Übungsbeispiele Programmierpraktikum

Harald Schilly & Andreas Ulovec
harald.schilly+pp@univie.ac.at & andreas.ulovec+pp@univie.ac.at
http://harald.schil.ly/teaching/ue-pp-13

Sommersemester 2013

Einleitung. Die folgenden Beispiele sollen mit 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 die restlichen Beispiele abgeben.
Die Beispiele werden von den Übungsleitern und Tutoren kontrolliert. Die Zahl der Sterne $\boxed{\star\star\cdot}$ ist ein Hinweis für den Schwierigkeitsgrad und gibt gleichzeitig die Anzahl der maximal zu erreichenden Punkte an. Die Gesamtpunktezahl ist 42.
Die Ziele der Übung sind, sich mit dem Implementieren von Algorithmen vertraut zu machen, Entwicklungsumgebungen kennenzulernen, Datenstrukturen zur Repräsentation von Information im Speicher verwenden zu können, Programm-Code ausreichend verständlich zu dokumentieren und Techniken zur Verarbeitung von Daten kennenzulernen. Weitere Informationen, Hinweise und Literatur befinden sich am Schluss nach den Beispielen und auf der Webseite.
Der erste Teil muss bis spätestens 16. bzw. 17. April per E-Mail an die Übungsleiter abgeben werden. Die Abgabe besteht primär aus den `*.java` Dateien, in denen jeweils Name und Matrikelnummer als Kommentar zwischen `/* ... */` 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.
Ergänzend bzw. alternativ zu Beispielen aus Teil 1 und 2 kann man auch die schwierigeren, mit $\oplus$ gekennzeichneten Zusatzbeispiele am Ende des Übungsblattes, machen. Diese zählen jeweils 6 Punkte und setzen meist Programme aus dem zweiten Teil voraus. Für die Benotung zählt die Summe aller Punkte der korrekt gelösten Beispiele, egal aus welchem Teil.
Die restlichen Beispiele sind bis spätestens 18. bzw. 19. Juni abzugeben. Am 25. bzw. 26. Juni gibt es ein Abschlussgespräch und die Benotung.
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. (Kontrollstrukturen I, ⋆) Definiere Variablen, die Werte für Kilometer, Meter, Zentimeter, Millimeter und Mikrometer beinhalten und weise ihnen sinnvolle Werte zu. Das Programm soll die Summe in Meter (`double`-Typ) dieser Längenangaben berechnen und ausgeben.

   Beispiel: 2km, 818m, 101cm, 0mm und 291µm = ? m

2. (Kontrollstrukturen II, ⋆) Kehre das vorhergehende Programm auf die Art um, dass aus einem `double`-Wert in Metern die entsprechenden Größen in Kilometer, Meter, Zentimeter, Millimeter und Mikrometer ausgeben werden.

   Teste, ob die Werte übereinstimmen, wenn das vorhergehende und dieses Programm kombiniert werden.

   Beispiel: 2819,010291 m = ?km, ?m, ?cm, ?mm, ?µm

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

   Gefragt ist nun die Summe von n gleich weit voneinander entfernten Zahlen, welche bei `lb` beginnen und bis einschließlich `ub` gehen. Berechne dies mittels einer laufenden Summe in einer `for`-Schleife.

   Beispiel: $lb = 1$, $ub = 5$, $n = 3$ liefert $1 + 3 + 5 = 9$.

4. (Kontrollstrukturen IV, ⋆) Implementiere einen Algorithmus zur Lösung einer quadratischen Gleichung $ax^2 + bx + c = 0$ in einer Funktion `static void solve(double a, double b, double c)` und gib in dieser das Ergebnis aus.

   Für die quadratische Gleichung genügt es die reellen Lösungen zu behandeln, aber vergiss *keine* Sonderfälle (z. B. "Komplexe Lösung" oder "Leere Menge" als Rückmeldung zu geben) – `NaN` als "Lösung" ist nicht erlaubt.

   Zeige, dass der Algorithmus funktioniert, indem er sowohl eine normale Doppellösung als auch einen Sonderfall zu berechnen hat.

