Programmieren-Aufgabenkatalog

für die Lehrveranstaltungen:
Programmieren I und II sowie
Grundlagen und Vertiefung der Programmierung

16. März 2018

Inhaltsverzeichnis

Einleitung

Programmieren lernt man – wie Fahrrad fahren oder Klavier spielen – nicht vom zusehen oder hören einer Vorlesung. Donald Knuth spricht letztlich nicht umsonst von "The Art of Programming". Eine Vorlesung ist wertvoll, um wesentliche Inhalte zu strukturieren und einen Pfad vorzuschlagen, der einem das Erwerben von Programmierfähigkeiten erheblich erleichtern kann. Jedoch ist eine Vorlesung nur eine Seite der Medaille und reicht alleine nicht aus.

Programmieren lernt man nur durch Programmieren.

Dieser Aufgabenkatalog beinhaltet mehrere Aufgabenblätter, die im Rahmen verschiedener Lehrveranstaltungen durch Studierende im Rahmen praktischer Übungen und Praktika zu bearbeiten sind. Dieser Aufgabenkatalog wird in folgenden Lehrveranstaltungen genutzt:

Lehrveranstaltung	**LV Kürzel**	**Semester**	**Studiengang**	**Hochschule**
Grundlagen der Programmierung	GProg	1	Informationstechnologie und Design (ITD)	FH Lübeck
Vertiefung der Programmierung	VProg	2	Informationstechnologie und Design (ITD)	FH Lübeck
Programmieren I	Prog I	1	Informatik/Softwaretechnik (INF)	FH Lübeck
Programmieren II	Prog II	2	Informatik/Softwaretechnik (INF)	FH Lübeck

Der Aufgabenkatalog wird kontinuierlich fortgeschrieben und beinhaltet Aufgabenblätter mit abgestimmten Aufgaben zu den einzelnen Units der oben genannten Vorlesungen. Der vorliegende Katalog unterscheidet dabei folgende Aufgabenarten:

- **Hausaufgaben** sind im Rahmen der Vorbereitung auf die nächste Übung/das nächste Praktikum zu bearbeiten und dienen vor allem der Nacharbeitung des theoretischen Vorlesungsstoffs.
- **Abgabeaufgaben** sind bewertungsrelevant und sind ggf. über das Moodle System hochzuladen.
- **Übungsaufgaben** sind während einer Übung oder im Rahmen eines Praktikums in Präsenz durch die Studierenden zu bearbeiten. Diese Aufgaben können jedoch vorher wie eine Hausaufgabe berarbeitet werden.

Nicht alle im folgenden Katalog aufgeführten Aufgabe werden auch im Rahmen einer Lehrveranstaltung Aufgabe für Aufgabe bearbeitet. Welche Aufgaben zu bearbeiten und abzugeben sind, geht aus den über Moodle bereitgestellten Informationen hervor und wird aufgrund des Lernfortschritts der Studierenden kurz vor den Übungs- bzw. Praktikumsterminen festgelegt. Ggf. werden auch Ergänzungsaufgaben in den Katalog eingepflegt – so dies aus Einschätzung des Lernfortschritts sinnvoll erscheint oder interessante Aufgaben aus Reihen der Studierenden, sonstigen Quellen herangetragen werden oder sich aufgrund der Weiterentlickung genutzter Programmiersprachen (bspw. die Java Version 8) anbieten.

Alle nicht im Rahmen der Lehrveranstaltung oder Übung/Praktika bearbeiteten Aufgaben können im Selbststudium durch die Studierenden bearbeitet werden. Dies wird auch für eine **zielgerichtete Klausurvorbereitung** empfohlen.

Auch für das anstehende Semester wurden wieder mehrere Aufgaben diesem Katalog hinzugefügt. Diese sind als **!!! NEU !!!** gekennzeichnet.

Ich wünsche Ihnen viel Erfolg und Erkenntnisgewinn bei der Bearbeitung der nachfolgenden "Fingerübungen".

Lübeck im September 2016

Nane Kratzke

Teil I.

Aufgaben zu Unit 1 Grundbegriffe und Softwareentwicklungsumgebung

1. Installation des JAVA Software Development Kits (SDK)

Zum Entwickeln von JAVA Programmen benötigen Sie die erforderlichen Tools wie Compiler und die JAVA Virtual Machine. In dieser Übung werden Sie die entsprechenden Tools installieren. Die Anleitungen sind einmal für Windows und Linux Systeme und einmal für Mac OS X Systeme formuliert. Führen Sie die Aufgabe für das System aus, mit dem Sie beabsichtigen die Übungen zu bestreiten.

1.1. Aufgabe: Installieren der JAVA SDK für Windows, Mac OS X, Linux oder Solaris

Installieren Sie die JAVA Entwicklungsumgebung wie folgt:

- Öffnen Sie in einem Webbrowser die folgende URL, um die die JAVA Platform JDK (Version 8, Standard Edition) herunterzuladen

  `http://www.oracle.com/technetwork/java/javase/downloads/index.html`

- Wählen Sie die JDK Version, die für Ihr System passt und laden Sie die Installationsdatei herunter (ca. 80 bis 100 MB).
- Starten Sie die Installationsdatei und folgen Sie den Anweisungen.
- Nach erfolgreicher Installation fahren Sie mit Aufgabe 2.1 fort.

2. Installation der Software Entwicklungsumgebung ECLIPSE

2.1. Aufgabe: Installieren von ECLIPSE

Im Rahmen der Übungen zu Programmieren I und II wird die integrierte Softwareentwicklungsumgebung ECLIPSE verwendet. Eine Softwareentwicklungsumgebung (auch IDE genannt – Integrated Development Platform) ist ein Programm, dass zum Entwickeln, Testen und Ausführen von Software verwendet wird. ECLIPSE ist im JAVA Umfeld die am weitesten verbreitete IDE, die durch zahlreiche Plugins ergänzt werden kann. Die Wahrscheinlichkeit, dass Sie mit dieser Umgebung in ihrem weiteren Studium und im anschließenden Beruf zu tun haben werden, ist sehr hoch, daher wird ECLIPSE als Lehrumgebung verwendet.

ECLIPSE ist selber in JAVA entwickelt und daher plattformunabhängig. Die Anleitungen sind daher sowohl für Windows, Linux als auch Mac OS X Systeme gültig.

- Öffnen Sie in einem Webbrowser die URL `http://www.eclipse.org/downloads/`
- Laden Sie die Installationsdatei für die **ECLIPSE Standard IDE for JAVA Developers** herunter (ca. 200 MB, je nach System).
- Starten Sie die Installationsdatei und folgen Sie den Anweisungen.
- Starten Sie anschließend das neu auf ihrem Rechner installierte Programm ECLIPSE.
- Fahren Sie fort mit der Bearbeitung der Aufgabe 2.2.

2.2. Aufgabe: Anlegen eines ersten Hello World Programms in ECLIPSE

ECLIPSE verwaltet zu entwickelnde Software in sogenannten Projekten. In den Übungen werden Sie pro Aufgabe ein Projekt anlegen und in diesem Software entwickeln. Teile dieser Projekte sind bewertungsrelevant und müssen durch Sie in Moodle hochgeladen werden. In dieser und den folgenden Aufgabe werden Sie lernen wie Sie Projekte erstellen, diese exportieren und in Moodle hochladen sowie Projekte wieder in ECLIPSE importieren können. Diese Form des Datenaustauschs ist für abzugebende Übungen wichtig, da nicht importierbare Projekte nicht bewertet werden können.

- Starten Sie ECLIPSE (sofern noch nicht geschehen)
- Wählen Sie **File** -> **New** -> **JAVA Project** (ein Dialog Fenster erscheint)
- Geben Sie folgende Daten ein:
  - **Project Name:** HelloWorld (zusammengeschrieben)
  - Anschließend: **Finish**
- Im Project Explorer erscheint nun das neu angelegte Projekt HelloWorld

- Selektieren Sie dieses Projekt und öffnen Sie mit der rechten Maustaste ein Context Menu
- Wählen Sie **New** -> **Class** (ein Dialog Fenster erscheint)
- Geben Sie folgende Daten ein:
  - **Name:** HelloWord (zusammengeschrieben)
  - Anschließend: **Finish**
- Eine Quelltextdatei mit folgenden Inhalt wurde nun angelegt.

```
1 public class HelloWorld {
2
3 }
```

- Fügen Sie nun in diesen Quelltext eine `main` Methode ein, die "Hello World!!!" auf der Konsole ausgeben soll. Kopieren Sie hierzu bitte den folgenden Quelltext zwischen die geschweiften Klammer {} der Klasse `HelloWorld`.

```
1 public static void main(String[] args) {
2   System.out.println("Hello World !!!");
3 }
```

- Selektieren Sie die Datei `HelloWorld.java` im Project Explorer und wählen sie über die rechte Maustaste **Run As** -> **JAVA Application**
- Im Konsolenfenster unterhalb des Quelltexteditors sollte nun der Text "Hello World !!!" erscheinen.

Fahren Sie nun bitte mit Bearbeitung der Aufgabe 2.3 fort.

2.3. Aufgabe: Exportieren eines ECLIPSE Projekts

Exportieren Sie nun bitte das in Aufgabe 2.2 angelegte Projekt als ECLIPSE Projektarchiv. Gehen Sie hierzu bitte wie folgt vor:

- Selektieren Sie im Projekt Explorer das HelloWorld Projekt.
- Über die rechte Maustaste wählen Sie bitte Export (ein Export Dialog geht auf).
- Wählen Sie **General** -> **Archive File**
- Geben Sie folgende Daten im Export Dialog ein:
  - **To archive file:** Wählen Sie über den File Dialog den Desktop als Speicherplatz und vergeben Sie den folgenden Namen: `Prog-Aufgabe2.3-MaxMustermann.zip` (für MaxMustermann geben Sie bitte ihren eigenen Namen an).
  - Anschließend: **Finish**
- Das Archivfile sollte nun auf ihrem Schreibtisch erscheinen. Es beinhaltet alle relevanten Daten ihres Projekts und kann nun in jede andere ECLIPSE Softwareinstallation importiert werden (z.B. in den Installationen in den Programmierlaboren).

2.4. Aufgabe: Importieren eines ECLIPSE Projekts

Im Rahmen der Übungsbearbeitung werden Sie vielleicht Lösungen von Mitstudierenden interessieren, die Sie in ihre IDE importieren wollen. Vielleicht bereiten Sie auch bereits zu Hause ihre Übungen für die Programmierlabore vor, um gezielt Fragen stellen zu können. Dann ist es für Sie hilfreich, ein Projekt zu Hause vorzubereiten, z.B. in Moodle hochzuladen und in der Übung herunterzuladen und in die ECLIPSE IDE im Programmierlabor zu importieren. Wie dies funktioniert, wird nun erläutert.

Laden Sie zuerst ihr in Moodle hochgeladenes Projekt herunter. Gehen Sie hierzu bitte wie folgt vor:

- Loggen Sie sich in Moodle mit der Kennung ein, die sie mit der Immatrikulation erhalten haben.
- Wählen Sie den Kurs Grundlagen der Programmierung (ITD) oder Programmieren I (Inf)
- Klicken Sie auf die Aufgabe im Block Übungen die sie hochgeladen haben.
- Die Datei die Sie hochgeladen erscheint unter Lösungsentwurf bzw. Aufgabenabgabe. Klicken Sie darauf um diese wieder aus Moodle herunterzuladen. Speichern Sie diese an einem beliebigen Ort, z.B. dem Desktop.

Importieren Sie dieses Projekt nun bitte wieder in ECLIPSE. Gehen Sie hierzu bitte wie folgt vor:

- Starten Sie ECLIPSE (sofern noch nicht geschehen)
- Wählen Sie **File -> Import** (ein Import Dialog Fenster erscheint)
- Wählen Sie im Import Dialog General -> Existing Projects into Workspace anschließend Next. Wählen Sie nun im Dialog
  - **Select Archive File**
  - Anschließend: **Browse (nach Wahl des Projektarchivs) erscheinen alle in diesem Projektarchiv befindlichen Projekte**
  - Selektieren Sie die zu importierenden Projekte (in diesem Fall HelloWorld)
  - Anschließend: Finish (das Projekt wird danach im Projekt Explorer angezeigt)

Die in dieser Übung gezeigten Schritte sind relevant, wenn Sie Übungen abgeben und für eine Bewertung freigeben sollen.

2.5. Aufgabe: Schreiben einer Adressausgabe

Schreiben Sie nun ein Programm und erzeugen Sie ein entsprechendes Projekt, dass ihren Namen und ihre Adresse in folgender Form ausgibt:

```
Maren Musterfrau
Beispielgasse 123
12345 Exemplaria
Vereinigte Staaten von Europa
```

2.6. Aufgabe: Schreiben eines Multiplikationsprogramms

Schreiben Sie nun ein Programm und erzeugen Sie ein entsprechendes Projekt, dass die Zahlen von 1 bis 10 miteinander multipliziert und in folgender Form ausgibt.

```
1 * 2 * 3 * ... * 10 = ???
```

Die Fragezeichen bezeichnen hoffentlich, den durch ihr Programm korrekt berechneten Wert.

Teil II.

Aufgaben zu Unit 2
Imperative Programmierung

3. Eingaben und Ausgaben

3.1. Aufgabe: Frage nach einem Namen

Schreiben Sie nun ein Programm das mit folgender Zeile

```
Bitte geben Sie Ihren Namen ein:
```

nach einem Namen fragt und folgende Antwort ausgibt:

```
Hallo ??? schön Dich zu sehen.
Wie geht es Dir?
```

Die ??? sollen dabei durch den eingegebenen Namen ersetzt werden.

