Statistik I, ★★) Die Aufgabe dieses Beispiels besteht darin, ein Analyseprogramm für Texte zu schreiben. Eine gute Quelle für Texte ist das Projekt Gutenberg (`http://www.gutenberg.org/catalog/`). Lade reine Textdateien herunter, welche von dem Programm nach folgenden Kriterien analysiert werden sollen:

    - Anzahl der Wörter und Buchstaben;
    - Länge (minimal, durchschnittlich, maximal) der Wörter;
    - die 10 häufigsten Wörter;
    - die 10 häufigsten Buchstabenpaare (ignoriere Groß-/Kleinschreibung) und deren jeweilige Anzahl;

[13] oder verwende das JUnit Framework

Beachte, dass alle Sonderzeichen ignoriert werden sollen und fange etwaige Sonderfälle ab. Hinweis: Die Klasse `java.util.Scanner` wird sich als nützlich erweisen und für die Statistiken verwende die Klasse `java.util.HashMap` bzw. eine Matrix.

Quiz: Was ist eine “Regular Expresson”? (Gib Beispiele an.)

22. (Analysis II, ASCII II, ⋆⋆⋆) In diesem Beispiel geht es um die numerische Lösung von gewöhnlichen Differentialgleichungen. Implementiere das explizite Euler’sche Verfahren[14], welches mittels N kleiner Schritte der Länge δ die Lösung approximiert: $x(t+\delta t) = x(t) + \delta \dot{f}(t, x)$.

    Wir betrachten hierbei die Lotka-Volterra-Gleichung [15] aus der Mathematischen Ökologie:

$$\dot{f}(t) = \begin{cases} \dot{b} &= b(\epsilon_b - \gamma_b r) \\ \dot{r} &= -r(\epsilon_r - \gamma_r b) \end{cases}$$

    wobei die Variablen r für die Anzahl der Räuber, und b für die der Beutetiere steht. Die fixen Parameter (jeweils $\in [0, 1]$) sind: ϵ_b Reproduktionsrate der Beutetiere, γ_b Sterberate der Beute pro Räuber, ϵ_r Sterberate der Räuber (wenn keine Beute vorhanden), und γ_r ist die Reproduktionsrate der Räuber pro Beute.

    Löse $x(t)$ (also $b(t)$ und $r(t)$) numerisch und gib eine paarweise Liste der Werte aus. Nützlich wird auch ein “Plot” beider Werte sein, entweder nur als vertikaler Graph oder ein “XY”-Plot wie im letzten Beispiel des ersten Teils. Der Startwert $x(0)$ ist frei wählbar, z. B. $(10, 10)$. Falls es eine zu lange Liste ist, reduziere diese indem Durchschnitte berechnet werden.

    Aufgabe: Durch die Wahl welcher Parameter lassen sich periodische Schwankungen oder das Aussterben einer Art[16] simulieren?

    Teste, wie sehr sich die Schrittweite δ auf das Ergebnis am Ende des Durchlaufs auswirkt (beachte, dass hierbei $\delta * N$ jeweils gleich sein muss!).

[14] `http://de.wikipedia.org/wiki/Explizites_Euler-Verfahren`
[15] `http://de.wikipedia.org/wiki/Lotka-Volterra-Gleichungen`
[16] Anzahl ≤ 1

Weiterführende Beispiele

Die folgenden Beispiele sind zur tieferen Auseinandersetzung bzw. als Zusatzbeispiele gedacht.

23. (Optimierung, ⊕) Ziel dieser Aufgabe ist, einen sogenannten “Solver” zur Lösung einer mathematischen Optimierungsaufgabe zu schreiben. Es soll hierbei eine Funktion `public Double f(Double x)`, von der man keine Ableitung kennt und auch keine bestimmt werden soll, durch ein beliebiges Verfahren minimiert werden.

    Für x gibt es untere und obere Schranken, welche den Suchraum auf einen endlichen Bereich einschränken.

    Teste diesen Solver mit den folgenden Funktionen und gib das jeweils gefundene Minimum (x und $f(x)$ Wert) aus:

    $$f_0(x) = -cos(x^2) * x \qquad \text{mit} \quad 0 \leq x \leq 10$$
    $$f_1(x) = -\left|log(1+x) - (1-x^2) * (4-x)^3\right| \qquad \text{mit} \quad 0 \leq x \leq 5$$
    $$f_2(x) = \sum_{i=0}^{100} x * sin(x - 100 * k) \qquad \text{mit} \quad -7 \leq x \leq 7$$

    Aufgabe: Beschreibe die Strategie, um das Minimum zu finden und erläutere diese anhand des Codes. Welche Probleme haben sich ergeben?

24. (Modellierung, ⊕) Die Aufgabe besteht darin, Schelling’s “Segregation Model” (SSM)[17] in einer einfachen Variante zu Programmieren.

    In diesem Modell werden zwei Gruppen von Bewohnern einer Stadt in einer $n \times n$ Matrix durch zwei Symbole (z. B. X und 0) dargestellt. Es sollen etwa 80% der Felder der Matrix sollen besetzt sein. In einem iterativen Prozess wechseln die “Bewohner” an einen zufälligen neuen Platz, wenn in ihrer unmittelbaren Umgebung (8 Felder) die Anzahl an “Freunden” einen bestimmten Grenzwert unterschreitet.