5. (Kontrollstrukturen V, ⋆) "Würfle" in einer `do/while`-Schleife mit einem "virtuellen" 6-seitigen Würfel, bis exakt 3-mal hintereinander dieselbe Augenzahl geworfen wird. Wiederhole dieses "Experiment" 10-mal mittels einer äußeren `for`-Schleife und gib jeweils die Anzahl der dafür notwendigen Würfe aus.

   Hinweis zur Modellierung eines Würfels: `Math.random()` liefert gleichverteilte Zufallszahlen im Intervall $[0, 1)$ und "Casting" via `(int)` rundet auf eine `int`-Ganzzahl ab.
   Frage: Könnte dieser Algorithmus Probleme machen?

6. (Kontrollstrukturen VI, ⋆⋆) Hier soll die *Fläche* unter einer Funktion $f(x)$ in einem beliebigen Intervall $[a, b]$ mit $a \leq b$ 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.

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

7. (Kommandozeile, ⋆) In diesem Beispiel lesen wir auf der Kommandozeile an das Programm übergebene Parameter aus. Diese stehen in der Variablen `args`, wenn sie in der `public static void main(String[] args)` Funktion so benannt werden. Insbesondere sollen diese übergebenen Werte nicht im Programm selbst direkt an Variablen zugewiesen werden! (In Eclipse: Runtime Configuration ⇒ Arguments; Netbeans: Run ⇒ Set Project Configuration ⇒ Customize: Run ⇒ Arguments)

   Die Aufgabe besteht darin, zu jedem übergebenen Parameter (also `String`) die Länge zu ermitteln und die minimale, durchschnittliche und maximale Länge zu berechnen.

   Beispiel:

   ```
   $ java Kommandozeile A lot of code
   Minimal: 1
   Durchschnitt: 2.5
   Maximal: 4
   ```

8. (ISBN, ⋆⋆⋆) Bücher sind häufig mit einem 13-stelligen ISBN Code versehen. Die letzte Stelle ist hierbei eine Prüfziffer, die folgendermaßen berechnet wird:

$$x_{13} = \left(10 - \left(x_1 + 3x_2 + x_3 + 3x_4 + \cdots + x_{11} + 3x_{12}\right) \bmod 10\right) \bmod 10$$

   wobei sich 1 und 3 als Koeffizienten abwechseln.

   Schreibe ein Programm, welches eine ISBN Zahl von der Kommandozeile einliest, die Prüfziffer kontrolliert und eine entsprechende Rückmeldung ausgibt. Vermeide jede der Variablen einzeln zu definieren, sondern verwende z. B. zwei separate `for`-Schleifen mit passender Schrittweite um alle Indizes abzuarbeiten.

   Hinweis: Benütze die `<String>.substring(i, i+1)` und `Integer.valueOf(...)` Funktionen, um einzelne Ziffern in Zahlenwerte umzuwandeln.

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!

```java
public static void main(String... args) {
  String a1 = "foo";
  String a2 = a1;
  testStrings("a1, a2", a1, a2);
  String a3 = "foo";
  testStrings("a1, a3", a1, a3);
  String a4 = new String("fo") + "o";
  testStrings("a1, a4", a1, a4);
  String a5 = a4.intern();
  testStrings("a1 a5", a1, a5);
}

static void testStrings(String info, String a, String b) {
    System.out.println(info);
    System.out.printf("<%s>.equals(<%s>) -> %s\n", a, b, a.equals(b));
    System.out.printf("<%s>   ==   <%s>  -> %s\n\n", a, b, a == b);
}
```

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

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

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.