3.2. Aufgabe: Geben Sie Ihren Namen in Großbuchstaben aus

Schreiben Sie nun ein Programm, dass ihren Vornamen in folgender Form ausgibt, wenn Ihr Name JAVA wäre.

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

Hinweis: *Der Name soll nicht über die Tastatur eingelesen werden, sondern kann hardcodiert werden!*

3.3. Aufgabe: Berechnen Sie ein paar Eingaben

Schreiben Sie nun ein Programm, dass nach zwei ganzzahligen Operanden a und b fragt

```
Bitte geben Sie den Operanden a ein:
Bitte geben Sie den Operanden b ein:
```

und für beide Operanden den folgenden Term berechnet

$a + (\frac{b}{a} - \frac{a}{b}) * b$

und dieses in folgender Form ausgibt:

```
Ergebnis: ???
```

Die ??? sollen dabei durch das Berechnungsergebnis ersetzt werden.

Testen Sie ihr Programm mit den folgenden Eingaben:

- Erhalten Sie für die Eingaben $a = 1$ und $b = 3$ auch das Ergebnis 10?
- $a = 5$ und $b = 3$: Welches Ergebnis wird berechnet? Erscheint Ihnen das plausibel?
- $a = -2$ und $b = 0$: Welches Ergebnis wird berechnet? Erscheint Ihnen das plausibel?

3.4. Aufgabe: Cowsay (oder: Wie macht die Kuh?)

Diese Aufgabe stellt eine etwas erweitere Version von HelloWorld dar. Als Vorbild dient das Unix-Programm Cowsay. Diesem Programm übergibt man einen Text und als Ergebnis erhält man eine kleines Bild (als ASCII-Art) auf der Konsole, welches den übergebenen Text enthält. Dies kann bspw. so aussehen:

```
Wie macht die Kuh? Muuh

 ------
< Muuh >
 ------
    \   ^__^
     \  (oo)\_______
        (__)\       )\/\
            ||----w |
            ||     ||
```

Gibt man hingegen "Java rocks!" ein, so soll sich folgende Ausgabe ergeben.

```
Wie macht die Kuh? Java rocks!

 -------------
< Java rocks! >
 -------------
    \   ^__^
     \  (oo)\_______
        (__)\       )\/\
            ||----w |
            ||     ||
```

Die Länge der Sprechblase ist also abhängig von der Eingabe des Nutzers. Ihr Nutzer soll einen beliebigen Text auf der Konsole eingegeben können und eine Ausgabe wie oben gezeigt erhalten.

Da sie noch keine Schleifen kennen gelernt haben, kommt hier der Tipp, das Problem mittels der `String.replaceAll` Methode zu lösen, die ihnen Java standardmäßig anbietet. Lesen sie doch hier nach, welche Methoden Java für `Strings` anbietet (das sind einige). Machen sie sich insbesondere über die genannte Methode schlau.

`https://docs.oracle.com/javase/8/docs/api/java/lang/String.html`

Vielleicht probieren Sie auch mal die folgenden beiden Zeilen Code aus, und versuchen herauszufinden, was diese so machen?

```java
// Dies ist im uebrigen eine studentische Loesung
// von Alenka Rixen (ITD, WS 2016/17),
// die wesentlich einfacher als die urspruengliche Musterloesung ist.
String eingabe = "Nur ein Test ...";
String line = eingabe.replaceAll(".", "-");
```

Wie sie sehen, kann das Internet und Google hilfreich bei der Lösung von Aufgaben sein. Sie müssen aber natürlich immer erklären und verstehen können, was sie da programmieren. Ganz nebenbei haben Sie so die technische Dokumentation von Java genutzt, um ihr Problem zu lösen. Für alle Methoden und Datentypen in Java gibt es eine solche Beschreibung. Google findet diese Seiten im allgemeinen recht zuverlässig. Der Trick sind die folgenden Suchworte:

```
"Java 8 <Datentyp, der mir helfen könnte> <ggf. Methode, die mir helfen könnte>"
```

3.5. Aufgabe: Berechnen Sie das Volumen einer Kugel

Schreiben Sie ein Program, das vom Nutzer den Durchmesser einer Kugel in cm einliest und daraus das Volumen der Kugel berechnet.

```
Bitte geben Sie den Durchmesser einer Kugel ein [cm]: 1
Das Volumen der Kugel beträgt [cm^3]: 0.5235987755982988
```

3.6. !!! NEU !!! Aufgabe: Finden Sie den Format-String

Die Methode `String.format()` nimmt als Parameter einen Format-String und beliebig viele Argumente an. Dabei kann es sich z.B. um weitere Strings, Integerwerte oder Fließkommazahlen handeln, die in den Format-String eingebettet werden sollen. Weitere Informationen zur Formatstrings finden sie unter folgenden Links:

- `https://docs.oracle.com/javase/8/docs/api/java/util/Formatter.html`
- `https://dzone.com/articles/java-string-format-examples`
- `http://www.codejava.net/java-se/file-io/java-string-format-examples`

Werden außer dem Format-String keine weiteren Argumente übergeben, gibt die Methode diesen unverändert zurück:

```
1 String format = "Hello World!";
2 String result = String.format(format);
3 System.out.print(result); //out: Hello World!
```

Noch kürzer schreibt sich das, wenn man die `printf()`-Methode des jeweiligen PrintStreams verwendet:

```
1 System.out.printf("Hello World!"); //out: Hello World!
```

Schauen Sie sich im Internet ein paar Beispiele an, und finden Sie die fehlenden Format-Strings in folgendem Code-Snippet (das '\n' erzeugt einen Zeilenumbruch, sonst würde printf alle Ausgaben direkt hintereinander schreiben):

```
1 // a) Strings einfügen
2 System.out.printf("???\n", "Walter"); // "Hallo Walter!"
3
4 // b) Integerwerte einfügen
5 System.out.printf("???\n", 3);        // "Aller guten Dinge sind 3!"
6
```

```
7  // c) Fließkommazahlen runden
8  System.out.printf("???\n", Math.PI);  // "3.1416: Das Ingenieurs-Pi"
9
10 // d) Fließkommazahlen runden und untereinander ausrichten
11 String format = "???\n";
12 System.out.printf(format, 1234.5678); // "1234.57"
13 System.out.printf(format, 10.0 / 3.0);// "    3.33"
14 System.out.printf(format, 42.);       // "   42.00"
```

4. Datentypen

Für diese Aufgaben in diesem Kapitel benötigen Sie ein paar Informationen, die Sie ggf. **noch** nicht in der Vorlesung gehört haben. Die Ihnen noch nicht bekannten Konzepte werden hier kurz erläutert. Sie sind nicht schwer zu verstehen. Von Ihnen wird aber **noch** nicht verlangt, diese selber produzieren zu können.

In einigen der nachfolgenden Übungen dieses Kapitels müssen Sie Zeichenketten durchlaufen. Entweder Zeichen für Zeichen oder Wort für Wort. Hierzu benötigen Sie sogenannte Kontrollstrukturen, die es Ihnen ermöglichen einzelne Anweisungen mehrmals hintereinander auszuführen (nämlich pro Zeichen oder pro Wort).

Zeichenketten Zeichen für Zeichen durchlaufen

Eine **Zeichenkette `str` können Sie wie folgt Zeichen für Zeichen** durchlaufen:

```
1 String str = "Beispiel";
2 for (char c : str.toCharArray()) {
3   System.out.println(c);
4 }
```

Diese sogenannte `for` Schleife können Sie in etwa wie folgt lesen: **Führe für jedes Zeichen `c` in der Zeichenkette `str` die Anweisungen im folgenden `for` Block aus.** Oben stehende Anweisungen erzeugen folgende Ausgabe:

```
B
e
i
s
p
i
e
l
```

Zeichenketten Wort für Wort durchlaufen

Eine **Zeichenkette `str` können Sie aber auch Wort für Wort** durchlaufen:

```
1 String str = "Dies ist ein Beispiel";
2 for (String w : str.split(" +")) {
3   System.out.println(w);
4 }
```

Diese `for` Schleife können Sie in etwa wie folgt lesen: **Führe für jedes Wort (das durch mindestens ein Leerzeichen getrennt ist) die Anweisungen im folgenden `for` Block aus.** Oben stehende Anweisungen erzeugen dann folgende Ausgabe.

```
Dies
ist
ein
Beispiel
```

Anweisungen nur unter einer Bedingung ausführen

Zudem müssen Sie zur Lösung der nachfolgenden Aufgaben in einigen Fällen einige Anweisungen nur ausführen, wenn eine gewisse Bedingung erfüllt ist. Hierfür sehen Programmiersprachen sogenannte **bedingte Anweisungen** vor. Die entsprechende Kontrollstruktur nennt man `if` Anweisung. Sie prüft eine Bedingung - ist diese erfüllt, dann wird ein zugehöriger `if` Block ausgeführt, andernfalls nicht. Ein Beispiel sehen sie nachfolgend:

```
1 int a = 10;
2 if (a > 5) {
3   System.out.println("Der Wert ist größer als fünf.");
4 }
```

Oben stehende Anweisungen erzeugen die folgende Ausgabe

```
Der Wert ist größer als fünf.
```

da die Variable `a` mit dem Wert 10 belegt wurde und die entsprechende `if` Bedingung also erfüllt ist (zehn ist größer als fünf). Nachfolgende Anweisungen würden keine Ausgabe produzieren.

```
1 int a = 4;
2 if (a > 5) {
3   System.out.println("Der Wert ist kleiner als fünf.");
4 }
```

Die Online Dokumentation von JAVA nutzen

Zudem werden Sie zur Verarbeitung von Zeichenketten einige Operationen benötigen, die nicht in der Vorlesung explizit angesprochen worden. Sie sollen auch lernen, die Dokumentation eines sogenannten **Application Programming Interface (API)** zur Lösung ihrer Problemstellungen heranzuziehen. Der Datentyp `String` bietet eine Vielzahl von sogenannten Methoden an, die Sie zur Lösung Ihrer Probleme nutzen können (z.B. alle Zeichen einer Zeichenkette in Großbuchstaben wandeln, bestimmen einer Teilzeichenkette, Zerlegen einer Zeichenkette in Wörter oder Zeichen, etc.).

Alle Methodenbeschreibungen zum Datentyp `String` finden Sie in der Online Dokumentation zu JAVA unter folgendem Link:

```
http://docs.oracle.com/javase/7/docs/api/java/lang/String.html
```

Im übrigen findet sich zu jedem Datentyp von JAVA eine derartige Beschreibung in der API Dokumentation. Wenn Sie auf der Suche nach Antworten auf Detailfragen sind oder sie Funktionalitäten suchen, die ggf. schon implementiert sein könnten, ist die API Dokumentation von JAVA ein hervorragendes Nachschlagewerk. Als Lehr- und Ausbildungswerk ist es allerdings weniger geeignet.

Programmieren Sie algorithmische Beschreibungen

Insbesondere für die Übungsaufgaben 4.2, 4.3 und 4.4 gilt die Empfehlung, dass Sie erst versuchen sollten, das Problem algorithmisch zu beschreiben und dann erst die Implementierung in JAVA vornehmen. Lösen Sie also erst einmal das Problem an sich, bevor sie es versuchen dem Rechner in einer konkreten Programmiersprache "beizubringen".

4.1. Aufgabe: Deklarieren Sie möglichst speicherschonend Variablen

Sie sollen verschiedene Variablen in einem Programm deklarieren. Finden Sie den passenden (möglichst speicherplatzsparenden) Typ für eine Variable, die angibt,

- was Ihre Initialen sind,
- wie viele Menschen in Deutschland leben,
- wie viele Menschen auf der Erde leben,
- ob es gerade Tag ist,
- wie hoch die Trefferquote von Mario Gomez ist,
- wie viele Semester Sie studieren zu beabsichtigen und
- wie viele Studierende sich für den Studiengang Informationstechnologie und Gestaltung angemeldet haben.

Deklarieren Sie die Variablen und verwenden Sie sinnvolle Bezeichner. Verketten Sie alle Variablen mit dem + Operator und geben Sie das Ergebnis mit einer `System.out.println` Anweisung aus.

Ein typischer Programmablauf könnte wie folgt aussehen:

```
Wie lauten Ihre Initialen? NK
Wie viele Menschen leben in Deutschland? 80000000
Wie viele Menschen leben auf der Erde? 6775235700
Ist es gerade Tag? true
Wie hoch ist die Trefferquote von Mario Gomez? 0,23
Wie viele Semester beabsichtigen Sie zu studieren? 6
Wie viele Studierende haben sich in diesem Jahr immatrikuliert? 1000
Dies ergibt:
NK8000000067752357000true0.2361000
```

Hinweis: *Bei dieser Aufgabe sollten Sie jeweils eine byte, short, boolean, int, long, String und float Variable nutzen. Dies ergibt einen minimalen Speicherbedarf.*

4.2. Aufgabe: Spiegeln Sie Wörter

Sie sollen nun ein Programm entwickeln, dass eine Zeichenkette von einem Nutzer einliest und diese Zeichenkette in gespiegelter Form ausgibt. Aus der Zeichenkette Beispiel soll z.B. die Zeichenkette

```
BeispielleipsieB
```

werden.

