# Übungsbeispiele Programmierpraktikum

## Sommersemester 2012

**Harald Schilly & Andreas Ulovec**
**harald.schilly@univie.ac.at & andreas.ulovec@univie.ac.at**
**http://harald.schil.ly/teaching/ue-pp-12**

**Einleitung.** Die folgenden Beispiele sollen mit der Programmiersprache Java gelöst werden. Es soll 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 im Rechteck ist ein Hinweis für den Schwierigkeitsgrad. Die Ziele der Übung sind, sich mit dem Implementieren von Prozeduren 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 17./20. April per E-Mail an die Übungsleiter abgeben werden. Die Abgabe soll primär aus den `*.java` Dateien bestehen, in denen jeweils Name und Matrikelnummer als Kommentar 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. Die Beispiele sind wieder mit Schwierigkeitsgraden (erkennbar durch die Anzahl der Sterne im Rechteck) versehen, die auch gleichzeitig die Anzahl der Punkte für ein vollständig abgegebenes Beispiel wiederspiegeln. Die Gesamtpunktezahl ist 42.
Ergänzend bzw. alternativ zu Beispielen aus Teil 1 und 2 kann man auch die schwierigeren, mit ⊕ 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 19. bzw. 22. Juni abzugeben. Am 26. bzw. am 29. 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 Jahrhunderte, Jahre, Monate, Tage, Stunden und Minuten beinhalten und weise ihnen beliebige Anfangswerte zu. Das Programm soll die Gesamtzahl der Minuten (`long`-Typ) dieser Zeitspanne berechnen und ausgeben. Dabei nehmen wir vereinfacht an, dass ein Monat immer 30 Tage und ein Jahr immer 360 Tage hat.
   Beispiel: 2 Jahrhunderte, 1 Jahr, 3 Monate, 20 Tage, 12 Stunden, und 2 Minuten = 104.357.522

2. (Kontrollstrukturen II, ⋆) Kehre das vorhergehende Programm so um, dass aus der Gesamtzahl der Minuten wieder die ursprünglichen Werte von Jahrhunderten, Jahren, Monaten, Tagen, Stunden, und Minuten berechnet und ausgeben werden. Teste, ob die Werte übereinstimmen, wenn das vorhergehende und dieses Programm verknüpft werden.
   Beispiel: 602 = 10 Stunden und 2 Minuten

3. (Kontrollstrukturen III, ⋆) Speichere in den Variablen x und y eine untere und obere positive Zahl als Grenze und in einer weiteren Variablen s die positive Schrittweite (jeweils `double`). Teste, dass $0 < x < y$ und $s > 0$ ist. Alle Zahlen sollen vom Typ `double` sein. Berechne das Produkt von **allen** Zahlen von x bis y mit der Schrittweite s, also dieser Formel:
   $$\prod_i r_i \text{ mit } r_i = x + i \cdot s,\ r_i \leq y,\ i \in \{0, 1, 2, \dots\}$$

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

5. (Kontrollstrukturen V, ⋆) Addiere in einer Schleife gleichverteilte Fließkomma-Zufallszahlen zwischen −5 bis +5, bis der Absolutbetrag der Summe den Wert einer festen Konstante K (z. B. 42) überschreitet. Wiederhole dies 10-mal und gib jeweils die Anzahl der dafür notwendigen Schleifendurchläufe aus.
   Hinweis: gleichverteilte Zufallszahlen im Intervall $[0, 1)$ liefert `Math.random()`.
   Frage: Könnte dieser Algorithmus Probleme machen?

6. (Kontrollstrukturen VI, ⋆⋆) Modelliere mittels `Math.random()` und “casting” auf den Typ `int` einen virtuellen Würfel mit den Seiten von 1 bis einschließlich 6. Würfle 9999-mal mit jeweils drei Würfeln. Wie hoch ist die ermittelte Wahrscheinlichkeit, dass ...
   a) ... alle drei Würfel gleichzeitig die selbe Zahl zeigen?
   b) ... nur zwei der drei Würfel die selbe Zahl zeigen?
   c) ... alle drei Würfel jeweils $\leq 3$ zeigen?
   d) ... bei zwei aufeinanderfolgenden Würfen die Zahl 6 jeweils exakt einmal sichtbar ist?