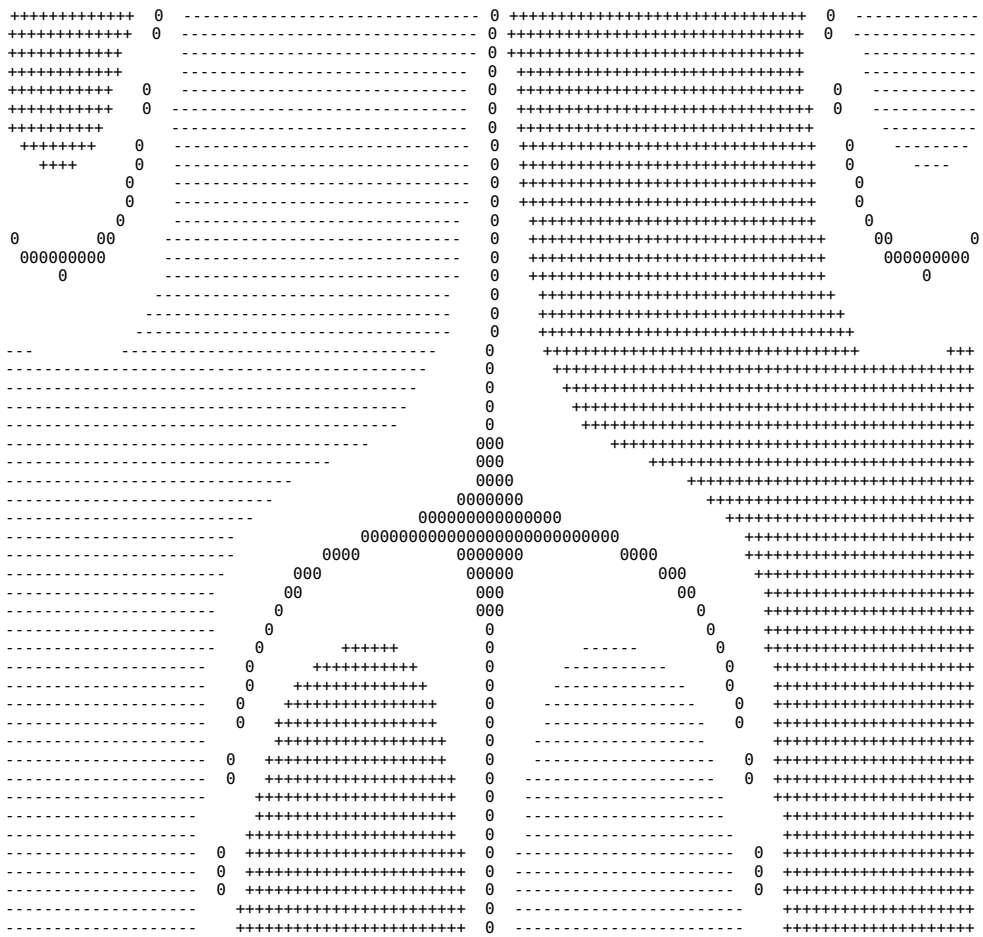
e) Finde die Klasse `HashSet` und dessen Interface `Set`. Erkläre anhand der Beschreibungen, was sie leistet und beschreibe einen möglichen Anwendungsfall.

11. (ASCII I, ★★★) Das folgende Programm soll eine mathematische Funktion in zwei Variablen in der Konsole zeichnen. Definiere hierzu eine statische Funktion $f(x, y)$ mit der Signatur `static double f(double x, double y)`, die z. B. `Math.sin(x) * y + .5 * x` berechnet.

    Weiters definiere mittels der Variablen `xlow`, `xhigh`, `ylow` und `yhigh` die Grenzen des Rechtecks, in dem diese Funktion ausgewertet wird; z. B. die Intervalle $[-5, 5], [-5, 5]$. Skaliere die Differenz von `xhigh-xlow` auf eine sinnvolle Breite von 80–100 Zeichen und ebenso die Differenz von `yhigh-ylow` in die vertikale Richtung von etwa 40–50 Zeichen in der Konsole.

    Die Funktion selbst soll dermaßen dargestellt werden, dass negative Werte < -1 mit `-`, positive > 1 mit einem `+` und Werte die beinahe 0 sind[3] mit einer `0` dargestellt werden.

    Beispiel für `Math.sin(x) * y + .5 * x`:



    Es steht es frei, weitere Zeichen zur detaillierteren Darstellung der Funktion einzuführen bzw. die Grenzwerte für die Wahl der Zeichen dynamisch an die Werte der Funktion anzupassen.