Ein typischer Programmablauf könnte wie folgt aussehen:

```
Bitte geben Sie die zu spiegelnde Zeichenkette ein: Hello World
Hello WorlddlroW olleH
```

4.3. Aufgabe: Bestimmen Sie Buchstabenhäufigkeiten

Sie sollen nun ein Programm entwickeln, dass die Häufigkeit der Buchstaben **a, e, f, r, q und z** in einer durch den Nutzer einzugebenden Zeichenkette in absoluter und relativer Form bestimmt und ausgibt. Es sollen nur die Buchstabenhäufigkeiten ausgegeben werden, die auch tatsächlich in der zu untersuchenden Zeichenkette vorkommen.

Die Buchstabenhäufigkeit der Zeichenkette

```
Ein Zebra
```

ist beispielsweise:

```
a: 1 (11,1111%)
e: 2 (22,2222%)
r: 1 (11,1111%)
z: 1 (11,1111%)
```

Die absolute Häufigkeit eines Buchstabens ist die Anzahl der Vorkommen des Buchstabens in der Zeichenkette. Dabei sind sowohl die Groß- als auch die Kleinschreibweise eines Buchstabens zu berücksichtigen (also sowohl 'a' als auch 'A' zu zählen). Die relative Häufigkeit ergibt sich aus dem Verhältnis der absoluten Häufigkeit zur Länge der Zeichenkette in Zeichen. Bitte geben Sie die relative Häufigkeit in Prozent an.

Ein typischer Programmablauf könnte beispielsweise so aussehen:

```
Bitte geben Sie eine auszuwertende Zeichenkette ein:
Am Fluss springen die Affen quer über die Zebras.
a: ? Vorkommen (??.?????%)
e: ? Vorkommen (??.?????%)
f: ? Vorkommen (??.?????%)
r: ? Vorkommen (??.?????%)
q: ? Vorkommen (??.?????%)
z: ? Vorkommen (??.?????%)
```

Wie lauten die durch Sie berechneten Werte für die folgende Zeichenkette?

```
Am Fluss springen die Affen quer über die Zebras.
```

4.4. Aufgabe: Ein kleiner Wordprocessor

Sie sollen nun ein Programm schreiben, dass eine Zeichenkette vom Nutzer einliest, welches einzelne Wörter in dieser Zeichenkette in Abhängigkeit eines Formatierungssymbols komplett in Groß- oder Kleinbuchstaben wandelt.

Die Formatierung funktioniert dabei nach folgendem Prinzip.

- Wird einem Wort ein _ als Formatierungszeichen **vorangestellt**, so soll dieses Wort komplett **groß** geschrieben werden.

- Wird einem Wort ein _ als Formatierungszeichen **angehängt**, so soll dieses Wort komplett **klein** geschrieben werden.
- Ist einem Wort sowohl ein _ vorangestellt als auch angehängt (z.B. '_Hello_'), so gilt das letzte Formatierungszeichen als ausschlaggebend für die Groß- und Kleinschreibung des Wortes (aus '_HellO_' wird also bspw. 'hello').
- Hat ein Wort kein Formatierungszeichen, bleibt es in der Schreibweise bestehen.
- Das Formatierungszeichen _ wird niemals mit ausgegeben (ein _ das in der Mitte eines Wortes steht, gilt allerdings nicht als Formatierungszeichen, z.B. in 'Hello_World').
- Ein Wort ist von einem anderen Wort durch mindestens ein Leerzeichen getrennt (Satz- und Sonderzeichen gelten als normale einem Wort zugeordnete Zeichen, bspw. wird also 'Hello_!!!! World???!' in die Worte 'Hello_!!!' und 'World???!' zerlegt).

Bspw. wird die Zeichenkette

```
meiN_ _Name iSt _Hase. Ich weiß BESCHEID._ _wirklich!!!
```

also wie folgt ausgegeben:

```
mein NAME iSt HASE. Ich weiß bescheid. WIRKLICH!!!
```

Ein typischer Programmablauf könnte wie folgt aussehen:

```
Bitte geben Sie eine zu formatierende Zeichenkette ein:
meiN_ _Name iSt _Hase. Ich weiß BESCHEID._ _wirklich!!!
mein NAME iSt HASE. Ich weiß bescheid. WIRKLICH!!!
```

4.5. Aufgabe: Glücksspiel

Der Rechner soll sich nun eine geheime Zahl zwischen 1 und 100 ausdenken. Diese Zahl können sie beispielsweise mit folgender Zeile bestimmen.

```
1 int geheim = (int)(Math.random() * 100) + 1;
```

Anschließend sollen zwei Spieler ihre Zahlen eingeben können.

Der Rechner soll denjenigen Spieler bestimmen, der näher an der geheimen Zahl liegt.

Ein typischer Spielablauf könnte wie folgt aussehen:

```
Ich denke mir eine Zahl zwischen 1 und 100 aus.
Wer von Euch beiden näher rät, hat gewonnen.
Spieler 1, bitte gib Deine Zahl ein:  23
Spieler 2, bitte gib Deine Zahl ein:  78
Vielen Dank.
Meine Zahl lautete: 98
Spieler 2 hat gewonnen.
```

4.6. !!! NEU !!! Aufgabe: Letzte Ziffer

Schreiben Sie ein Programm, dass zwei ganze Zahlen von der Konsolte einliest und prüft ob die letzte Ziffer beider Zahlen übereinstimmt.

- 156, 36 => letzte Ziffer stimmt überein
- 156, 55 => letzte Ziffer stimmt nicht überein

Ein typischer Programmablauf könnte so aussehen:

```
Bitte geben Sie eine ganze Zahl ein? -1256
Bitte geben Sie eine weitere ganze Zahl ein? 65
Die letzten Ziffern stimmen nicht überein.
```

4.7. !!! NEU !!! Zeichenketten rotieren

Schreiben Sie ein Programm, dass eine Zeichenkette von der Konsole einliest sowie eine ganze positive Zahl ,um dessen die Zeichenkette "rotiert" werden soll.

Die "Rotation" soll wie folgt funktionieren.

- "Hallo" um 0 rotieren bleibt "Hallo"
- "Hallo" um 1 rotieren wird zu "alloH"
- "Hallo" um 2 rotieren wird zu "lloHa"
- ...
- "Hallo" um 5 rotieren wird zu "Hallo"

Ihr Programm soll negative Rotationen oder Rotationen, die länger sind als die eingegebene Zeichenkette, nicht durchführen. In diesem Fällen soll das Program mit der Ausgabe

"Nicht möglich"

abbrechen.

Ein typischer Programmablauf könnte so aussehen:

```
Bitte geben Sie eine Zeichenkette ein: Hallo Welt
Bitte geben Sie eine Rotation ein? 4
Ausgabe: o WeltHall
```

5. Berechnungen und Logik

5.1. Aufgabe: Sekunden umrechnen

Schreiben Sie ein Programm, das eine vorgegebene Zahl von Sekunden in Jahre, Tage, Stunden, Minuten und Sekunden zerlegt. Das Programm soll z.B. für einen Sekundenwert 158036522 folgendes ausgeben:

```
Welchen Sekundenwert wollen Sie zerlegen? 158036522
158036522 Sekunden entsprechen:
5 Jahren,
4 Tagen,
3 Stunden,
2 Minuten und
2 Sekunden.
```

Der zu zerlegende Sekundenwert soll vom Nutzer eingelesen werden.

Hinweis: Sie müssen keine Schaltjahre berücksichtigen und können daher mit 365 Tagen für ein Jahr rechnen.

5.2. Aufgabe: Dosenberechnung

Schreiben Sie nun ein Programm für Berechnungen, die für die Herstellung von Konservendosen aus einem Blechstück mit

- Umfang u (Umfang der Dose in Zentimetern) und
- Höhe h (Höhe der Dose in Zentimetern)

anfallen. Ausgehend von den durch einen Nutzer eingegebenen Werten für u und h und unter Verwendung der Konstante $\pi = 3.141592$ (bzw. `Math.PI`) die folgenden Werte berechnet und ausgibt:

- den Durchmesser des Dosenbodens: $d_{boden} = \frac{u}{\pi}$,
- die Fläche des Dosenbodens: $f_{boden} = \pi(\frac{d_{boden}}{2})^2$,
- die Mantelfläche der Dose: $f_{mantel} = uh$,
- die Gesamtfläche der Dose: $f_{gesamt} = 2f_{boden} + f_{mantel}$,
- das Volumen der Dose: $v = f_{boden}h$

5.3. Aufgabe: Wann ist eine Stadt eine Metropole?

Gegeben seien die Variablen, die Auskunft über Eigenschaften einer bestimmten Stadt geben:

```
String name;
boolean istHauptstadt;
int anzahlEinwohner;
double steuernProEinwohner;
```

Dabei gilt folgendes:

- `name` bezeichnet den Namen der Stadt
- `istHauptstadt` ist genau dann `true`, wenn die Stadt eine Hauptstadt ist.
- `anzahlEinwohner` gibt die Anzahl der Einwohner der Stadt an.
- `steuernProEinwohner` ist die durchschnittliche Steuerabgabe pro Einwohner und Jahr.

Wir definieren eine Metropole als eine Stadt, die Hauptstadt ist mit mehr als 100 000 Einwohner oder die mehr als 200 000 Einwohner hat und im Jahr mindestens 1 000 000 000 Euro Steuereinnahmen hat.

Schreiben Sie ein Programm, dass die Eigenschaften einer Stadt abfragt und sagt, ob diese Stadt eine Metropole ist oder nicht.

```
Name der Stadt? Lübeck
Einwohner? 210000
Hauptstadt (true oder false)? false
Steuern pro Einwohner im Jahr? 3000
Lübeck ist keine Metropole

Name der Stadt? Hamburg
Einwohner? 1700000
Hauptstadt (true oder false)? true
Steuern pro Einwohner im Jahr? 4000
Hamburg ist eine Metropole
```

Um welchen Betrag müsste Lübeck seine Steuereinnahmen pro Jahr und Einwohner erhöhen, um zur Metropole zu werden?

5.4. Aufgabe: Zeilen sparen

Gegeben ist folgende Berechnung.

```
1 double x = 15.5;
2 double a = Math.sqrt(3.5 + x);
3 double b = a * 5;
4 double c = b / 3;
5 double d = x + 10;
6 double e = x - 4.1;
7 double f = d * e;
8 double g = c - f;
9 System.out.println(g);
```

Drücken Sie diese Berechnung in einem Ausdruck so aus, dass Sie dasselbe Ergebnis in einer Zeile berechnen können.

5.5. Gestrichen

Dies Aufgabe wurde im WS 2016/17 gestrichen.

5.6. Aufgabe: Schaltjahr

Sie sollen nun ein Programm zur Prüfung von Schaltjahren (Gregorianischer Kalender) entwickeln. Ein typischer Programmablauf kann wie folgt aussehen:

```
Bitte geben sie eine beliebige Jahreszahl ein: 2016
2016 ist ein Schaltjahr.
```

Wikipedia hilft ihnen weiter, wenn Sie nicht wissen, wie man ein Schaltjahr im Gregorianischen Kalender bestimmt.

`https://de.wikipedia.org/wiki/Schaltjahr`

5.7. !!! NEU !!! Aufgabe: Narzistische Armstrong Zahlen

Ja, so etwas gibt es! Eine Armstrong Zahl ist eine Zahl, deren Summe ihrer Ziffern, jeweils potenziert mit der Stellenanzahl der Zahl, wieder die Zahl selbst ergibt (vgl. `https://de.wikipedia.org/wiki/Narzisstische_Zahl`).

Bspw. sind die folgenden Zahlen Armstrong Zahlen

- $153 = 1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$
- $1634 = 1^4 + 6^4 + 3^4 + 4^4 = 1 + 1296 + 81 + 256 = 1634$
- $54748 = 5^5 + 4^5 + 7^5 + 4^5 + 8^5 = 3125 + 1024 + 16807 + 1024 + 32768 = 54748$

Schreiben Sie nun bitte ein Programm, das für eine beliebige Eingabe prüft, ob diese eine Armstrong-Zahl ist.

Hinweis: *Aus einer Zeichenkette können sie mittels der `Integer.parseInt()`-Methode wie folgt eine Zahl machen.*

```
1 int x = Integer.parseInt("5"); // x == 5
```

6. Ablaufsteuerung

6.1. Aufgabe: Weihnachtsbaum

Schreiben Sie nun ein Programm, das einen Weihnachtsbaum mit Hilfe von `for` Schleifen zeichnet. Lesen Sie die gewünschte Höhe des Baums von der Tastatur ein und geben Sie entsprechend einen Baum wie im folgendem Beispiel aus:

```
Anzahl der Zeilen: 5
    *
   ***
  *****
 *******
*********
    I
```

6.2. Aufgabe: Zahlenraten

Programmieren Sie nun eine Zahlenraten-Spiel. Im ersten Schritt soll der Spieler begrüßt werden und kurz über die Regeln des Spiels informiert werden. Danach soll durch die Anweisung

```
1 int geheimzahl = (int)(Math.random() * 100) + 1;
```

eine Zufallszahl zwischen 1 und 100 durch das Programm generiert werden. Der Spieler soll nun versuchen diese Zahl zu erraten. Programmieren Sie dazu eine Schleife, in der in jedem Durchlauf jeweils

- darüber informiert wird, um den wievielten Rateversuch es sich handelt,
- ein Rateversuch eingegeben werden kann und
- darüber informiert wird, ob die geratene Zahl zu groß, zu klein oder korrekt geraten ist.