    Iteriere entweder so lange, bis keine Platzwechsel mehr nötig sind, oder ein vorgegebenes Iterationslimit überschritten wird. Variiere den Grenzwert ab dem ein Bewohner sich für einen Ortswechsel entscheidet, und ebenso die Dichte von 80%.

    Die Ausgabe soll aus zwei Teilen bestehen:

    - Gib für jede Iteration die durchschnittliche Anzahl an benachbarten “Freunden” pro Gruppe an.
    - Gib das Bild der Stadt am Ende der Iterationen in der Konsole aus.

    Hintergrund dieser Aufgabe ist die Frage, wann und warum sich selbst bei nur sehr milden Bedingungen an die Nachbarschaft, großflächige homogene Bereiche bilden.

25. (Statistik II, ⊕) Ziel der folgenden Übung ist, Formeln aus der Literatur der Informatik in einem (relativ einfachen) Programm zu implementieren. Es soll die Entropie eines gegebenen Textes nach Shannon berechnet werden. Dies wird in *“Prediction and Entropy of Printed English”*, (1950) auf Seite 51 in Formel (1) erklärt:

[17] `http://sdl.soc.cornell.edu/curricular/schelling_chapter_on_segregation.pdf`

Die Entropie ist

$$F_N = -\sum_{i,j} \Pr(b_i, j) \log_2(\Pr(j|b_i))$$

wobei b_i alle n-gramme sind, (b_i, j) bedeutet, dass Buchstabe j an das n-gramm b_i angehängt wird und $\Pr(j|b_i)$ ist die bedingte Wahrscheinlichkeit, dass ein j auf b_i folgt – das ist $\Pr(b_i, j)/\Pr(b_i)$.

Gibt es Unterschiede zwischen Deutsch, Englisch, Französisch, ... ?

Tipp: Pass beim Parsen des Textes auf, nur tatsächliche Wörter und keine sonstigen Zeichen einzulesen. Verwandle alle Zeichen in Großbuchstaben und arbeite nur mit diesen! Dann generiere eine Liste b_i von *allen* n-grammen (das sind Buchstaben-Arrays der Länge n) – ein guter Wert für n ist 3, teste später auch 4 und 5 (siehe Formel (2)). Anschließend iteriere über alle n-gramme, durchsuche die Texte und ermittle die beiden Wahrscheinlichkeiten für die Formel.

Literatur: `http://languagelog.ldc.upenn.edu/myl/Shannon1950.pdf`

26. (Graph II, ⊕) Im Beispiel “Graph I” ging es um das Netzwerk einer Bahnlinie. Ein Teil der Aufgabe bestand darin, einen zusammenhängenden Pfad im Netzwerk anzugeben und dessen Länge auszurechnen. Dieses Beispiel baut darauf auf: Gesucht ist eine Prozedur, welche für ein beliebiges Netzwerk die kürzeste zusammenhängende Strecke zwischen zwei beliebig wählbaren Stationen berechnet.

27. (Rekursion III, Matrix II, ⊕) Implementiere als zusätzliche, schnellere Multiplikationsroutine der `Matrix`-Klasse für quadratische Matrizen den “Strassen-Algorithmus”[18]. Es ist weiters nötig, für kleine Submatritzen auf die herkömmliche Multiplikationsmethode zurückzugreifen (“cutoff”-Wert). Welcher “Cutoff”-Wert erweist sich als günstig, d. h. ab welcher Größe ist Strassen schneller als die “naive” Methode? Mache Benchmarks um die Implementation dieser Methode mit der naiven Methode vergleichen zu können und um sie zu “tunen”.

    Für sehr große Matrizen muss eventuell der maximal zur Verfügung gestellte Heap-Speicher vergrößert werden (`-Xmx<Zahl>m` Parameter der JVM).

    Teste für Größen zwischen 50×50 bis 1000×1000 ob die beiden Multiplikationsmethoden identische Ergebnisse liefern und gibt die Zeiten der Benchmarks aus.

[18] `http://de.wikipedia.org/wiki/Strassen-Algorithmus`

Appendix

Tipps

- Fehlermeldung `java.lang.NoClassDefFoundError` beim Starten von der Kommandozeile: Die kompilierte JAVA Klasse – oder besser gesagt das Wurzelverzeichnis des kompilierten Programms – ist nicht im durchsuchten Klassenpfad. Die Lösung ist, den `classpath` zu setzen, zum Beispiel:

  `java -cp <pfad> <KlassenName> [Argumente]`

  oder in einer Verzeichnishierarchie für Pakete das Wurzelverzeichnis:

  `java -cp <Wurzelverzeichnis> paket/hierachie/<KlassenName> [Parameter]`

  für eine Klasse im Paket `paket.hierachie.<KlassenName>` dessen Wurzelverzeichnis in `<Wurzelverzeichnis>` liegt.