   Gib die Werte aus.
   Hinweis: Unter der ermittelten Wahrscheinlichkeit versteht man den Quotienten aus “erwünschten Ereignissen” / “allen Ereignissen” nach allen simulierten Würfen.

7. (Codeanalyse, ★★★) Ein Programm, so wie es in all den anderen Beispielen gefordert ist, ist eine für den Computer verständliche Kodierung einer (deutschen) Beschreibung. In diesem Beispiel drehen wir das um! Erkläre, was die folgenden beiden Programme machen, versuche den "Sinn" zu verstehen, und eventuell einen Angabetext zu verfassen, der das jeweilige Beispiel als mögliche Antwort hat.

   Programm 1:

```
1  boolean ex1(int x) {
2    if (x > 2 && x % 2 == 0) return false;
3    for(int i = 3; i <= Math.sqrt(x); i+=2) {
4      if(x % i == 0) return false;
5    }
6    return true;
7  }
```

   Programm 2:

```
1  long hilfsfunktion(int x) {
2    long p = 1;
3    for (int i = 1; i <= x; i++) {
4      p *= i;
5    }
6    return p;
7  }
8
9  double ex2() {
10   double ret = 1;
11   for (int i = 1; i < 50; i++) {
12     ret += 1. / hilfsfunktion(i);
13   }
14   return ret;
15 }
```