Diese Schleife soll so lange durchlaufen werden, bis die Zahl erraten ist. Nachfolgend ein Programm-Ablauf-Beispiel zum besseren Verständnis:

```
Willkommen beim Zahlenraten.
Ich denke mir eine Zahl zwischen 1 und 100. Rate diese Zahl!
1. Versuch: 50
Meine Zahl ist kleiner!
2. Versuch: 25
Meine Zahl ist kleiner!
3. Versuch: 12
Du hast meine Zahl beim 3. Versuch erraten!
```

6.3. Aufgabe: Mehrere Dreiecke ausgeben

Programmieren Sie nun ein Programm, dass mehrere Dreiecke ausgibt. Das Programm soll sie als erstes Fragen wieviele Dreieecke sie ausgeben wollen. Anschließend soll gefragt werden, wie hoch jedes Dreieck sein soll.

Nachfolgend ein Programm-Ablauf-Beispiel zum besseren Verständnis:

```
Wieviele Dreiecke wollen Sie ausgeben? 2
Wie hoch soll jedes Dreieck sein? 3

*
**
***

*
**
***
```

6.4. Aufgabe: Hat Gauß einen Fehler gemacht?

Dem Mathematiker Gauß wird nachgesagt, dass er in der Schule schnell gelangweilt war und den Unterricht störte. Eines Tages soll ihm ein Lehrer die Aufgabe gegeben haben, die Zahlen von 1 bis 100 zu addieren. Der kleine Gauß antwortete 5050 - und zwar auf der Stelle. Der Lehrer war verblüfft und wiederholte die Aufgabe mit mehreren anderen Zahlen.

Das Gauß diese Aufgabe so schnell beantworten konnte, hängt angeblich an folgender mathematischen Gleichung.

$\sum_{i \in 1..n} i = 1 + 2 + 3 + ... + n = \frac{n*(n+1)}{2}$

Testen Sie mit einem Programm, ob diese Gleichung tatsächlich auch für andere Werte als 100 gilt - immerhin wird durch zwei geteilt, also sollten doch ein paar Ergebnisse nicht ganzzahlig sein - oder hat 51 / 2 etwa ein ganzzahliges Ergebnis?

Dieses Programm soll für eine eingegebene Zahl

- einmal die Summe als Summe der Werte von 1 bis n, also $1 + 2 + ... + n$
- und dann anhand der Formel $\frac{n(n+1)}{2}$ berechnen
- die Werte miteinander vergleichen und beide Ergebnisse in folgender Form ausgeben.

```
Welche Zahl wollen Sie prüfen: 100
1 + 2 + ... + 100 = 5050
100 * (100 + 1) / 2 = 5050
Das Ergebnis ist für 100 gleich!
```

Finden Sie eine Zahl mit der Sie Gauß widerlegen können?

6.5. Aufgabe: Ausgabe des Ein-Mal-Eins

Schreiben Sie ein Programm, das das Ein-Mal-Eins berechnet und tabellarisch auf dem Bildschirm ausgibt. Um die auszugebenen Zahlwerte geeignet einzurücken, sollten Sie bei der Ausgabe den Tabulator `"\t"` verwenden. Es empfiehlt sich der Einsatz geschachtelter `for` Schleifen.

Beim Ein-Mal-Eins werden alle Produkte $i * j$ mit $0 < i \leq 10$ und $0 < j \leq 10$ gebildet. Die Ausgabe sollte also etwa wie folgt aussehen:

```
1    2    3    4    5    6    7    8    9    10
2    4    6    8    10   12   14   16   18   20
3    6    9    12   15   18   21   24   27   30
4    8    12   16   20   24   28   32   36   40
5    10   15   20   25   30   35   40   45   50
6    12   18   24   30   36   42   48   54   60
7    14   21   28   35   42   49   56   63   70
8    16   24   32   40   48   56   64   72   80
9    18   27   36   45   54   63   72   81   90
10   20   30   40   50   60   70   80   90   100
```

6.6. Aufgabe: Wurzelziehen wie Newton

Implementieren Sie die Wurzelberechnung mit dem Newton-Verfahren. Sie sollen beliebige vom Nutzer eingegebene Werte berechnen können. Beachten Sie, dass aus negativen Werten per Definition keine Wurzel gezogen werden kann (komplexe Zahlen lassen wir hier mal außen vor)!

Mit dem Newton-Verfahren nähert man sich der Wurzel von x mithilfe eine Startwerts z durch Wiederholen der folgenden Berechnung an:

$z' = z - \frac{z^2 - x}{2z}$

z' soll hier den neuen Wert für z bezeichnen, der sich aus einer Berechnung ergibt.

Sie finden auf Wikipedia weitere Informationen zum Newton-Verfahren:

- `https://de.wikipedia.org/wiki/Newton-Verfahren` (Weitere Anwendungsbeispiele: Berechnung der Quadratwurzel)
- `https://de.wikipedia.org/wiki/Heron-Verfahren` (Heron-Verfahren)

Implementieren Sie bitte als ersten Schritt, dass die Berechnung 10 mal wiederholt wird und bestimmen Sie, wie nahe Sie an das richtige Ergebnis für diverse Werte (1, 2, 3, ...) für z kommen.

In einem nächsten Schritt lösen Sie das Problem mit einer Schleife. Beenden Sie die Schleife, sobald sich das Ergebnis nicht mehr (oder nur noch um ein sehr kleines Delta, z.B. 0.0000001) ändert (d.h. $|z' - z| < \delta$). Benötigen Sie jetzt mehr oder weniger Iterationen? Wie nahe sind Sie mit ihrer Lösung der Java Standardmethode zum Wurzelziehen `Math.sqrt()` gekommen?

Ein typischer Programmablauf könnte so aussehen:

```
Geben Sie bitte eine Zahl x ein: 2
Die Wurzel von 2 lautet: 1.4142135623730951
Ich musste hierzu 5 Iterationen durchführen.
Java behauptet die Wurzel von 2.0 sei 1.4142135623730951.
```

6.7. !!! NEU !!! Aufgabe: Kisten stapeln

Wir wollen nun Kisten stapeln. Es gibt

- große Kisten der Höhe 5 und

- kleine Kisten der Höhe 1.

Sie sollen nun ein Programm schreiben, dass es ermöglicht einzugeben wieviele

- große und kleine Kisten es geben soll
- und wie hoch der Kistenstapel sein soll.

Das Programm soll beantworten, ob das Problem lösbar und mit welchen Kisten.

Beispiele:

- Mit 3 kleinen Kisten und einer großen Kiste kann ein Kistenstapel der Höhe 8 gebaut werden.
- Mit 3 kleinen Kisten und einer großen Kiste kann kein Kistenstapel der Höhe 9 gebaut werden.
- Mit 3 kleinen Kisten und zwei großen Kiste kann ein Kistenstapel der Höhe 10 gebaut werden.
- **Achtung:** Mit 3 kleinen Kisten und zwei großen Kiste kann kein Kistenstapel der Höhe 9 gebaut werden (obwohl alle Kisten zusammen die Höhe 13 ergeben).

Ein typischer Programmablauf könnte so aussehen:

```
Geben Sie bitte die Anzahl kleiner Kisten ein: 2
Geben Sie bitte die Anzahl großer Kisten ein: 2
Geben Sie bitte die Höhe des Zielstapels ein: 11
Ja, ein Stapel der Höhe 11 kann aus 2 großen und 1 kleinen Kisten gebaut werden.
```

7. Funktionalitäten kapseln

7.1. Aufgabe: Ausgabe einer Multiplikationstabelle

Diese Aufgabe ist eine parametrisierte Weiterentwicklung der Aufgabe 6.5. Schreiben Sie nun ein Programm, dass zwei ganze Zahlen abfragt und aus diesen eine Multiplikationstabelle erstellt. Die erste der beiden Zahlen, darf dabei nicht größer als 15 und nicht kleiner als null sein, die zweite nicht kleiner als null.

Dies soll vor Aufruf einer zu definierenden Methode zur Berechnung der Multiplikationstabelle abgefragt werden. Wird gegen eine dieser beiden Bedingungen verstoßen, soll das Programm mit einer Fehlermeldung terminieren.

Nachfolgend ein beispielhafter Programmablauf für korrekte Eingaben.

```
Geben Sie bitte eine Zahl i ein: 13
Geben Sie bitte eine Zahl j ein: 3
Die Multiplikationstabelle lautet:
1  2  3  4  5  6  7  8  9  10 11 12 13
2  4  6  8  10 12 14 16 18 20 22 24 26
3  6  9  12 15 18 21 24 27 30 33 36 39
```

Die Multiplikationstabelle soll durch folgendes Statement ausgegeben werden:

```
1 System.out.println(mtab(i, j));
```

Sie müssen also die in Aufgabe 6.5 implementierte Funktionalität in einer Methode `mtab` kapseln und aufrufen.

7.2. Aufgabe: Volumen- und Gesamtflächenberechnung einer Dose

Diese Aufgabe ist eine parametrisierte Weiterentwicklung der Aufgabe 5.2. Sie sollen nun zwei Methoden schreiben.

- Die Methode `dosenVolumen` soll das Volumen einer Dose berechnen, wenn Umfang und Höhe der Dose in cm bekannt sind.
- Die Methode `dosenOberflaeche` soll die Gesamtoberfläche einer Dose berechnen, wenn ebenfalls Umfang und Höhe der Dose in cm bekannt sind.

Ansonsten soll das zu entwicklende Programm so funktionieren, wie bereits in Aufgabe 5.2, allerdings sollen nur Dosenvolumen und Dosenoberfläche berechnet und ausgegeben werden.

7.3. Aufgabe: Passwortgenerator

Damit man sich Passwörter gut merken kann, ohne das diese sich leicht erraten lassen, kann man Passwörter aus Merksätzen bilden. Wir wollen nach folgenden Regeln Passwörter aus Sätzen bilden:

- Das Passwort wird als Folge der **ersten Zeichen** eines jeden Wortes im Satz gebildet.
- Wörter des Satzes sollen alle durch beliebig viele Leerzeichen abgetrennten Teilfolgen sein.
- Werden Satzzeichen direkt hinter ein Wort geschrieben, so gehören diese zu dem Wort.
- Das Passwort soll durch eine Zahl abgeschlossen werden, die sich aus der Anzahl der Einzelwörter des Satzes ergibt.

Beispiel: "Wenn es regnet, dann wird es nass ." besteht aus den Teilwörtern "Wenn" "es" "regnet," "dann" "wird" "es" "nass" "." und das dazugehörige Passwort ist "Werdwen.8"

Implementieren Sie nun eine Methode `pwdgen`, die zu einem Satz (`String`) ein Passwort nach obigen Regeln zurückgibt (`String`). Die Methode `pwdgen()` soll beliebige Strings (ungleich `null`) verarbeiten und daraus Passwörter gem. oben stehenden Bildungsregeln erzeugen.

Verwenden Sie die Methode `split()` von `String`, um die Teilwörter zu erzeugen. Beachten Sie auch die anderen Methoden von `String`, die Ihnen bei dieser Aufgabe behilflich sein könnten, sehen Sie hierzu z.B. in der API Dokumentation von Java nach:

`http://download.oracle.com/javase/7/docs/api/java/lang/String.html`

Sie können übrigens mittels folgender Zeilen eine Zeichenkette in ihre durch Leerzeichen separierten Teilwörter zerlegen und Wort für Wort durchlaufen:

```
1 String s = "Dies ist ein Beispiel.";
2 for (String w : s.split(" +")) {
3   System.out.println(w);
4 }
```

Obiger Quelltext erzeugt folgende Konsolenausgabe:

```
Dies
ist
ein
Beispiel
```

Testen Sie ihr Programm mit den folgenden Eingaben:

- *"Mein Fahrrad fährt Tango."* (ergibt `MFfT4`)
- *"Ein Passwort ist nur so gut , wie sein geheimer Platz ."* (ergibt `EPinsg,wsgP.12`)
- *"Selber programmieren ist überraschend !"* (`Spiü!5`)

7.4. Aufgabe: Kommentieren mit Javadoc

Beim Programmieren geht es nicht darum, eine Funktionalität zu implementieren, sondern diese auch nachvollziehbar und pragmatisch zu implementieren. "JavaDoc" ist hierzu im Java Umfeld ein häufig anzutreffendes Tool. Auch die kontextsensitiven Hilfen der Softwareentwicklungsumgebung Eclipse werden aus speziellen Javadoc Kommentaren erzeugt.

Recherchieren sie im Internet daher zum Schlagwort "JavaDoc". Als Einstieg seien Ihnen die folgenden Links empfohlen.

- Javadoc auf Wikipedia (`http://de.wikipedia.org/wiki/Javadoc`)
- Writing Javadoc Comments in Eclipse (`http://www.youtube.com/watch?v=6XoVf4x-tag`)
- Eclipse Javadoc Tools (`http://www.youtube.com/watch?v=G1cbk3ch7y0`)
- How to Write Doc Comments for the Javadoc Tool (`http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html`)

Ihre Aufgabe ist es nun, alle in dieser Unit erstellten Methoden mittels aussagekräftiger Javadoc Kommentare zu dokumentieren.

Nutzen Sie insbesondere die folgenden Tags

- `@author` um den/die Autor/in einer Methode/Klasse zu beschreiben,
- `@param` um Parameter einer Methode zu beschreiben,
- `@return` um die Rückgabe einer Methode zu beschreiben.

Im weiteren Verlauf sind Abnahmeaufgaben grundsätzlich mit Javadoc Kommentaren zu versehen. Liegen diese nicht vor, wird es Punktabzüge geben. Dies gilt insbesondere für die Aufgaben 7.1, 7.2 und 7.3 und alle folgenden.

Wesentlich bei einer gut kommentierten Methode ist dabei, dass sie das "Was?" (macht die Methode) beschreiben, nicht das "Wie?" (macht die Methode etwas).