[3] z.B. `Math.abs(f(x,y)) <= 0.05`

Teil 2

12. (Rekursion I, Benchmark, ★★) Die "Telefon-Zahl" ist u. A. die Anzahl von allen möglichen paarweisen Verknüpfungen aller n Teilnehmer in einem Telefonnetzwerk[4].

    Die Formel der dazugehörigen Rekursion ist gegeben durch:

    $$T(n) = T(n-1) + (n-1)\,T(n-2)$$

    mit den Anfangswerten $T(0) = T(1) = 1$.

    Es sollen die Folgenglieder dieser Rekursion sowohl *iterativ*[5] als auch *rekursiv*[6] 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 wieviel. (Um auf aussagekräftige Zahlen zu kommen, sollte dieselbe Berechnung genügend oft wiederholt werden; siehe Glossar "Benchmark").

13. (Rekursion II, Benchmark, ★★) Verbessere die rekursive Berechnung der Folgenglieder des vorhergehenden Beispiel derart, dass Zwischenergebnisse für $T(i)$ 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. Der erwünschte Effekt ist, dass unnötige Rekursionen eingespart werden. Mache erneut ein aussagekräftiges Benchmark (siehe Glossar).

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

14. (Klassen, Vektorrechung I.1, ★★★) Dieses und folgende Beispiele handeln vom Rechnen mit dreidimensionalen Vektoren im Raum. Hierfür soll zuerst eine Klasse `Vektor3D` entsprechend der Definition eines Vektors programmiert werden, d. h. ein Tripel von Fließkommazahlen für die Koordinaten soll mittels drei `private` Feldern des Typs `double` gespeichert werden.

    Die Klasse soll mehrere Konstruktoren haben:

    - `public Vektor3D(double x, double y, double z)` – Hauptkonstruktor
    - `public Vektor3D()` – Vektor (0, 0, 0)
    - `public Vektor3D(Vektor3D v)` – Generiere ein **unabhängiges**, neues Vektor3D-Objekt aus dem existierenden Vektor3D Objekt v ("*Copy-Constructor*").

    Damit Rechnungen mit diesen Vektoren möglich werden, muss es Methoden geben, welche ein anderes Objekt der Klasse `Vektor3D` als Argument haben und ein *neues* Objekt als "Ergebnis" generieren[7].

    Insgesamt sollen folgende Methoden programmiert werden:

    `Vektor3D`:

    - `public Vektor3D add(Vektor3D other)` – Addition des gegebenen Vektors mit `other`.
    - `public Vektor3D sub(Vektor3D other)` – Subtraktion.
    - `public Vektor3D cross(Vektor3D other)` – Kreuzprodukt zweier Vektoren.

[4] `http://en.wikipedia.org/wiki/Telephone_number_(mathematics)`
[5] d.h. mit einer `for`-Schleife und mehreren Variablen
[6] d.h. mittels Aufruf der `f(n)` Funktion innerhalb ihrer selbst!
[7] d. h. es muss in der Methode ein new `Vektor3D(...)` zurückgegeben werden

- `public Vektor3D mult(double scale)` – skaliere den Vektor um den Faktor `scale`.
- `public double length()` – gibt die Länge zurück.
- `public boolean parallel(Vektor3D other)` – gib `true` zurück, wenn die beiden Vektoren parallel sind, sonst `false`.
- `public String toString()` – generiert eine für den Menschen lesbare Ausgabe (z. B. "(2.0, -3.0, 1.1)"), welche dann z. B. in `System.out.println(vektor)` automatisch für Objekte dieser Klasse aufgerufen wird.

Anschließend implementiere folgende Rechnungen und gib das Ergebnis aus:

- Die Länge des Kreuzprodukts der Vektoren $\overrightarrow{1,\ 1,\ 1}$ und $\overrightarrow{2,\ 2.0001,\ 2.0002}$.
- Sind $\overrightarrow{2.2,\ -0.1,\ 2.2}$ und $\overrightarrow{-20.20,\ -0.1010,\ 20.20}$ parallel?