8. (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 }
```

9. (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 in der API. Notiere die URL und alle Methodensignaturen dieser Klasse.

d) Suche die Klasse `java.lang.String` und erkläre, was die Methoden `length()`, `substring()` und `trim()` machen.

e) Finde die Klasse `ArrayList` und erkläre anhand der Beschreibung, was sie leistet, und beschreibe einen möglichen Anwendungsfall.

10. (Kommandozeile, ⋆) In diesem Beispiel lesen wir die 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 wurde. Insbesondere dürfen die Werte nicht im Programm selbst in einer 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 berechnen und auszugeben. Außerdem soll die durchschnittliche Länge berechnet werden.

    Beispiel:

    ```
    $ java Kommandozeile Marmelade ist süß
    Marmelade: 9
    ist: 3
    süß: 3
    Durchschnitt: 5.0
    ```

11. (ASCII II, ⋆⋆⋆) Das folgende Programm soll eine mathematische Funktion in der Konsole zeichnen. Definiere eine statische Funktion $f(x)$ mit der Signatur `static double f(double x)`, die z. B. `return x*Math.sin(x)/2;` berechnet. Weiters gibt es Variablen $a$ und $b$ für die linke und rechte Grenze der Darstellung. Skaliere die Differenz von $b - a$ auf eine sinnvolle Breite von 80–100 Zeichen und ebenso in die vertikale Richtung von etwa 40–50 Zeichen ($\max(f(x)) - \min(f(x))\ \forall x \in [a, b]$). Die Funktion selbst soll als “Kette” von *zusammenhängenden* $*$-Zeichen dargestellt werden. Beispiel:

    ```
    ***
       **               *****
         *          ***
         *       **
         *      *
          *    *
            ***
    ```

    Es ist außerdem günstig Grenzen, sowie Minima und Maxima vertikaler Werte anzugeben.

## Teil 2

12. (Rekursion I, Benchmark, ⋆⋆) Berechne die Folgenglieder einer Rekursion. Es soll möglich sein mit beliebigen Anfangswerten zu beginnen. Die rekursive Formel ist:

$$f(n) = f(n-1) + 101 * f(n-2) + 503 * f(n-3) + n^2 \pmod{997}$$

mit z. B. den Anfangswerten $f(0) = f(1) = 0$ und $f(2) = 1$.
Programmiere die Folge sowohl *iterativ* als auch *rekursiv*.
Teste diese rekursive Methode im Vergleich zum iterativen Ansatz in einem Benchmark (um auf aussagekräftige Zahlen zu kommen, sollte dieselbe Berechnung genügend oft wiederholt werden; siehe Glossar "Benchmark").
Berechne die ersten 30 Zahlen und überlege, welches Verhalten diese Zahlenfolge zeigt!

13. (Rekursion II, Benchmark, ⋆⋆) Verbessere die rekursive Berechnung der Folgenglieder im vorhergehenden Beispiel derart, dass Zwischenergebnisse für $f(i)$ in einem Array an der Position $i$ gespeichert werden. Ist das Zwischenergebnis bekannt, gib es zurück – wenn nicht, führe die Berechnung aus und speichere das Ergebnis. Der Effekt ist, dass unnötige Rekursionen eingespart werden. Mache erneut ein aussagekräftiges Benchmark (siehe Glossar).

14. (Klassen, Vektorrechung I.1, ⋆⋆⋆) Programmiere eine Klasse `Vektor2D`, welche das Rechnen mit zweidimensionalen Vektoren im mathematischen Sinn ermöglicht. Ein Vektor ist definiert als ein Paar von Fließkommazahlen für die erste bzw. zweite Richtung in der Ebene. Die jeweiligen Richtungen bzw. Koordinaten sollen *private* Felder des Typs `double` sein.

    Die Klasse soll mehrere Konstruktoren haben:

    - `public Vektor2D(double r, double i)` – Hauptkonstruktor
    - `public Vektor2D()` – Vektor mit dem Wert (0, 0)
    - `public Vektor2D(Vektor2D c)` – Generiere ein **unabhängiges**, neues Vektor2D-Objekt aus einem existierenden Vektor2D Objekt ("*Copy-Constructor*").