1. Sie sollten in einem bis zwei Sätzen beschreiben, was die Methode macht, nicht wie sie es macht (das steht im Code, das kann jeder selber nachvollziehen).
2. Sie sollten jeden Parameter mittels @param erläutern (nicht den Datentyp, das steht im Code, das kann jeder selber nachvollziehen). Sie sollten ferner angeben, ob für den Parameter Einschränlungen existieren, die die Methode nicht verarbeiten kann.
3. Sie sollten die Rückgabe mittels @return erläutern. Erläuterungen wie "Liefert eine Zahl" sind dabei nicht hilfreich. Erklären sie lieber wie die Zahl berechnet wird (machen sie dies ggf. an einem Beispiel deutlich.)

Sie sollten also wie folgt kommentieren:

```
1  /**
2    * Bestimmt die Haeufigkeit der Vorkommen des Zeichens c
3    * in einer Zeichenkette input.
4    * Die Methode arbeitet case sensitive.
5    * @param input Zu untersuchende Zeichenkette (ungleich null).
6    * @param c Zu zaehlendes Zeichen.
7    * @return Anzahl der Zeichen c in der Zeichenkette input
8    *         (z.B. 1 fuer 'e' in "Hello" oder 0 fuer 'X' in "World").
9    */
10 public static int countChar(String input, char c) {
11   int i = 0;
12   for (char current : input.toCharArray()) {
13     i = current == c ? i + 1 : i;
14   }
15   return i;
16 }
```

und nicht so:

```
/**
  * Durchlaeuft eine Zeichenkette mittels einer foreach
  * Schleife Zeichen fuer Zeichen. Inkrementiert einen Zaehler
  * i immer dann, wenn ein Zeichen gefunden wurde.
  * @param input Eingabe-Zeichenkette
  * @param c Zeichen
  * @return ganze Zahl
  */
public static int countChar(String input, char c) {
  int i = 0;
  for (char current : input.toCharArray()) {
    i = current == c ? i + 1 : i;
  }
  return i;
}
```

7.5. Aufgabe: Hangman

Implementieren Sie nun bitte das Spiel Hangman.

`https://de.wikipedia.org/wiki/Galgenm%C3%A4nnchen`

Sie können Hangman online spielen, um sich mit dem Spielprinzip vertraut zu machen.

`http://hangman24.de/`

Sie sollen eine Konsolenvariante des Spiels entwickeln. Das Spiel beginnt mit der Darstellung eines leeren Galgens.

```
  +-----+
  |/
  |
  |
  |
 ***
***********

Suchwort: _ _ _ _
```

Bei Hangman wird ihnen zufällig ein Suchwort vorgeschlagen. Sie können nun Buchstaben raten. Haben Sie den richtigen Buchstaben geraten, werden im Suchwort alle Vorkommen dieses Buchstabens eingeblendet. Haben Sie einen nicht im Suchwort vorkommenden Buchstaben eingeben, so wird ihr Fehlerzähler um eins nach oben gezählt und ein Hangman ausgegeben.

```
  +-----+
  |/    O
  |
  |
  |
 ***
***********
```

```
8
9 Suchwort: R _ _ e
```

Mit jedem weiteren Fehler wird ihr Hangman vollständiger (erst Kopf, dann Rumpf, dann linker Arm, dann rechter Arm, dann linkes Bein, dann rechtes Bein).

```
1   +-----+
2   |/    O
3   |   --+--
4   |     |
5   |    / \
6  ***
7 ************
8
9 Suchwort: R a _ e
```

Das Spiel ist beendet, wenn der Hangman vollständig ist und sie das Suchwort nicht entschlüsseln konnten (d.h. Sie haben verloren). Das Spiel ist ebenfalls beendet, wenn der Hangman nicht vollständig ist und Sie das Suchwort in weniger als sechs Schritten enträtseln konnten (d.h. Sie haben gewonnen).

Versuchen Sie ihre Lösung sinnvoll in Methoden zu gliedern, um dieses Problem zu lösen. Ansonsten werden Sie schnell verzweifeln.

7.6. Aufgabe: Verschlüsseln mit der Caesar Chiffre

Die Caesar-Verschlüsselung ist ein einfaches symmetrisches Verschlüsselungsverfahren, das zwar heutzutage nicht mehr als sicher gelten kann, aber ein gutes Anschauungsobjekt ist, ein Problem in Teilprobleme zu gliedern, d.h. mittels mehreren Methoden umzusetzen.

Das Prinzip der Caesar-Verschluesselung ist, dass Buchstaben des Klartextes durch Buchstaben, die um n Stellen im Alphabet nach links oder rechts verschoben sind, ersetzt werden. Dieses Schieben um n Stellen ist der Schlüssel[1]. Schiebt man nach links, entschlüsselt man. Schiebt man nach rechts, verschlüsselt man (oder umgekehrt). Wie die Caesar-Verschlüsselung im Detail funktioniert, können sie bei Wikipedia in Erfahrung bringen:

`https://de.wikipedia.org/wiki/Caesar-Verschl%C3%BCsselung`

Analysieren sie dieses Verfahren und setzen sie es in Code um. Achten Sie dabei darauf, dass sie ihren Code in sinnvolle Teilaufgaben mittels Methoden gliedern.

Damit das Caesar-Verfahren funktioniert, muss ein Klartext also erst einmal in eine standardisierte Form gebracht werden (denn die ganzen nicht druckbaren Sonderzeichen des UTF-8 Standardzeichensatzes würden sie sonst implementierungstechnisch um den Verstand bringen). Sie sollen aus einem Klartext einen **vereinfachten Text** machen, indem sie

- Alle Großbuchstaben durch Kleinbuchstaben ersetzen.
- Alle ä, ö, ü, ß durch ihre Entsprechungen ae, oe, ue, ss ersetzen.

[1] Der Schlüssel ist also sehr kurz im Verhältnis zu heute üblichen und als sicher geltenden Schlüsseln. Ferner erzeugt der Schlüssel auch kein "statistisches Rauschen" des Informationsgehalts im Chiffrat. D.h. die Buchstabenhäufigkeit kann relativ einfach zum Angriff auf den Schlüssel genutzt werden. Sollten sie solche und ähnliche Überlegungen interessieren, sei ihnen die Wahlpflichtveranstaltung "Kryptologie" ans Herz gelegt. In dieser Veranstaltung befassen sie sich fast ausschließlich mit solchen Fragestellungen.

- Mehrere aufeinander folgende Leerzeichen, sind durch ein einzelnes Leerzeichen zu ersetzen.
- Und alle Zeichen, die nicht zum Alphabet [a-z], nicht zu Satzzeichen .,:;?! gehören oder das Leerzeichen sind, sind aus dem Klartext zu löschen bevor dieser an die Verschlüsselung übergeben wird.

Der Klartext

```
Hallo, dies ist _ein_     großartiges Beispiel!
```

hat also die **vereinfachte Form**

```
hallo, dies ist ein grossartiges beispiel!
```

Nur auf diese vereinfachte Form sollen sie die Caesar-Verschlüsselung anwenden. Die Caesar-Verschlüsselung soll dabei keine Satzzeichen und Leerzeichen (Ausnahmen) verschlüsseln und mittels der Methoden `verschluesseln()` und `entschluesseln()` entsprechend ver- und entschlüsseln.

```
1 String klartext = "Ich bin !!! TOP SECRET !!!";
2
3 String verschluesselt = verschluesseln(klartext, 1);
4 // => "jdi cjo !!! upq tfdsfu !!!"
5
6 String entschluesselt = entschluesseln(verschluesselt, 1);
7 // => "ich bin !!! top secret !!!"
```

Beachten Sie bitte, dass sie durchaus negative Schlüssel angeben können (ihr Schlüssel beinhaltet dann als zusätzliche Information, dass die Ersetzung durch Linksverschiebung bestimmt werden soll).

```
1 String klartext = "Ich bin !!! TOP SECRET !!!";
2
3 String verschluesselt = verschluesseln(klartext, -2);
4 // => "gaf zgl !!! rmn qcapcr !!!"
```

Beachten Sie bitte ferner, dass sie auch Verschiebungen (Schlüssel) nutzen können, die größer als 26 (Anzahl der Zeichen [a-z]) sind. Dann wird bei der Caesar-Chiffre einfach wieder von links rein rotiert, wenn sie rechts raus rotieren. D.h. eine Verschiebung um 27, ist dasselbe wie eine Verschiebung von 1.

```
1 String klartext = "Ich bin !!! TOP SECRET !!!";
2
3 String verschluesselt = verschluesseln(klartext, 27);
4 // => "jdi cjo !!! upq tfdsfu !!!"
5
6 verschluesselt = verschluesseln(klartext, 1);
7 // => "jdi cjo !!! upq tfdsfu !!!"
```

Das funktioniert natürlich auch in die Gegenrichtung.

```
1 String klartext = "Ich bin !!! TOP SECRET !!!";
2
3 String verschluesselt = verschluesseln(klartext, -27);
4 // => "hbg ahm !!! sno rdbqds !!!"
5
6 verschluesselt = verschluesseln(klartext, -1);
7 // => "hbg ahm !!! sno rdbqds !!!"
```

```
8
9  verschluesselt = verschluesseln(klartext, 25);
10 // => "hbg ahm !!! sno rdbqds !!!"
```

Anders ausgedrückt: Alle 26-fache eines Schlüssels erzeugen dasselbe Chiffrat[2].

Beachten Sie ferner: Der 0 Schlüssel (und alle seine "Vielfachen", d.h. 26, 52, 78, ... aber auch -26, -52, -78, ...) ist die Identität, d.h. er verschlüsselt eigentlich nicht, sondern erhält den vereinfachten Klartext.

```
1  String klartext = "Faszinierend! /:-|";
2  verschluesselt = verschluesseln(klartext, 0);
3  // => "faszinierend! :"
4
5  verschluesselt = verschluesseln(klartext, 26);
6  // => "faszinierend! :"
7
8  verschluesselt = verschluesseln(klartext, 52);
9  // => "faszinierend! :"
10
11 verschluesselt = verschluesseln(klartext, -26);
12 // => "faszinierend! :"
13
14 verschluesselt = verschluesseln(klartext, -52);
15 // => "faszinierend! :"
```

D.h. die Verschlüsselung mit 0 reduziert den Verschlüsselungsalgorithmus auf die Generierung des vereinfachten Textes.

Implementieren Sie nun bitte ein Programm unter sinnvoller Nutzung von Methoden, um eine interaktive Verschlüsselung/Entschlüsselung für beliebige Texte auf der Konsole zu ermöglichen.

Ein möglicher **Programmablauf zur Verschlüsselung** könnte so aussehen:

```
Bitte geben Sie den Klar-/Geheimtext ein:
Dies ist ein Beispiel
Dann ist dies ihr vereinfachter Klar-/Geheimtext:
dies ist ein beispiel
Wollen Sie diesen verschlüsseln [v] oder entschlüsseln [default]? v
Bitte geben Sie ihren ganzzahligen Schlüssel zum Verschlüsseln ein: -3
Dann ist dies ihr verschlüsselter Text:
afbp fpq bfk ybfpmfbi
```

Ein möglicher **Programmablauf zur Entschlüsselung** könnte so aussehen:

```
Bitte geben Sie den Klar-/Geheimtext ein:
afbp fpq bfk ybfpmfbi
Dann ist dies ihr vereinfachter Klar-/Geheimtext:
afbp fpq bfk ybfpmfbi
Wollen Sie diesen verschlüsseln [v] oder entschlüsseln [default]? n
Bitte geben Sie ihren ganzzahligen Schlüssel zum Entschlüsseln ein: -3
Dann ist dies ihr entschlüsselter Text:
dies ist ein beispiel
```

[2] Ein weiterer Grund, warum dieses Verfahren nicht als sicher gelten kann. Denn es gibt damit effektiv nur 26 Schlüssel.

7.7. !!! NEU !!! Aufgabe: Sandwich

Ein Sandwich besteht aus zwei Schreiben Brot mit etwas dazwischen. Schreiben Sie eine Methode `getSandwich()`, die das erste und letzte Vorkommen von "bread" sucht und alles dazwischen zurückgibt. Wenn es keine zwei Vorkommen von "bread" gibt, soll die leere Zeichenkette zurückgegeben werden.

Beispiele:

- `getSandwich("breadjambread")` → `"jam"`
- `getSandwich("breadhambreadeggsbread")` → `"hambreadeggs"`
- `getSandwich("xxbreadjambreadyy")` → `"jam"`
- `getSandwich("xxbreadyy")` → `""`

Hinweis: *Es kann durchaus vorkommen, dass nicht nur zwei, sondern drei, vier oder sogar noch mehr Vorkommen von "bread" in einer Zeichenkette vorkommen.*

7.8. !!! NEU !!! Aufgabe: Lucky Sum

Entwickeln Sie nun bitte eine Methode `luckySum()` die eine variable Anzahl von ganzzahligen Parametern nur solange aufaddiert bis der Wert 13 iauftaucht.

Beispiele:

- `luckySum(1, 2, 3, 13, 4, 5, 6)` → `6`
- `luckySum(13, 2, 3, 13, 4, 5, 6)` → `0`
- `luckySum()` → `0`
- `luckySum(13)` → `0`
- `luckySum(1, 2, 3, 4)` → `10`
- `luckySum(1, 2, 3, 4, 13)` → `10`

7.9. !!! NEU !!! Aufgabe: Sanduhr

Entwickeln Sie nun bitte eine Methode `hourglass()` die mittels einer übergebenen Zeichenkette und einer Höhe h Sanduhren z.B. für Konsolen erzeugen kann. Hier ein Beispiel für eine Sanduhr der Höhe $h = 5$ und dem Zeichen $c =' *'$.