15. (Klassen, Vektorrechung I.2, ★★) Eine Klasse `Punkt3D` soll ganz ähnlich für einen Punkt im Dreidimensionalen stehen. Für die drei Konstruktoren verfahre ganz analog wie für `Vektor3D`.

    `Punkt3D`:

    - `public Vektor3D diff(Punkt3D other)` – `Vektor3D` ist die Differenz zweier `Punkt3D` Objekte.
    - `public double distance(Punkt3D other)` – für die Distanz zweier Punkte benütze eine passende Methode von `Vektor3D` und dupliziere keinen Code!
    - `public static double area(Punkt3D p1, Punkt3D p2, Punkt3D p3)` – Dies soll eine *statische* (!) Methode sein, welche die Fläche des von den drei angegebenen Punkten aufgespannten Dreiecks berechnet.

    Anschließend implementiere folgende Rechnungen und gib das Ergebnis aus:

    - Die Distanz zwischen den Punkten $(-4.2,\ 42,\ 0.42)$ und $(-1,\ 0,\ 1)$.
    - Fläche des Dreiecks $\Delta\{(-1,\ -1,\ -1),\ (0,\ 5,\ 0),\ (1,\ -1,\ 3)\}$.

16. (Vektorrechnung II, Tests, ★★) Schreibe für die Klassen `Vektor3D` und `Punkt3D` aus dem vorhergehenden Beispielen *Tests*. Das heißt, eine weitere Klasse `VektorrechungTest`, welche zuerst einige fix vorgegebene Punkte und Vektoren instanziert, dann alle Methoden für jede Berechnung aufruft und explizit überprüft, ob jedes Ergebnis korrekt ist, indem jedes berechnete Ergebnis immer mit einem explizit eingegebenen Wert verglichen wird. *Alle* Methoden sollen separat getestet werden!

    Hinweis: Verwende das Schlüsselwort `assert` und füge zu den Klassen `Vektor3D` und `Punkt3D` jeweils eine Methode `public boolean equals(Vektor3D|Punkt3D other)` hinzu, welche auf Äquivalenz testet.

17. (Javadoc, Sprachelemente, Code Conventions, ★) Lies die notwendigen Kapitel der Dokumentation für "javadoc" durch. Formatiere den Code der `Vektor3D`- und `Punkt3D`-Klassen entsprechend der "Java Code Conventions" (siehe Literaturangaben).

    Insbesondere korrigiere Zeilenumbrüche, Einrückungen und beachte die korrekte Groß- und Kleinschreibung und identifiziere die unterschiedlichen Sprachelemente. Formuliere passende Beschreibungen für die Klassen selbst, für *alle* Methoden (und Parameter) und *alle* Felder, trage den eigenen Namen als Autor in den Header ein und mach einen erklärenden Querverweis (Schlüsselwort "`@link`") innerhalb des Beschreibungstextes der Methode `Punkt3D.distance(Punkt3D other)` auf die Methode `Vektor3D.length()`.

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

18. (Klassen, 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 $0 \ldots n-1$. Dokumentiere die Klasse und alle Methoden passend für "javadoc" und schreibe ein Beispielprogramm, das jede Methode aufruft und auf der Konsole ausgibt.