    Damit das Rechnen mit diesen Vektoren möglich ist, muss es Methoden geben, welche ein anderes Objekt der Klasse `Vektor2D` als Argument haben und ein *neues* Objekt als "Ergebnis" generieren.

    Insgesamt sollen folgende Methoden programmiert werden:

    `Vektor2D`:

    - `public Vektor2D add(Vektor2D other)` – Addition des Vektors `other` zu dem des entsprechenden Objekts (`this`).
    - `public Vektor2D sub(Vektor2D other)` – Subtraktion.
    - `public double scalar(Vektor2D other)` – Skalarprodukt zweier Vektoren.
    - `public Vektor2D mult(double scale)` – Skaliere um den Faktor `scale`.
    - `public double winkel(Vektor2D other)` – Winkel zwischen den beiden Vektoren, in Radiant.
    - `public double length()` – gibt die Länge zurück (verwende `Math.hypot()`).

- `public boolean parallel(Vektor2D 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, -3)"), 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:

- Der Winkel zwischen den Vektoren $\overrightarrow{1, 1}$ und $\overrightarrow{2, 2.0001}$.
- Sind $\overrightarrow{4.4, -1.1}$ und $\overrightarrow{-44.44, 11.11}$ parallel?

15. (Klassen, Vektorrechung I.2, ★★) Weiters brauchen wir auch eine Klasse `Punkt2D`, welche ganz ähnlich für einen Punkt im Zweidimensionalen steht. Für die Konstruktoren verfahre ganz analog wie für `Vektor2D`.

    `Punkt2D`:

    - `public Vektor2D diff(Punkt2D other)` – Differenz zweier Punkte: `Vektor2D`.
    - `public double distance(Punkt2D other)` – Distanz zweier Punkte. Benütze eine passende Methode bei `Vektor2D` und dupliziere keinen Code!
    - `public int quadrant()` – Gib zurück, in welchen Quadranten sich der Punkt befindet.
    - `public static double area(Punkt2D p1, Punkt2D p2, Punkt2D p3)`
      Dies soll eine *statische* (!) Methode sein, welche die Fläche des Dreiecks berechnet, das von den drei angegebenen Punkten aufgespannt wird.

    Anschließend implementiere folgende Rechnungen und gib das Ergebnis aus:

    - Die Distanz zwischen den Punkten $(-4.4, 42)$ und $(11, 3.1415)$.
    - Fläche des Dreiecks $\Delta\{(-1, -1), (0, 5), (3, -1)\}$.

16. (Vektorrechnung II, Tests, ★★) Schreibe für die Klassen `Vektor2D` und `Punkt2D` aus dem vorhergehenden Beispiel 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 es jeweils mit einem explizit eingegeben bekannten Ergebnis verglichen wird. *Alle* Methoden sollen einzeln getestet werden!
    Hinweis: Verwende das Schlüsselwort `assert` und erweitere die Klasse `Vektor2D` und `Punkt2D` um eine Methode `public boolean equals(Vektor2D|Punkt2D other)`, welche auf Äquivalenz testet.

17. (Javadoc, Sprachelemente, Code Conventions, ★) Lies die notwendigen Kapitel der Dokumentation für "javadoc" durch. Formatiere den Code der `Vektor2D`- und `Punkt2D`-Klassen entsprechend der "Java Code Conventions" (siehe Literaturangaben).

    Insbesondere korrigiere Zeilenumbrüche, Einrückungen und beachte die korrekte Groß-/Kleinschreibung. Lerne dabei die unterschiedlichen Sprachelemente zu identifizieren! Schreibe für den Code passende Beschreibungen für die Klassen selbst, *alle* Methoden (und Parameter) und Felder, trage den eigenen Namen als Autor in den Header ein und mach einen erklärenden Querverweis (Schlüsselwort "`@link`") innerhalb des Beschreibungstextes der Methode `Punkt2D.distance(Punkt2D other)` auf die Methode `Vektor2D.distance(Vector2D other)`.
    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 `Vektor2DMatrix`, welche die wichtigsten Matrix-Operationen implementiert. Die Klasse soll allgemeine $m \times n$ Matrizen mit `Vektor2D` Einträgen als Elemente ermöglichen. 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 `Vektor2DMatrix` beherrschen:

```
1  // Constructor für eine n x n Matrix
2  public Vektor2DMatrix(int n)
3  // Constructor für eine rows x cols Matrix
4  public Vektor2DMatrix(int rows, int cols)
5  // Zufallsmatrix, n x n Matrix mit zufälligen Einträgen
6  public static Vektor2DMatrix random(int n)
7  // Größe der Matrix, [rows, cols]
8  public int[] size()
9  // gib Element an Position (r,c) zurück
10 public Vector2D get(int r, int c)
11 // setze Element (r,c) auf den Wert v
12 public void set(int r, int c, Vektor2D v)
13 // addiere eine Matrix a
14 public Vektor2DMatrix add(Vektor2DMatrix a)
15 // berechne die vektorielle Summe eine Zeile
16 public Vector2D rowsum(int r)
17 // berechne die vektorielle Summe eine Spalte
18 public Vector2D colsum(int c)
19 // multipliziere Matrix mit Skalar a
20 public Vektor2DMatrix mult(double a)
21 // Darstellung
22 public String toString()
```

Tipps:

- `toString()` soll die `toString()` Methode der `Vektor2D`-Klasse benützen und stellt die Matrix in tabellarischer Form z. B. mittels `String.format(...)` dar.
- Für die Matrix mit den zufälligen Einträgen implementiere eine `public static Vector2D random()` Methode in der `Vektor2D` Klasse.

Bemerkung: Falls die Beispiele für Vektorrechnung ausgelassen werden, und nur dann, so kann dieses Beispiel für den Typ `double` gemacht werden.

19. (Collections, ★★) Oft wird nicht nur ein Objekt benötigt, sondern eine ganze "Sammlung" von ähnlichen Objekten (Instanzen derselben Klasse, bzw. mit demselben Interfaces), welche wiederum in einem "Sammel"-Objekt gespeichert werden. Das ist eine Weiterentwicklung von Arrays (<Klasse>[]) und kann in den unterschiedlichsten Situationen nützlich sein. Erstelle ein kurzes Programm, welches Folgendes bewerkstelligt:
    - Speichere 20 zufällig gewählte ganze Zahlen von $-100$ bis 100 in
        - einem `int[]`–Array.
        - in einer `ArrayList<Integer>`.
          (Dabei gibt der Klassenname in den spitzen Klammern an, welchen Typ diese `ArrayList` beinhaltet; Stichwort "*Generics*").
        - in einer sortierten Menge, also der Klasse `TreeSet<Integer>`.
    - Anschließend mach jeweils Folgendes:
        - Gib alle "Sammel"-Objekte mittels `System.out.println()` aus. Tipp: Für die korrekte Ausgabe des Arrays ist `Arrays.toString(Variable)` hilfreich.

- Berechne die Summe, den arithmetischen Durchnschnitt und die minimale und maximale Zahl, jeweils separat für jede der “Sammlungen”.
- Teste, ob eine zufällig generierte Zahl in der jeweiligen “Sammlung” vorkommt. Wiederhole diesen Test so lange, bis er “wahr” ist und gib die Position dieser generierten Zahl in der jeweiligen Datenstruktur aus.

Weiterführende Informationen hier[2] und in der API hier[3].
Bemerkung: Seit Java mit der Version 1.5 sind `int` und `Integer` äquivalent – das wird “autoboxing” genannt.

20. (Collections II, $\boxed{\star}$) Es soll das Wechselgeld in einem Kaffeeautomaten berechnet werden. Ein Kaffee kostet $k$ Cent (z. B. 55) und es wird ein Betrag $s$ Cent in einer bestimmten Stückelung eingeworfen. Die Stückelung ist eine assoziative Liste (`Map<Integer, Integer>`) von `Integer`-Werten der Stückelungsgröße zu deren Anzahl als `Integer` Wert. z. B.: $\{ s[1] : 0, s[5] : 3, s[10] : 0, s[50] : 0, s[100] : 2, s[200] : 0 \}$.
Diese bedeutet, dass keine 1-Cent Münze, 3-mal eine 5-Cent Münze, keine mit 10 oder 50 Cents, 2-mal eine 1 Euro Münze und keine 2-Euro Münze eingeworfen wurde. Beachte, der Algorithmus soll mit einer beliebiger Stückelung arbeiten können, dies ist nur ein Beispiel!

    Ausgabe: Gib eine assoziative Map $w$ mit passendem Wechselgeld aus, also $k = s - w$, sodass so wenig Münzen wie möglich zurückgegeben werden!

21. (Statistik I, $\boxed{\star\star\star}$) 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[4]. (dieser Link ist der “Download to Spreadsheet” Link bei den historischen Informationen von Aktien).
    Genausogut gibt es auch für Googles Aktien bei Yahoo! Finance[5] unter “historical Prices” ein “Download to Spreadsheet”. Die täglichen Daten für Googles Aktie gibt es hier[6].
    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 programmieren. Vergiss nicht, auch das Datum mitzuspeichern.
    c) Berechne folgende Kennzahlen:
        i. Minimum und Maximum des ’Close’ Wertes, an welchen Tagen war dies?
        ii. Mittelwert und Median des ’Close’ Wertes.
        iii. Jeweils das Datum der Top-3 Tage, an denen der Kurs am stärksten gestiegen bzw. gefallen ist, i. e. Differenz von “Open” zu “Close”.
        iv. Berechne eine monatliche Zusammenfassung der täglichen Daten auf folgende Art: Für jedes Monat soll der nach dem “Volumen” gewichteten Mittelwert[7] des Mittelwerts aus dem “Low” und “High” Wertes berechnet werden. Gib die Liste dieser Kennzahlen mit dem jeweiligen Monat und Jahr aus.

[2] `http://download.oracle.com/javase/tutorial/collections/TOC.html`
[3] `http://download.oracle.com/javase/6/docs/technotes/guides/collections/index.html`
[4] `http://www.google.com/finance/historical?q=NASDAQ:AAPL&output=csv`
[5] `http://finance.yahoo.com/`
[6] `http://ichart.finance.yahoo.com/table.csv?s=GOOG&a=07&b=19&c=2004&d=01&e=20&f=2012&g=d&ignore=.csv`
[7] Mittelwert mit Gewichtungsvektor $w_i$: $\bar{x}|_w = \left(\sum_{i=1}^{n} w_i \cdot x_i\right) \Big/ \left(\sum_{i=1}^{n} w_i\right)$

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:

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

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

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