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

Folgende Aufrufe

```
String glass1 = hourglass(5, "? ");
String glass2 = hourglass(4, "+");
String glass3 = hourglass(3, "z");
String glass4 = hourglass(2, "x");
String glass5 = hourglass(1, ".");

System.out.println("glass1: \n" + glass1);
System.out.println("glass2: \n" + glass2);
System.out.println("glass3: \n" + glass3);
System.out.println("glass4: \n" + glass4);
System.out.println("glass5: \n" + glass5);
```

sollen die folgende Ausgabe erzeugen (allerdings nicht nebeneinander, sondern natürlich untereinander):

```
glass1:          glass2:          glass3:          glass4:          glass5:
? ? ? ? ?        ++++             zzz              xx               .
  ? ? ?           ++               z               xx
    ?             ++              zzz
  ? ? ?           ++++
? ? ? ? ?
```

Teil III.

Aufgaben zu Unit 3
Selbstdefinierbare Datentypen und Collections

8. Definition zusammengesetzter Datentypen

8.1. Aufgabe: Adressdatensatz

Sie sollen nun einen Datentyp `Adresse` entwickeln. Eine `Adresse` besteht dabei aus:

- einem Vornamen
- einem Nachnamen
- einer Straße
- einer Hausnummer
- ggf. einem Hausnummer Zusatz (a, b, c, ...)
- einer Postleitzahl
- einer Stadt
- dem Land

Eine Adresse soll in folgender Form angelegt werden können:

```
Adresse adr1 = new Adresse(
  "Max", "Mustermann",
  "Beispielgasse", 12, 'c',
  12345, "Exemplaria", "Germany"
);

Adresse adr2 = new Adresse(
  "Maren", "Musterfrau",
  "Beispielgasse", 12, 'c',
  12345, "Exemplaria", "Germany"
);
```

Die Ausgabe einer Adresse soll wie folgt funktionieren. Der Aufruf von

```
System.out.println(adr1);
```

soll dies erzeugen:

```
Max Mustermann
Beispielgasse 12c
12345 Exemplaria
Germany
```

Die Adresse soll inhaltlich verglichen werden können. So soll beispielsweise immer

```
adr.equals(adr.clone()) == true
(adr == adr.clone()) == false
```

zu `true` ausgewertet werden.

Implementieren Sie nun den Datentyp `Adresse` und testen Sie ihn mit folgenden Codezeilen:

```
Adresse adr1 = new Adresse(
  "Max", "Mustermann",
  "Beispielgasse", 12, 'c',
  12345, "Exemplaria",
  "Germany"
);

Adresse adr2 = new Adresse(
  "Maren", "Musterfrau",
  "Beispielgasse", 12, 'c',
  12345, "Exemplaria",
  "Germany"
);

Adresse adr3 = adr1; // adr3 und adr1 ist also dieselbe Adresse
Adresse adr4 = adr1.clone(); // adr4 und adr1, zwei inhaltlich gleiche Adressen

System.out.println("adr1 == adr2" + (adr1 == adr2)); // false
System.out.println("adr1 == adr3" + (adr1 == adr3)); // true
System.out.println("adr1 == adr4" + (adr1 == adr4)); // false
System.out.println("adr1.equals(adr4):" + adr1.equals(adr4)); // true
System.out.println("adr4.equals(adr1):" + adr4.equals(adr1)); // true

System.out.println("Adresse 1:");
System.out.println(adr1);
System.out.println("Adresse 2:");
System.out.println(adr2);
System.out.println("Adresse 3:");
System.out.println(adr3);
System.out.println("Adresse 4:");
System.out.println(adr4);
```

8.2. Aufgabe: Mehrere Adressen eingeben

Sie sollen nun ein Programm schreiben, mit dem Sie mehrere Adressen eingeben können. Nach jeder Adresseingabe soll der Nutzer von dem Programm gefragt werden, ob er weitere Datensätze eingeben möchte. Bejaht er dies nicht, so soll das Programm abbrechen und die Anzahl aller bislang eingegeben Datensätze ausgeben.

Hierzu sollen Sie den Datentyp `Adresse` so mittels einer Klassenvariablen `anzahl` erweitern, dass in `anzahl` die Menge aller bislang angelegten Adressdatensätze gespeichert wird und mittels

```
System.out.println(Adresse.anzahl);
```

ausgegeben werden kann.

Ein typischer Programmablauf könnte wie folgt aussehen:

```
Vorname: Maren
Nachname: Musterfrau
Strasse: Beispielgasse
...
Land: Deutschland
Sie haben folgende Adresse eingegeben:
Maren Musterfrau
Beispielgasse 12c
12345 Exemplaria
Deutschland
Wollen Sie eine weitere Adresse eingeben (JA oder NEIN)? JA
Vorname: Max
Nachname: Mustermann
...
Sie haben folgende Adresse eingegeben:
Max Mustermann
Beispielgasse 1b
12345 Exemplaria
Deutschland
Wollen Sie eine weitere Adresse eingeben (JA oder NEIN)? n
Sie haben 2 Datensätze eingegeben.
```

9. Arrays

9.1. Aufgabe: Schachbrett nummerieren

Schreiben Sie nun ein Programm, dass ein zweidimensionales Array über Integer in Schachbrettgröße erzeugt. Die Werte des Arrays sollen wie nachfolgend dargestellt belegt werden.

```
1  2  3  4  5  6  7  8
2  3  4  5  6  7  8  9
3  4  5  6  7  8  9  10
4  5  6  7  8  9  10 11
5  6  7  8  9  10 11 12
6  7  8  9  10 11 12 13
7  8  9  10 11 12 13 14
8  9  10 11 12 13 14 15
```

Das Programm soll dieses Array anschließend wie oben angegeben auf der Konsole ausgeben.

9.2. Aufgabe: Arrays mit flexibler Länge ein- und ausgeben

Schreiben Sie nun ein Programm, dass ein eindimensionales Array über Integer erzeugt, indem von Nutzer folgende Informationen abgefragt werden:

1. Anzahl der einzugebenden Zahlen
2. n mal den Wert beim Nutzer abfragen

Anschließend soll dieses Array in umgekehrter Reihenfolge ausgegeben werden und für jeden Wert soll die Position (Index) innerhalb des Arrays ausgegeben werden.

Ein typischer Programmablauf könnte etwa wie folgt aussehen:

```
Wie viele Werte wollen Sie eingeben? 3
Geben Sie den 1. Wert ein: 12
Geben Sie den 2. Wert ein: 2
Geben Sie den 3. Wert ein: 3
Wert an Position 2 im angelegten Array: 3
Wert an Position 1 im angelegten Array: 2
Wert an Position 0 im angelegten Array: 12
```

9.3. Aufgabe: Werte in Arrays tauschen

Schreiben Sie nun ein Programm, dass eine Anzahl von Werten vom Nutzer abfragt und in einen passend langes Array schreibt. Anschließend soll der Nutzer befragt werden, welche zwei Elemente innerhalb des Arrays getauscht werden soll.

Entwickeln Sie hierzu eine Hilfsmethode

```
public static void swap(int[] xs, int p1, int p2)
```

die die Werte an Position p1 und p2 im Array xs tauscht sowie eine Methode

```
public static String array_to_string(int[] xs)
```

die ein Array xs für folgenden Aufruf

```
1 int[] xs = { 8, 5, 3, 2, 1 };
2 System.out.println(array_to_string(xs));
```

in folgende Ausgabe wandelt.

```
[8, 5, 3, 2, 1]
```

Ein typischer Programmablauf könnte etwa wie folgt aussehen:

```
Wie viele Werte wollen Sie eingeben: 5
Wert Nr. 1: 5
Wert Nr. 2: 6
Wert Nr. 3: 4
Wert Nr. 4: 2
Wert Nr. 5: 10
Sie haben folgendes Array erzeugt: [5, 6, 4, 2, 10]
Welche Positionen wollen Sie tauschen?
Position Nr. 1: 3
Position Nr. 2: 1
Ihr Array sieht nun wie folgt aus: [5, 2, 4, 6, 10]
```

9.4. Aufgabe: Suchworträtsel-Generator

Sie sollen nun ein Suchworträtsel-Generator für Rätselhefte schreiben. Hierzu werden auf einem rechteckigen Feld mehrere Wörter in unterschiedlichen Richtungen zufällig verteilt und die verbleibenden Felder des Suchworträtsels durch zufällige Buchstaben belegt. Innerhalb dieses Buchstabenchaos sind nun die Suchwörter zu suchen.

Nachfolgend ein Beispiel für ein Suchworträtsel.

Rätsel	**Lösung**
i s t x I P r V z u	i s t . . . .
i T K R r v f K w m	. . . .
B e i s p i e l K n	B e i s p i e l . n
R J C J T F e R T u	. . . . . e . . u
C l l Q K k i Y b r	. . . . . i . . r
r n o T v V n w V i	. . . . . n . . .
G H O b i s r V d i	. . . . .
o G S p r A O X B o	. . . . .
W h O x p W d s i W	. . . . .
O M D i e s X X U s	. . D i e s . . . .

Suchwörter können in einem Suchworträtsel anhand folgender Richtungen platziert werden. Abhängig von der Platzierungsrichtung sind diese Suchwörter eher einfach, mittel oder schwierig zu erkennen.

E (einfach)

```
. B e i s p i e l .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
```

SE (mittel)

```
. . . . . .
B . . . . . .
. e . . . . . .
. . i . . . . . .
. . . s . . . . . .
. . . . p . . . . .
. . . . . i . . . .
. . . . . e . . .
. . . . . l . .
. . . . . .
```

S (einfach)

```
. . . . . .
. . . . B . . . . .
. . . . e . . . . .
. . . . i . . . . .
. . . . s . . . . .
. . . . p . . . . .
. . . . i . . . . .
. . . . e . . . . .
. . . . l . . . . .
. . . . . .
```

SW (schwierig)

```
. . . . . .
. . . . . B .
. . . . . e . .
. . . . . i . . .
. . . . . s . . . .
. . . . p . . . . .
. . . i . . . . . .
. . e . . . . . .
. l . . . . . .
. . . . . .
```

W (schwierig)

```
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. l e i p s i e B .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
```

NW (schwierig)

```
. . . . . .
. l . . . . . .
. . e . . . . . .
. . . i . . . . . .
. . . . p . . . . .
. . . . . s . . . .
. . . . . i . . .
. . . . . e . .
. . . . . B .
. . . . . .
```

N (schwierig)

```
. . . . . .
. . . . . l . . . .
. . . . . e . . . .
. . . . . i . . . .
. . . . . p . . . .
. . . . . s . . . .
. . . . . i . . . .
. . . . . e . . . .
. . . . . B . . . .
. . . . . .
```

NE (mittel)

```
. . . . . .
. . . . . l .
. . . . . e . .
. . . . . i . . .
. . . . . p . . . .
. . . . s . . . . .
. . . i . . . . . .
. . e . . . . . .
. B . . . . . .
. . . . . .
```

Suchwörter sollen dabei so für ein Suchworträtsel platziert werden,

- dass der Anfangsbuchstabe eines Suchwortes zufällig auf dem Suchfeld platziert wird und die Folgebuchstaben gem. der Platzierungsrichtung folgen,
- dass Suchwörter in ganzer Länge und ohne Umbruch in das Suchworträtsel passen,
- dass sich zwei Suchwörter niemals in einem Suchworträtsel überlappen.
- Sind zuviele Wörter für ein Suchworträtsel definiert worden (können diese also nicht mehr platziert

werden) werden diese Suchwörter gestrichen. Für jedes zu platzierende Suchwort sind mindestens 10 zufällige Platzierungsversuche vorzunehmen.

- Suchwörter die zu lang sind, werden nie platziert sonder immer gestrichen.