    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
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 eine Matrix a (neues Objekt zurückgeben!)
14 public Matrix add(Matrix a)
15 // berechne die Summe einer Zeile
16 public Double rowsum(int r)
17 // berechne die Summe einer Spalte
18 public Double colsum(int c)
19 // multipliziere die Matrix mit dem Skalar a
20 public Matrix mult(double a)
21 // "wahr", wenn quadratisch
22 public boolean isSquare()
23 // String-Darstellung
24 public String toString()
```

19. (Collections, ★★★) Üblicherweise wird in komplexeren Programmen nicht nur ein einzelnes, isoliertes Objekt benötigt, sondern eine ganze "Sammlung" von ähnlichen Objekten bzw. deren Assoziationen untereinander. Erstelle ein kurzes Programm, welches folgendes bewerkstelligt:

    - Speichere 100 zufällig gewählte Buchstabenpaare aus der Menge {`AA`, `AB`, ... `ZZ`} in
        - einem `String[]`–Array.
        - in einer `ArrayList<String>`. (Dabei gibt der Klassenname in den spitzen Klammern an, welchen Typ diese `ArrayList` beinhaltet; Stichwort: "*Generics*").
        - in einer sortierten Menge, also der Klasse `TreeSet<String>`.

    - Anschließend gib jeweils folgende Informationen an:
        - Gib alle "Sammel"-Objekte mittels `System.out.println()` aus. Tipp: Für die korrekte Ausgabe des Arrays ist `Arrays.toString(Variable)` hilfreich.
        - Ermittle, wie oft jeder der Buchstaben von A bis Z an erster, zweiter oder beliebiger Stelle vorkommt. Es soll hierfür die Datenstruktur `HashMap<String, Integer>` verwendet werden, welche zu jedem der einzelnen Buchstaben deren jeweilige Anzahl assoziativ festhält.
        - Teste, ob ein zufällig generiertes Buchstabenpaar in der jeweiligen "Sammlung" vorkommt. Wenn ja, gib die Position in der jeweiligen Datenstruktur an.
        - Wie hoch ist der prozentuelle Anteil von Palindromen?

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

20. (Statistik I, ★★★) Aufgrund der Entwicklungen der letzten Jahre ist die Finanzwirtschaft auch für Mathematiker und Mathematikerinnen interessant geworden. Dieses Beispiel soll folgendes machen:

    a) Lade historische (tägliche) Daten von Aktien herunter. Beispiel: Apples Aktie von Google Finanace im CSV Textformat[10]. (dieser Link ist der "Download to Spreadsheet" Link bei den historischen Informationen von Aktien). Genausogut gibt es auch für Googles Aktien bei Yahoo! Finance[11] unter "historical Prices" ein "Download to Spreadsheet".

    b) Parse die CSV Daten und speichere diese in einer passenden Klasse. Es bietet sich an, mehrere gleich lange Vektoren für die jeweiligen Spalten zu verwenden. Das Datum soll mittels der `SimpleDateFormat` Klasse geparsed, als `Date` Objekt gespeichert und mittels `Calendar` ausgelesen werden.

    c) Berechne folgende Informationen:

        i. Minimum und Maximum des 'Open' Wertes, an welchen Tagen war dies?

        ii. Jeweils das Datum der Top-3 Tage, an denen der Kurs am stärksten gestiegen bzw. gefallen ist, i. e. die Differenz von "Open" zu "Close" war am größten.

        iii. Programmiere eine Berechnung des 'Simple Moving Averages"[12] (SMA) von *n* gegebenen, aufeinanderfolgenden Tagen. Dieser Indikator wird an jedem Tag für die vorhergehenden *n* Tage berechnet und dessen Entwicklung wird gerne zum Erfassen von Trends verwendet. Gib an, an welchen Tagen der SMA über 7 Tage den SMA über 21 Tage kreuzt und leite ab, ob jeweils gekauft oder verkauft werden soll. Wenn der kurzfristige Trend unter den längerfristigen sinkt, ist das ein Zeichen für eine fallende Tendenz – das Gegenteilige ist eher ein Signal zu kaufen.

    Hinweise: Für das Einlesen der CSV Datei kann die Klasse `java.util.Scanner`, und für die Statistiken die Klasse `java.util.HashMap` nützlich sein!

    Zum Download baue auf folgenden Code auf:

```
1 URL url = new URL("http://...");
2 InputStreamReader in = new InputStreamReader(url.openStream());
3 BufferedReader data = new BufferedReader(in);
4 for(String line; (line = data.readLine()) != null;) {
5    System.out.println(line);
6 }
```

    Alternativ lade die CSV Datei herunter und lies sie über einen `FileReader` ein.

    Das Datumsfeld wird am besten mit einem Parser eingelesen und in einer "`Calendar`" Instanz gespeichert:

```
1 // 1. Global definiertes Datumsformat, so wie es in der CSV Datei vorkommt:
2 final DateFormat df = new SimpleDateFormat("yyyy-MM-dd");
3 ...
4 String date = "2011-01-11";
5 Date date = df.parse(date);
6 ...
7 // Spezifische Felder auslesen. Achtung: Monat Januar hat den Wert "0"!
8 final Calendar pdate = Calendar.getInstance();
9 pdate.setTime(date);
```

[8] http://download.oracle.com/javase/tutorial/collections/TOC.html
[9] http://download.oracle.com/javase/7/docs/technotes/guides/collections/index.html
[10] http://www.google.com/finance/historical?q=NASDAQ:AAPL&output=csv
[11] http://finance.yahoo.com/
[12] http://en.wikipedia.org/wiki/Moving_average#Simple_moving_average

```
10 System.out.println("day: " + pdate.get(Calendar.DAY_OF_MONTH)); //   11
11 System.out.println("month: " + pdate.get(Calendar.MONTH));      //    0
12 System.out.println("year: " + pdate.get(Calendar.YEAR));        // 2011
```

21. (Statistik II, ★★★) 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
    - Länge (minimal, durchschnittlich, maximal) der Wörter,
    - die 10 häufigsten Wörter,
    - die 10 häufigsten Wörter für jede vorkommende Wortlänge (also eine separate Zählung für jede Länge) und deren jeweilige Anzahl,
    - die 10 häufigsten Buchstabenpaare innerhalb eines Wortes (Es genügt, alle Buchstaben als Großbuchstaben zu behandeln) und deren jeweilige Anzahl.

    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.

22. (Statistik III, ASCII II, ★★) Erstelle eine Übersichtsmatrix für die Häufigkeiten aller Buchstabenpaare aus “Statistik II”. Diese Matrix ist quadratisch und jeder Eintrag soll proportional zur jeweiligen Häufigkeit durch passende Zeichen dargestellt werden; beispielsweise durch die Reihe “ ”, `.`, `x`, `I`, `0` und `X`. Die Grenzen der jeweiligen Wertebereiche sollen so gewählt werden, dass der gesamte Wertebereich gleichmäßig abgedeckt wird. Verwende hierfür “Quantile”[13]! Vergiss nicht, die Zeilen und Spalten der ausgegebenen Matrix mit den jeweiligen Buchstaben zu beschriften.
    Hinweis: Es genügt, alle Buchstaben als Großbuchstaben zu behandeln.

[13] `http://de.wikipedia.org/wiki/Quantil`

Weiterführende Beispiele

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

1. (Modellierung, ⊕) Die Aufgabe besteht darin, Schelling's "Segregation Model" (SSM)[14] 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. Etwa 80% der Felder der Matrix sollen besetzt sein und in einem iterativen Prozess sollen diejenigen "Bewohner" an einen zufälligen neuen Platz wechseln, 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.

2. (Statistik IV, ⊕) 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:

   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`

3. (Geometrie, ⊕) In den Übungsbeispielen zu den "Klassen" wurden zwei für dreidimensionale Punkte und Vektoren eingeführt. Erweitere dieses sehr einfache System zur Berechnung von geometrischen Objekten um folgende Klassen und Features:

   - `Gerade3D` – Das ist eine Klasse, die durch einen Punkt und einen Vektor definiert ist.

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

- `Kugel3D` – Diese Klasse besteht aus einem Punkt und einem `Double` Wert als Radius.
- `Ebene3D` – Diese Klasse definiert mittels eines Punktes und einenes Normalvektors eine Ebene; diese soll sich aber auch mittels drei gegebener `Punkt3D` Objekten konstruieren lassen.

Erweitere diese Klassen bzw. – nur wenn notwendig und passend – die Klassen `Vektor3D` oder `Punkt3D`, damit folgende Fragen berechnet werden können:

- Schnittpunkt einer Geraden mit einer Ebene: Rückgabe soll entweder `null` oder ein `Punkt3D` sein. Teste beide Fälle!
- Der Mittelpunkt zweier Punkte, also der Methode `Punkt3D getMittelpunkt(Punkt3D other)`.
- Berechnung der kleinsten Kugel, welche gerade noch die Ebene und einen gegebenen Punkt berührt.

Bei Interesse implementiere noch weitere Methoden und Routinen!

4. (Mathematikspiel, ⊕) Programmiere ein Lernspiel für einfaches Bruchrechnen. Das Spiel soll mindestens folgende Features haben:
   - Stelle mindestens 5 verschiedene Typen von zufällig generierten Rechnungen als Frage, beispielsweise: $2/3 + 2/5 =$, $3/4 * (1 - 1/5) =$, $(18/9)/3 =$, ... Wichtig hierbei ist, dass die jeweiligen Rechnungen einen adaptiven Schwierigkeitsgrad haben sollen. Einen höheren Schwierigkeitsgrad erreicht man durch längere Rechnungen und größere Zahlen.
   - Die jeweiligen Ergebnisse soll eingeben werden, und das Programm überprüft diese auf Richtigkeit. Beachte, dass die Brüche auch bei leicht unterschiedlicher Schreibweise richtig eingelesen werden.
   - Adaptiere dynamisch den Schwierigkeitsgrad, je nach dem wie gut der Spieler antwortet.
   - Weiters soll es verschiedene Benutzer verwalten können. Hierfür wird beim Starten des Lernspiels der Benutzername an der Kommandozeile als Parameter übergeben.
   - Protokolliere in einer <Benutzername>.csv Datei zweilenweise diese Informationen:
     - ein aktueller Zeitstempel,
     - welche Frage gestellt wurde,
     - welche Antwort gegeben wurde,
     - wie lange die Beantwortung gedauert hat,
     - ob das Ergebnis richtig war.

     Die Daten sollen passend als Zeichenketten serialisiert werden und durch Strichpunkte getrennt sein.

   Hinweis: Eingaben in der Konsole kann man so einlesen:

```
1 Scanner scanner = new Scanner(System.in);
2 ...
3 System.out.print("<TEXT>: ");
4 String eingabe = scanner.nextLine();
```

5. (Statistik V, ⊕) In diesem Beispiel ist basierend auf den Techniken aus Statistik II und III ein Programm verlangt, welches die Sprache eines kurzen, neuen Textes erraten kann.

Lese hierfür mehrere Texte von mindestens 5 verschiedenen Sprachen ein und teste den eingegebenen Text gegen diese. Eine gute Heuristik könnte sein, für jede Sprache eine Statistik der gebräuchlichsten Wörter und häufigsten Buchstabenpaare anzulegen. Anschließend werden die Wörter der unbekannten Sprache gegen diese Statistiken getestet. Berechne für jede Sprache jeweils ein Maß der Übereinstimmung und gib die wahrscheinlichste Sprache zurück.

Tipp: Es kann praktisch sein, für jede Sprache ein separates Verzeichnis anzulegen um dort die Texte zu sammeln. Diese werden beim Start automatisch geladen und verarbeitet. Die Namen der Unterverzeichnisse legen die jeweilige Sprache fest. Achte auf die Skalierbarkeit und Performance, das heißt, speichere z. B. von den eingelesenen Texten nur die 10000 häufigsten Wörter und nicht den gesamten Text, um einen Speicherüberlauf zu verhindern.

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` und nur wenige, ausgewählte Methoden und Klassen auf `public` setzen,
  - 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[15]. 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.

[15] `http://download.oracle.com/javase/7/docs/`

- “Java Code Conventions”[16]. Da der Großteil der Zeit aus der Bearbeitung von bereits geschriebener Codes 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).
- “Documentation Comments with the Javadoc Tool”[17] 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, welche häufig ausgeführt werden, 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*[18], welches nicht nur technisch herausragend ist, sondern sich auch einer starken Verbreitung erfreut. Mehr darüber erfährt man z. B. im Git Buch[19] und Git Repositories lassen sich bei `github.com` oder `bitbucket.org` kostenlos hosten.

[16] `http://www.oracle.com/technetwork/java/codeconv-138413.html`
[17] `http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html`
[18] `http://git-scm.com/`
[19] `http://git-scm.com/book`