```
// 1. Global definiertes Datumsformat, so wie es in der CSV Datei vorkommt:
final DateFormat df = new SimpleDateFormat("yyyy-MM-dd");
...
String date = "2011-01-11";
// 2. Instanz des "Calendars", Datum setzen.
final Calendar pdate = Calendar.getInstance();
pdate.setTime(df.parse(date));
// 3. Spezifische Felder auslesen. Achtung: Monat Januar hat den Wert "0"!
System.out.println("day: " + pdate.get(Calendar.DAY_OF_MONTH)); //   11
System.out.println("month: " + pdate.get(Calendar.MONTH));      //    0
System.out.println("year: " + pdate.get(Calendar.YEAR));        // 2011
```

22. (Sortieren I, ★★ + ★) Erstelle eine Klasse, welche Wörter aus einer Textdatei nach einem beliebigen Verfahren sortiert. Z. B. funktioniert das "Bubble-Sort"-Verfahren so, dass durch fortlaufendes Vertauschen von benachbarten Einträgen die Liste sortiert wird. Es wird die Liste dabei so lange abgearbeitet, bis keine Vertauschungen mehr notwendig sind.

    Die Wörter sollen mittels eines `FileReader` oder `Scanner` eingelesen und anschließend in einer neuen Textdatei ausgegeben werden. Beachte, dass außschließlich Wörter und keine Satzzeichen mitgelesen werden; konvertiere außerdem alle Zeichen zu Kleinbuchstaben.

    - Der Algorithmus muss eigenständig implementiert werden. Die Verwendung fertiger Routinen wie zum Beispiel der `sort()`-Methode der Klasse `java.util.Arrays` ist nur für Vergleichszwecke zulässig.
    - Das Einlesen und der Sortiervorgang sollen in separaten Methoden gekapselt werden. Die Methoden sollen geeignete Parameter und Rückgabewerte enthalten.
    - Miss die durchschnittliche, minimale und maximale Geschwindigkeit des Programmes für jeweils unterschiedlich große Mengen an zu sortierenden Wörtern (Benchmark, siehe Glossar). Vergleiche mit anderen, mache Vermutungen über die Komplexität, implementiere eventuell eine zweite Sortiermethode!
    - Einen Extrapunkt ★ gibt es, wenn die Sortiermethode ganz allgemein für das Interface `Comparable` programmiert wird. Hierzu soll die Sortierroutine auf dieser Liste operieren: `List<Comparable>` und die Methode `compareTo(...)` verwenden.
      Teste die dadurch gewonnene Flexibilität, indem nun derselbe Code außer der Liste von `Strings` auch eine Liste von zufälligen `Integer`-Objekten sortieren kann.

23. (Statistik II, ASCII III, ★★) Erstelle für die Daten aus "Statistik I" ein Histogramm der häufigsten Kurswerte.

Stelle es als Balkendiagramm in der Kommandozeile dar und gib außerdem die Anzahl der Werte in dem jeweiligen Intervall als rechtsbündige Ganzzahl aus (verwende `System.out.printf`). Achte auf eine sinnvolle horziontale Skalierung.
Beispiel: Der Minimale Kurs lag bei “10.0” und der Maximale bei “22.0”. Das Histogram teilt diesen Bereich in z. B. 9 gleich große “Bins” ein und zählt, wie oft ein Kurswert in diesem Intervall liegt.