Folgender Aufruf soll bspw. ein einfaches Suchworträtsel auf einem 10 x 10 Feld mit vier Suchworten erzeugen.

```
List<String> words = new LinkedList<String>(Arrays.asList(
  "Dies", "ist", "ein", "Beispiel"
));

char[][] solution = generate(10, 10, '.', words, "E", "S");
System.out.println("Finden sie die folgenden Wörter:\n" + String.join(", ", words));
System.out.println(prettyPrint(noise(solution, '.', "abcdefghijklmnopqrstuvwxyz"));

System.out.println("Lösung:");
System.out.println(prettyPrint(solution));
```

Die Konsolenausgabe sähe aus wie folgt (und variiert natürlich zufällig):

```
Finden sie die folgenden Wörter:
Dies, ist, ein, Beispiel
i B p p D i e s e i
i i s B B l D s s B
i s i i s i n n i e
l i l e p i n B e i
i p s s n i B l s s
i B s B t D n i e p
n e e i n B i s e i
s i B n p p s i s e
s e i i i i i s t l
s e i D s s i s i l
Lösung:
. . . . D i e s . .
. . . . . . . . . B
. . . . . . . . . e
. . . . . . . . . i
. . . . . . . . . s
. . . . . . . . . p
. . e i n . . . . i
. . . . . . . . . e
. . . . . . i s t l
. . . . . . . . . .
```

Folgender Aufruf soll bspw. ein einfaches bis schweres Suchworträtsel auf einem 10 x 10 Feld mit zu vielen (d.h. nicht mehr platzierbaren) Suchworten erzeugen.

```
List<String> words = new LinkedList<String>(Arrays.asList(
  "Dies", "ist", "ein", "zu",
    "langes", "Beispiel", "mit",
  "zu", "vielen", "und",
  "überlangen", "Worten"
));
```

```
7
8  char[][] solution = generate(10, 10, '.', words, "E", "S", "SE", "NE");
9  System.out.println("Finden sie die folgenden Wörter:\n" + String.join(", ", words));
10 System.out.println(prettyPrint(noise(solution, '.', "abcdefghijklmnopqrstuvwxyz"));
11
12 System.out.println("Lösung:");
13 System.out.println(prettyPrint(solution)));
```

Die Konsolenausgabe sähe aus wie folgt (und variiert natürlich zufällig):

```
Finden sie die folgenden Wörter:
Dies, ist, ein, zu, langes, Beispiel, mit, zu, vielen, und
s v i e l e n s m c
w d D e x d k d s s
v u l i p t e e l i
x d n j e l g e y h
f r q d j n i z u y
m h e q a p b u v w
e i s l s f q n i x
i m t i s t x w u z
n m e q a w z z f j
b B e t r s e z e t
Lösung:
. v i e l e n . . .
. . D . . . . . s .
. u . i . . . e l .
. . n . e . g e . .
. . . d . n i z u .
m . . . a p . . . .
e i . l s . . . . .
i . t i s t . . u .
n . e . . . . z . .
. B . . . . . . . .
```

Es bietet sich an das Problem aus mehreren Einzelmethoden zu komponieren. Zu ihrer Hilfe sind nachfolgend noch einmal die Methodensignaturen der in den Codebeispielen genutzten Methoden aufgeführt.

```
1 char[][] generate(int rows, int cols, char empty, List<String> words, String... directions)
2 char[][] noise(char[][] table, char empty, String characters)
3 String  prettyPrint(char[][] table)
```

9.5. !!! NEU !!! Aufgabe: isEverywhere

Schreiben sie nun bitte eine Methode `isEverywhere()`, die für einen gegebenen Wert prüft, ob dieser überall in einem Arrays ist. Überall ist dabei wie folgt definiert: Ein Wert ist überall, wenn für alle benachbarten Elemente eines Arrays mindestens ein Element dieser Paare diesen Wert hat.

Beispiele:

- `isEverywhere([1, 2, 1, 3], 1)` → `true`
- `isEverywhere([1, 2, 1, 3], 2)` → `false`

- `isEverywhere([1, 2, 1, 3, 4], 1)` → `false`

9.6. !!! NEU !!! Aufgabe: whereIsTheMax

Schreiben Sie nun bitte zwei Methoden `int maxRow(int[][] data)` und `int maxCol(int[] row)` mit denen man wie folgt in beliebigen zweidimensionalen Arrays den Index der Zeile und Spalte bestimmen kann, die ein Maximum beinhaltet. Sollte kein Maximum existieren, sollen `maxRow()` und `maxCol()` jeweils `-1` zurückgeben. Sollten mehrere Maxima existieren, soll jeweils der Index des ersten Maximum zurückgeliefert werden.

```
int[][] data = {
  { 1, 2, 3, 4 },
  {},
  { -1, 5, 100, 1 },
  { 0, 2, 5, 6, 7, 8, 9 }
};
int row = maxRow(data);
int col = row == -1 ? -1 : maxCol(data[row]);
System.out.println("Zeile und Spalte mit dem Maximumwert sind: (" + row + ", " + col + ")");
    // => (2, 2)
```

10. Collections

10.1. Aufgabe: Ausgaberoutine für `Stack`, `List` und `Map`

Sie sollen nun eine überladene Methode schreiben, die für die abstrakten Datentypen `Stack`, `List` und `Map` eine gut lesbare Stringrepräsentation erzeugt. So soll für die folgenden Datenstrukturen

```
1 List v = new LinkedList(); for (int i = 1; i < 5; i++) v.add(i);
2 Stack s = new Stack(); for (int i = 1; i < 5; i++) s.add(i);
3 Map t = new HashMap(); for (int i = 65; i < 69; i++) t.put(i, (char)i);
4 // Die letzte erzeugt einen kleinen Teil der sogenannten ASCII Tabelle
```

die folgende Anweisungen folgende Ausgaben erzeugen:

```
1 System.out.println(collection_to_string(v));
```

```
[1, 2, 3, 4]
```

```
1 System.out.println(collection_to_string(s));
```

```
[4, 3, 2, 1]
```

```
1 System.out.println(collection_to_string(t));
```

```
[65 -> A, 66 -> B, 67 -> C, 68 -> D]
```

Implementieren Sie nun bitte die Methode `collection_to_string` und testen Sie diese mit den oben genannten Werten aus.

Hinweis: *Beachten Sie bitte, dass bei der Datenstruktur Map die Reihenfolge der Key/Value Paare nicht notwendig spezifiziert ist. Sie müssen in der Methode `collection_to_string` daher nur sicherstellen, dass alle in der Map befindlichen Key-Value Paare ausgegeben werden. Die Reihenfolge der Ausgabe muss durch Sie nicht programmiertechnisch beeinflusst werden.*

10.2. Aufgabe: Karten mischen

Sie sollen nun eine Methode `merge` schreiben,

```
public static Stack merge(Stack s1, Stack s2)
```

um zwei Kartenstapel zu mischen. Überlegen Sie sich welche Datenstrukturen sich zum Abbilden von Kartenstapeln und den nachfolgenden Anforderungen eignen.

Gegeben seien bspw. die zwei folgenden Kartenstapel

	Pik As
	Herz Bube
Kreuz Dame	Karo Dame
Karo 7	Kreuz Bube
Pik Bube	Pik 4
Herz As	Herz 8
Stapel 1	**Stapel 2**

die in folgender Form gemischt werden sollen.

1. Ist der Stapel 1 nicht leer, nehme die oberste Karte
2. Ist der Stapel 2 nicht leer, nehme die oberste Karte
3. Lege alle genommenen Karten auf den Zielstapel
4. Ist einer von beiden Stapeln (Stapel 1 und Stapel 2) nicht leer fahre mit Schritt 1 fort.

Die Methode `merge` soll als Ergebnis nach folgendem Aufruf

```
1 System.out.println(collection_to_string(merge(stapel1, stapel2));
```

den folgenden Zielkartenstapel erzeugen:

```
[Herz 8, Pik 4, Kreuz Bube, Herz As, Karo Dame,
Pik Bube, Herz Bube, Karo 7, Pik As, Kreuz Dame]
```

Die Methode `merge` soll darüber hinaus

- für beliebig lange und
- für unterschiedliche lange Kartenstapel funktionieren.

Hinweis: *Nutzen Sie die Ausgabemethoden der Aufgabe 10.1 zum Austesten Ihrer Methode.*

10.3. Aufgabe: Karten per Zufall mischen

Überlegen Sie sich nun eine vollkommen vom Zufall abhängige Mischstrategie zweier Kartenstapel. Ist der Datentyp `Stack` nun immer noch ihre Wahl, wenn Sie mitten in einen Kartenstapel hineingreifen wollen?

Gegeben seien wieder die zwei Mengen von Karten

```
[Herz As, Pik Bube, Karo 7, Kreuz Dame]
[Herz 8, Pik 4, Kreuz Bube, Karo Dame, Herz Bube, Pik As]
```

die aber nun rein zufällig mittels der Methode `randmerge` gemischt werden sollen.

`randmerge` soll sich zufällig aus den beiden Stapeln Karten nehmen und diese auf dem Zielkartenstapel ablegen und diesen anschließend zurückgeben.

Mittels des Ausdrucks

```
(int)(n * Math.random())
```

können Sie eine `Integer` Zahl zwischen 0 und n zufällig bestimmen.

Hinweis: *Nutzen Sie die Ausgabemethoden der Aufgabe 10.1 zum Austesten Ihrer Methode `randmerge`. Bei mehrmaligem Starten sollte jeweils eine andere zufällige Reihung erzeugt werden.*

10.4. Aufgabe: Typsichere Nutzung von Collections

Sofern noch nicht geschehen, schreiben Sie ihre Lösungen zu allen Aufgaben dieses Kapitels so um, dass sie **typsicher** sind.

10.5. Aufgabe: n-Gewinnt

Folgender Code soll zufällig ein 5-Gewinnt Spielfeld belegen. "r" soll dabei für rot und "b" für blau stehen. Beachten Sie bitte das wir ggf. auch "x" gegen "y" spielen lassen könnten, oder ggf. "a", "b", "c" und "d". Jedes Feld des Spielfelds soll zufällig mit der Kennung eines Spielers belegt werden, **dabei ist es nicht erforderlich, dass sie prüfen, ob es sich dabei um sinnvolle Spielsituationen handelt.**

Sie sollen vielmehr für x-beliebige n-Gewinnt Spielsituationen bestimmen können, welche Reihen Gewinnreihen sind. Folgender Code prüft für ein zufällig generiertes 5x5 Feld welche 4er- und 5er-Reihen Gewinnreihen wären und gibt diese aus.

```
1  char[][] feld = generiereZufaelligesFeld(5, 5, "rb");
2  System.out.println("Dieses Feld wurde zufällig bestimmt: \n");
3  System.out.println(feldToString(feld));
4
5  List<Reihe> reihen = bestimmeAlleReihen(feld);
6
7  for (int n = feld.length - 1; n <= feld.length; n++) {
8    System.out.println("Auswertung für " + n + "-Gewinnt:");
9    List<Reihe> gewinnReihen = filterReihen(reihen, 'r', n);
10   gewinnReihen.addAll(filterReihen(reihen, 'b', n));
11   for (Reihe r : gewinnReihen)
12     System.out.println("- Diese Reihe gewinnt " + n + "-gewinnt:\n " + r);
13     char[][] nGewinnt = markiereReihen(gewinnReihen, n, feld);
14     System.out.println("\n" + feldToString(nGewinnt));
15   }
16 }
```

Oben stehende Zeilen könnten folgende Ausgabe erzeugen (die Ausgabe ist natürlich zufällig).

```
Dieses Feld wurde zufällig bestimmt:
  b  b  b  b  b
  b  r  r  r  b
  r  b  b  r  r
  r  r  b  r  b
  r  r  r  r  b

Auswertung für 4-Gewinnt:
- Diese Reihe gewinnt 4-gewinnt:
  Zeichen: r; Start: (3, 1); Richtung: (0, 1); Länge: 4
- Diese Reihe gewinnt 4-gewinnt:
  Zeichen: r; Start: (0, 4); Richtung: (1, 0); Länge: 4
- Diese Reihe gewinnt 4-gewinnt:
  Zeichen: b; Start: (0, 0); Richtung: (1, 0); Länge: 5
- Diese Reihe gewinnt 4-gewinnt:
  Zeichen: b; Start: (1, 0); Richtung: (1, 0); Länge: 4

  b  b  b  b  b
  .  .  .  r  .
  .  .  .  r  .
  .  .  .  r  .
  r  r  r  r  .

Auswertung für 5-Gewinnt:
- Diese Reihe gewinnt 5-gewinnt:
  Zeichen: b; Start: (0, 0); Richtung: (1, 0); Länge: 5

  b  b  b  b  b
  .  .  .  .  .
  .  .  .  .  .
  .  .  .  .  .
  .  .  .  .  .
```

Beachten Sie, dass sich Reihen überlagern können. Eine 5er-Reihe, beinhaltet immer zwei 4er-Reihen, drei 3er-Reihen, usw.

Implementieren Sie die fehlenden Methoden, um oben stehendes Verhalten zu realisieren. Gliedern sie dabei sinnvoll das Problem in sinnvolle Teilprobleme und lösen sie die Teilprobleme jeweils mit einzelnen Methoden. Andernfalls werden Sie bei der Lösung ggf. verzweifeln. Aus dem oben stehenden Beispielcode können sie bereits eine Vielzahl an erforderlichen Methoden ableiten (inkl. deren Signaturen).

Sie benötigen zur sinnvollen Lösung dieser Aufgabe alles was bislang eingeführt wurde, d.h.

- Arrays
- Listen
- selbstdefinierte Datentypen
- Methoden
- Kontrollstrukturen wie Schleifen und bedingte Anweisungen.

10.6. Aufgabe: Primzahlen generieren

Eine Primzahl ist nur durch sich selbst und 1 teilbar. 1 ist keine Primzahl. Entwickeln Sie nun bitte eine Methode `boolean isPrim(int n)`, die für eine gegebene Zahl n prüft, ob diese eine Primzahl ist, oder nicht.

Nutzen Sie diese Methode `isPrim()`, um eine Methode `List<Integer> prims(int n)` zu entwickeln, die alle Primzahlen bis n bestimmt und als Liste zurückliefert.

Entwickeln sie ferner eine Methode `String prettyPrint(List<Integer> is, int cols)`, die eine Liste von ganzzahligen Werten in eine Zeichenkette wandelt, die auf der Konsole in `cols` Anzahl an Spalten ausgegeben wird.

Der folgende Aufruf

```
1 System.out.println(prettyPrint(prims(500), 7));
```

soll bspw. folgende Ausgabe erzeugen:

```
2   3   5   7   11  13  17
19  23  29  31  37  41  43
47  53  59  61  67  71  73
79  83  89  97  101 103 107
109 113 127 131 137 139 149
151 157 163 167 173 179 181
191 193 197 199 211 223 227
229 233 239 241 251 257 263
269 271 277 281 283 293 307
311 313 317 331 337 347 349
353 359 367 373 379 383 389
397 401 409 419 421 431 433
439 443 449 457 461 463 467
479 487 491 499
```

10.7. !!! NEU !!! Aufgabe: Gruppierte Armstrongzahlen

Sie haben Zahlen auf ihre Armstrong-Eigenschaft bereits in Aufgabe 5.7 geprüft. Sie sollen nun ein Programm entwickeln, dass alle Armstrongzahlen bis zu einer oberen Schranke bestimmt und diese nach ihrer Stellenanzahl gruppiert ausgibt.