- Programmiere wenn möglich so, dass
  - kein Code doppelt vorkommt;
  - alle Variablen und nicht mehr weiter abgeleitete Methoden das Schlüsselwort `final` haben;
  - alle Methoden möglichst eingeschränkten Zugriff haben, sprich, alles, worauf man im Paket Zugriff haben soll, auf “default” (d. h. keine Angaben); alles lokale auf `private` oder `protected` und nur wenige, ausgewählte Methoden und Klassen auf `public` setzt;
  - möglichst wenig neue Objekte generiert werden, vor allem nicht innerhalb von Schleifen, und
  - übergebene Parameter aus nicht vertrauenswürdigen Quellen überprüft werden.

Literatur

- Guido Krüger, Thomas Stark: “Handbuch der Java-Programmierung”, kostenloser Download bei `http://javabuch.de`. Dies ist eine öfters überarbeitete und gut durchdachte Einführung in die Java Sprache. Hervorzuheben sind die Kapitel: 1, 2.2, 2.3 (2.3.3), 4, 5, 6, 11, 11.4, 13.2, 15 für die Grundlagen, 7, 8, 9 für Objektorientierte Programmierung (OOP), des weiteren 10.4.1, 17.2 und 21 sowie 51.1, 51.2 und 51.5.

- “Oracle Java JDK 7 Dokumentation”, online unter[19]. Vollständige Dokumentation der J2SE (Standard Edition) inklusive der API. Im Zweifelsfall ist das die beste Quelle für alles, was Java betrifft. Trotz der etwas sperrigen Sprache ist es wichtig, sich in der API zurechtzufinden.

- “Java Code Conventions”[20]. Da der Großteil der Zeit aus der Bearbeitung von bereits geschriebenem Code besteht, ist es wichtig einen konsistenten Stil beizubehalten. Dies erleichtert das Erkennen bestimmter Elemente wie Klassen, Felder, Variablen, Strukturen und verringert die Einarbeitungszeit beim Lesen fremden Codes (siehe Kapitel 7 und 9).

[19] `http://download.oracle.com/javase/7/docs/`
[20] `http://www.literateprogramming.com/javaconv.pdf`

- “Documentation Comments with the Javadoc Tool”[21] Fast ebenso wichtig wie ein funktionierendes Programm ist eine Dokumentation der Klassen, Methoden und Felder. Diese Information wird mittels `javadoc` extrahiert und in HTML-Dokumenten (oder PDF, etc.) gesammelt oder kann beispielsweise von IDEs in der Kontexthilfe angezeigt werden. Dies erleichtert das spätere Verständnis des Code.

Glossar

- *Benchmark*: Das ist ein Test, um die Leistungsfähigkeit eines Programms zu bestimmen. Hierfür misst man die Zeit, die es zum Ausführen braucht. Dabei ist es oft nützlich, die Größe des Problems zu ändern, um eine Aussage über die Skalierbarkeit zu erhalten. Die Zeit misst man am besten über Differenzen in der Systemzeit: `System.currentTimeMillis()` in Millisekunden oder `System.nanoTime()` in Nanosekunden.

  Aufgrund der Eigenschaft der virtuellen Java-Maschine, Code zuerst nur zu interpretieren und im Laufe der Zeit die Teile in effizienten Maschinencode zu übersetzen, ist es schwierig Benchmarks zu machen. Daher ist es ratsam, erst nach vielen Wiederholungen die Zeiten zu messen, um auf aussagekräftige Werte zu kommen. Auch kann dies auf die Art verfeinert werden, dass in zwei verschachtelten Schleifen das Minimum über einen in der inneren Schleife berechneten Mittelwert berechnet wird.

- *Test*: Dies ist ein zusätzlicher Teil des Programms, welcher überprüft, ob Methoden oder Funktionen das Richtige berechnen. Beispielsweise kann eine Funktion `int = func1(int a)`, die zur übergebenen Variable a den Wert 10 addiert, dadurch kontrolliert werden, dass sie mit einen bestimmten Wert (z. B. 3) aufgerufen wird und die Ausgabe mit dem erwarteten Wert 13 verglichen wird.

  Dafür gibt es auch Frameworks wie `JUnit`, welche von allen gängigen Entwicklungsumgebungen unterstützt werden.

Versionskontrolle Bei der Entwicklung eines Programmes wird Code in Textdateien geschrieben. Meistens werden hierbei existierende Teile immer wieder überarbeitet und umgeschrieben. Es erweist sich hierbei von großem Nutzen, ältere Versionen zwischenspeichern zu können und später wieder abrufbar zu haben. Außerdem geschieht Softwareentwicklung üblicherweise in Teams, welche untereinander Änderungen an dem Code in effizienter und nachvollziehbarer Weise austauschen möchten.
All diese Anforderungen werden mittels Tools zur Versionskontrolle gelöst. Besonders hervorzuheben ist hierbei *Git*[22], welches nicht nur technisch herausragend ist, sondern sich auch einer starken Verbreitung erfreut. Mehr darüber erfährt man z. B. im Git Buch[23] und Git Repositories lassen sich bei `github.com` oder `bitbucket.org` kostenlos hosten.

[21] `http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html`
[22] `http://git-scm.com/`
[23] `http://git-scm.com/book`