```
10.0   1 *
11.5   2 *
13.0   8 ***
14.5  11 *****
16.0   9 ****
17.5  23 ***********
19.0   5 ***
20.5   6 ***
22.0   1 *
```

Hinweis: Eine “vernünftige” Anzahl von “Bins” ist die aufgerundete Quadratwurzel aus der Anzahl der Datensätze.

## Weiterführende Beispiele

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

1. (Produktverwaltung, ⊕) Dieses Beispiel hat zum Ziel, eine Verwaltung für Produkte in einem Geschäft zu schreiben. Grundsätzlich soll es zuerst eine Liste von Produkten verwalten können, anschließend soll es möglich sein "Einkäufe" zu erstellen. Von diesen wird dann der Gesamtpreis + Mehrwertsteuer in einem Rechnungszettel ausgegeben.

   Alle Preis-Werte sollen als `int` im Cent-Betrag gespeichert werden; das verhindert Rundungsfehler die bei `double` passieren könnten.

   Erstelle folgende Klassen:

   - `IMitPreis` – Ein "Interface" das all jene Klassen implementiert, die einen Preis angeben:
     - `int getNettoPreis()`
     - `int getMwstInEuro()`
     - `int getPreisMitMwst()`
     - `int getMwst()`
   - Klasse `Produkt implements IMitPreis`:
     - Konstruktor `Produkt(String name, int artikelNr, int stueckpreis, boolean lebensmittel)`
       die `artikelNr` soll eindeutig sein; das Flag `lebensmittel` dient zur Berechnung der Mehrwertsteuer, also 10% für Lebensmittel und 20% für alles andere.
     - `String getName()`
     - `int getArtikelNr()`
   - Klasse `Einkauf implements IMitPreis`:
     - Konstruktor `Einkauf(Date date)`
     - die Klasse beinhaltet in privaten Feldern eine Liste von Produkten als `ArrayList<Produkt>` und das Kaufdatum als `Date` Objekt.
     - `addProdukt(Produkt)` – fügt das Produkt hinzu.
     - `List<Produkt> getProdukte()` – die Liste aller Produkte.
   - Klasse `Produktkatalog` – diese beinhaltet eine Liste aller Produkte im Geschäft. Speichere alle Produkt-Typen in einer `HashMap<Integer, Produkt>`, welche die "`artikelNr`" mit dem jeweilige Produkt assoziiert.

     Eine Methode `Produkt getProdukt(int artikelNr)` gibt den Artikel mit gegebener Nummer zurück.
   - Klasse `BonDrucker` – diese Klasse gibt den Kassabon für ein Objekt `Einkauf` aus. Die soll so formatiert sein, dass alle Zahlen richtig positioniert sind (`System.out.printf`).
     Beispiel:

     ```
     ++++++++++ KASSABON ++++++++++
     Ihr Einkauf vom
     Tue Feb 21 12:00:34 CET 2012
     ```

```
------------------------------
Artikel        Preis/Stk  MWSt
------------------------------
10001 Apfel      0.34     10%
10002 Birne      0.34     10%
10002 Birne      0.34     10%
20001 Wodka      9.99     20%
==============================
Rechnungssumme: 11.01
          MWSt:  2.10
Vielen Dank für Ihren Einkauf!
```

- eine `Main` Klasse, die in der `main()` Routine ein Beispiel enthält.
  Skizze:

```
1 ProduktKatalog katalog = new ProduktKatalog();
2 katalog.addProdukt(new Produkt("..." ... );
3 ...
4 Einkauf einkauf = new Einkauf(new Date());
5 einkauf.addProdukt(katalog.getProdukt(...));
6 ...
7 BonDrucker drucker = new BonDrucker();
8 drucker.drucke(einkauf);
```

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. (Große Zahlen, ⊕) Schreibe eine Klasse `Ganzzahl`, welche das Rechnen mit beliebig großen ganzen Zahlen aus $\mathbb{Z}$ ermöglichen soll. Intern soll eine `ArrayList` vom Typ `Byte` zur Speicherung der einzelnen Ziffern von 0 bis 9 verwendet werden.
   Konstruktoren:

   - `Ganzzahl()` – vom Wert 0
   - `Ganzzahl(String)` – Parse den String, bestehend aus den Zeichen "-" und "0" bis "9".
   - `Ganzzahl(Ganzzahl)` – Copy-Constructor

- `Ganzzahl(Integer)` – aus herkömmlichen Integer erzeugen

Die folgenden Rechnungen sollen möglich sein und deren korrekte Funktionsweise mit entprechenden Tests untermauert werden.

- `String toString()` – Darstellung als Zeichenkette
- `Ganzzahl add(Ganzzahl other)` – Addition
- `Ganzzahl add(int other)` – Addition mit üblichem `int`
- `Ganzzahl mult(Ganzzahl other)` – Multiplikation
- `Ganzzahl mult(int other)` – Multiplikation mit üblichem `int`
- `Ganzzahl sub(Ganzzahl)` – Subtraktion
- `Double approx()` – Approximation der Ganzzahl als Fließkommazahl des Typs `Double`

Teste, ob sich auch noch Zahlen mit mehr als 1000 Stellen behandeln lassen!
Bemerkung: Es ist natürlich nicht erlaubt, `BigInteger` oder `BigDecimal` zu verwenden!

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

   - `Gerade2D` – Das ist eine Klasse, die einen Punkt und einen Vektor enthält.
   - `Kreis2D` – Das ist eine Klasse bestehend aus einem Punkt und einem `Double` Wert als Radius.
   - `Dreieck2D` – Diese Klasse besteht aus 3 Punkten.

   Erweitere diese Klassen bzw. (nur wenn notwendig und passend!) die Klassen `Vektor2D` oder `Punkt2D`, damit folgende Fragen berechnet werden können:

   - Schnitt von zwei Geraden: Rückgabe entweder `null` oder ein `Punkt2D`. Teste beide Fälle!
   - Mittelpunkt zwischen zwei Punkten, `Punkt2D getMittelpunkt(Punkt2D other)`.
   - Die Gerade, welche im rechten Winkel auf einen Vektor oder Gerade steht und durch einen gegebenen Punkt geht; i. e. `Gerade2D orthogonal(Punkt2D, Vektor2D)` und die Klasse `Gerade2D` hat eine Methode `Vektor2D getVektor()`.
   - Umkreis eines Dreiecks, i. e. `Kreis2D getUmkreis()`.

   Bei Interesse implementiere noch weitere Methoden und Routinen!

5. (Mathematikspiel, ⊕) Programmiere ein Lernspiel für Grundrechnungsarten. Das Spiel soll mindestens folgende Features haben:

   - Unterschiedliche zufällig generierte Rechnungen als Frage stellen. Beispielsweise: $2 + 2 =$, $3 * (21 - 5) + 1 =$, $18/9 + 1 =$, ...
   - Der Spieler soll das Ergebnis eingeben können, und das Programm überprüft dies. Bei Brüchen bzw. Fließkommazahlen als Ergebnis beachte, wenn nötig den Bruch richtig einzulesen bzw. bei Fließkommazahlen Rundungsfehler zu berücksichtigen.
   - Weiters soll es verschiedene Benutzer verwalten können. Hierfür soll der Benutzername an der Kommandozeile als Parameter übergeben werden. Siehe Beispiel “Kommandozeile”.
   - Protokolliere in einer <Benutzername>.log Datei Folgendes:
     - einen Zeitstempel

  - welche Frage gestellt wurde
  - welche Antwort gegeben wurde
  - wie lange die Beantwortung gedauert hat
  - ob das Ergebnis richtig war
- Adaptiere den Schwierigkeitsgrad je nach dem wie gut der Spieler antwortet. Einen höheren Schwierigkeitsgrad erreicht man durch längere Rechnungen und größere Zahlen.

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();
```

### 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”, 5. Auflage, 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 5/6 Dokumentation”, online unter[8]. 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”, online unter[9]. 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”, online unter[10]. 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 Verständnis von Code später und für andere.

**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 (entspricht 1/1000 Sekunde!) 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, mehrere Wiederholungen des Befehls zu machen, 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.

  Neben der Zeit, kann auch der verbrauchte Arbeitsspeicher interessant sein.

  Genaueres Monitoring ist mittels Tools wie `jvisualvm` möglich. Siehe[11] über CPU und Memory Profiling von Applikationen.
- *Rekursion*: So nennt sich eine Technik, wenn sich eine Funktion selbst erneut aufruft. Beachte stets, dass es immer einen passenden Abbruch gibt, um unbegrenzt tiefe Rekursionen zu vermeiden. In einigen Beispielen kann es günstig sein Zwischenergebnisse zu speichern, um wiederholtes Berechnen derselben Sub-Rekursion zu vermeiden. Das nennt sich “Dynamic Programming”.
- *Test*: Ein Test 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.

[8] `http://download.oracle.com/javase/6/docs/`
[9] `http://www.oracle.com/technetwork/java/codeconv-138413.html`
[10] `http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html`
[11] `http://download.oracle.com/javase/6/docs/technotes/guides/visualvm/index.html`