Folgender Code

```
1 int n = 10000;
2 System.out.println("Alle Armstrong-Zahlen bis " + n + ": ");
3 List<Integer> lance = armstrongs(n);
4 Map<Integer, List<Integer>> grouped = groupByLength(lance);
5 String pretty = prettyPrint(grouped);
6 System.out.println(pretty);
```

soll bspw. alle Armstrongzahlen bis zu 4 Stellen wie folgt auf der Konsole ausgeben:

```
Alle Armstrong-Zahlen bis 10000:
{
  1 => [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
  3 => [153, 370, 371, 407]
  4 => [1634, 8208, 9474]
}
```

Leiten Sie alle erforderlichen Methoden aus oben genanntem Beispielcode ab und implementieren Sie diese, so dass das gezeigte Verhalten realisiert wird. Testen Sie ihr Programm mit größeren n.

Hinweis: Für mehr als siebenstellige Armstrongzahlen wird ihr Programm vermutlich eine erhebliche Laufzeit (mehrere Minuten bis zu Stunden) zeigen.

Teil IV.

Aufgaben zu Unit 4
(Streambasierte) I/O Programmierung

11. Dateihandling und Streams

11.1. Aufgabe: Auflisten von Dateien

Sie sollen nun alle Dateien und Verzeichnisse im Home Verzeichnis des Nutzers auflisten und ausgeben.

Mittels

```
1 String homedir = System.getProperty("user.home");
```

können Sie dabei das Home Verzeichnis des Nutzers ermitteln. Sollten Sie eine sehr große und tief verschachtelte Dateistruktur in ihrem Home Verzeichnis haben, dürfen Sie sich auf ein Verzeichnis ihrer Wahl beschränken. Dieses Verzeichnis sollte jedoch mehr als zwei Verzeichnisebenen haben und idealerweise versteckte Dateien/Verzeichnisse beinhalten.

Entwicklen Sie hierzu eine Methode `dirTree()` mit folgender Signatur

```
public static String dirTree(File f, boolean showHidden)
```

die für ein gegebenes `File` Objekt (sollte es sich um ein Verzeichnis handeln) den darunter liegenden Verzeichnisbaum in folgender Form als Zeichenkette aufbereitet. Dabei sind folgende Randbedingungen einzuhalten:

- Es sind innerhalb eines Verzeichnisses erst Verzeichnisse und dann Dateien aufzulisten.
- Zu Dateien ist die Dateigröße in kByte anzugeben (1 kByte sind 1024 Byte).
- Versteckte Dateien/Verzeichnisse sind nur dann aufzulisten, wenn `showHidden == true`.

So haben bspw. die folgenden Anweisungen

```
1 File home = new File(System.getProperty("user.home") + File.separator + "Sites");
2 System.out.println(dirTree(home, false)); // Keine versteckten Dateien
```

folgende Ausgabe auf der Konsole erzeugt.

```
1  |_ Sites
2     |_ courses
3        |_ voop-ss2013
4           |_ exercises
5              |_ WebTech-Aufgabenkatalog.pdf (1156 kB)
6           |_ handouts
7              |_ VL_VOOP_Informationen.pdf (3781 kB)
8              |_ VL_VOOP_Unit01.pdf (8261 kB)
9              |_ VL_VOOP_Unit02.pdf (4444 kB)
10             |_ VL_VOOP_Unit03.pdf (5000 kB)
11             |_ VL_VOOP_Unit04.pdf (4490 kB)
12             |_ VL_VOOP_Unit06.pdf (4884 kB)
```

```
13            |_ VL_VOOP_Unit07.pdf (4512 kB)
14            |_ VL_VOOP_Unit10.pdf (4096 kB)
15         |_ solutions
16            |_ ML-Aufgabe-1_1-1_3.zip (7 kB)
17            |_ ML-Aufgabe-2_1-2_6.zip (22 kB)
18         |_ index.html (3 kB)
19   |_ css
20      |_ images
21         |_ education_front.png (236 kB)
22         |_ fhl.png (15 kB)
23         |_ logo.png (12 kB)
24         |_ research_front.png (247 kB)
25         |_ welcome_front.png (526 kB)
26      |_ style.css (4 kB)
27   |_ pages
28      |_ impressum
29         |_ index.html (6 kB)
30   |_ aws.png (7 kB)
31   |_ bitbucket.png (7 kB)
32   |_ index.html (3 kB)
33   |_ jetbrains.png (5 kB)
```

Jedoch haben bspw. die folgenden Anweisungen

```
1 File home = new File(System.getProperty("user.home") + File.separator + "Sites");
2 System.out.println(dirTree(home, true)); // Mit versteckten Dateien
```

folgende Ausgabe auf der Konsole erzeugt (siehe insbesondere Zeile 30, auf UNIX basierten Systemen werden Dateien versteckt indem ihnen per Konvention ein Punkt vorangestellt wird).

```
1  |_ Sites
2     |_ courses
3        |_ voop-ss2013
4           |_ exercises
5              |_ WebTech-Aufgabenkatalog.pdf (1156 kB)
6           |_ handouts
7              |_ VL_VOOP_Informationen.pdf (3781 kB)
8              |_ VL_VOOP_Unit01.pdf (8261 kB)
9              |_ VL_VOOP_Unit02.pdf (4444 kB)
10             |_ VL_VOOP_Unit03.pdf (5000 kB)
11             |_ VL_VOOP_Unit04.pdf (4490 kB)
12             |_ VL_VOOP_Unit06.pdf (4884 kB)
13             |_ VL_VOOP_Unit07.pdf (4512 kB)
14             |_ VL_VOOP_Unit10.pdf (4096 kB)
15          |_ solutions
16             |_ ML-Aufgabe-1_1-1_3.zip (7 kB)
17             |_ ML-Aufgabe-2_1-2_6.zip (22 kB)
18          |_ index.html (3 kB)
19    |_ css
20       |_ images
21          |_ education_front.png (236 kB)
22          |_ fhl.png (15 kB)
23          |_ logo.png (12 kB)
24          |_ research_front.png (247 kB)
```

```
25          |_ welcome_front.png (526 kB)
26        |_ style.css (4 kB)
27      |_ pages
28        |_ impressum
29          |_ index.html (6 kB)
30      |_ .DS_Store (6 kB)
31      |_ aws.png (7 kB)
32      |_ bitbucket.png (7 kB)
33      |_ index.html (3 kB)
34      |_ jetbrains.png (5 kB)
```

11.2. Aufgabe: Erzeugen von CSV oder XML-Dateien

Gegeben seien folgende Konstantendefinitionen,

```
1   /**
2    * Zu generierender Umsatzbereich.
3    * Zwischen [MIN_VOLUME ... MIN_VOLUME+VOLUME_RANGE].
4    */
5   public static final int VOLUME_RANGE = 0;
6
7   /**
8    * Minimaler zu generierender Umsatz (in USD).
9    */
10  public static final int MIN_VOLUME = 100000;
11
12  /**
13   * Regionen fuer die Umsatzdaten generiert werden sollen.
14   */
15  public static final String[] regions = {
16    "North America", "Middle America", };/*"South America",
17    "West Europe", "Central Europe", "East Europe",
18    "North Africa", "Central Africa", "East Africa",
19    "Central Asia", "North Asia", "Asia Pacific",
20    "China", "India"
21  };*/
22
23  /**
24   * Produktqualifizierer die fuer Produktgenerierung herangezogen werden sollen.
25   */
26  public static final String[] qualifier = {
27    "Smart", };/*"Mobile", "Intelli", "Brilliant", "Wonder", "Genius"
28  };*/
29
30  /**
31   * Produkte, die fuer Produktgenerierung herangezogen werden sollen.
32   */
33  public static final String[] products = {
34    "Phone", };/*"Tablet", "Glass", "Thing", "Thinker", "Cruncher"
35  };*/
36
37  /**
38   * Produktcodes, die fuer Produktgenerierung herangezogen werden sollen.
```

```
39     */
40    public static final String[] codes = {
41      "V0", };/*"1DX", "MX2", "U3R", "X4", "G5T", "HU6"
42    };*/
```

die durch Sie für Umsatzdatengenerierungen herangezogen werden sollen.

Aus den Konstanten

- `qualifier`
- `products` und
- `codes`

sollen sie alle möglichen kombinierbaren Produktbezeichner generieren. Insgesamt gibt es

$$252 = qualifier.length \bullet products.length \bullet codes.length$$

generierbare Produktbezeichner. Ein paar davon sind z.B.

- "*SmartGlass U3R*" oder
- "*IntelliThinker HU6*" oder
- "*WonderCruncher MX2*"

Für jedes dieser Produkte sollen sie für jede Region zufällige Umsatzdaten generieren (also

$$252 \bullet regions.length = 3528$$

). Die Umsätze sollen zwischen `MIN_VOLUME` und `MIN_VOLUME+VOLUME_RANGE` zufällig verteilt sein (Gleichverteilung). Zum Umgang mit Umsatzdaten nutzen Sie bitte die folgende Klasse `SaleData`:

```
1  import java.io.Serializable;
2
3
4  /**
5   * Umsatzdaten zu einem Produkt in einer Region in USD.
6   * @author Nane Kratzke
7   *
8   */
9  public class SaleData implements Serializable {
10
11   /**
12    * For serialization.
13    */
14   public static final long serialVersionUID = -1881535325376543699L;
15
16   /**
17    * Produkt mit dem der Umsatz gemacht wurde.
18    */
19   public String product;
20
21   /**
22    * Region in der der Umsatz gemacht wurde.
23    */
```

```
24    public String region;
25
26    /**
27     * Umsatz in einer Waehrung.
28     * @see #currency
29     */
30    public int volume;
31
32    /**
33     * 3 stelliges Waehrungssymbol.
34     */
35    public String currency;
36
37    /**
38     * Konstruktor zum Erzeugen eines Umsatzobjekts.
39     * @param p Produkt
40     * @param r Region
41     * @param a Umsatz
42     * @param c Waehrung (dreistelliges Symbol, z.B. USD)
43     */
44    public SaleData(String p, String r, int a, String c) {
45      this.product = p;
46      this.region = r;
47      this.volume = a;
48      this.currency = c;
49    }
50
51    /**
52     * Gibt eine kommaseparierte Repraesentation eines SaleData Objekts zurueck.
53     * @return Zeichenkette folgenden Formats: '"{product}", "{region}", "{amount}"\n'
54     */
55    public String toString() {
56      return "\"" + this.product + "\", " +
57          "\"" + this.region + "\", " +
58          "\"" + this.volume + "\", " +
59          "\"" + this.currency + "\"";
60    }
61
62    /**
63     * @see #toString()
64     */
65    public String toCSV() { return this.toString(); }
66
67    /**
68     * Gibt eine XML Repraesentation eines SaleData Objekts zurueck.
69     * @return Zeichenketten im in Aufgabe 11.2 definierten Format.
70     */
71    public String toXML() {
72      String p = "\t\t<product>" + this.product + "</product>\n";
73      String r = "\t\t<region>" + this.region + "</region>\n";
74      String v = "\t\t<volume>" + this.volume + "</volume>\n";
75      String c = "\t\t<currency>" + this.currency + "</currency>\n";
76      return "\t<SaleData>\n" + p + r + v + c + "\t</SaleData>";
77    }
78  }
```

Ihre Aufgabe ist es, diese Umsatzdaten in einer von den folgenden Formaten (ihre Entscheidung) zu speichern:

1. als CSV Datei (kommaseparierte Werte)
2. als XML Datei (Format für (semi-)strukturierte Werte)

Entwickeln sie hierzu bitte eine der folgenden Methoden

- `void storeCSV(List<SaleData> data, File f) throws IOException` oder
- `void storeXML(List<SaleData> data, File f) throws IOException`

die folgende Formate exemplarischen Inhalts erzeugen:

```
"Product", "Region", "Volume", "Currency"
"SmartPhone V0", "North America", "4970124", "USD"
"SmartPhone 1DX", "North America", "2197273", "USD"
"SmartPhone MX2", "North America", "2940532", "USD"
"SmartPhone U3R", "North America", "4087359", "USD"
"SmartPhone X4", "North America", "3198839", "USD"
"SmartTablet MX2", "North America", "10013237", "USD"
"SmartTablet U3R", "India", "930385", "USD"
"SmartTablet X4", "North America", "8683657", "USD"
"SmartTablet G5T", "Central Asia", "3448381", "USD"
"SmartGlass G5T", "East Europe", "561541", "USD"
"GeniusCruncher V0", "South America", "6528435", "USD"
"GeniusCruncher HU6", "India", "1817736", "USD"
```

Listing 11.1: CSV Format (gekürzt)

```
<Sales>
  <SaleData>
    <product>SmartPhone V0</product>
    <region>North America</region>
    <volume>4970124</volume>
    <currency>USD</currency>
  </SaleData>
  <SaleData>
    <product>SmartPhone 1DX</product>
    <region>North America</region>
    <volume>2197273</volume>
    <currency>USD</currency>
  </SaleData>
  <SaleData>
    <product>GeniusCruncher HU6</product>
    <region>India</region>
    <volume>1817736</volume>
    <currency>USD</currency>
  </SaleData>
</Sales>
```

Listing 11.2: XML Format (gekürzt)

11.3. Aufgabe: Auswerten von CSV oder XML-Dateien

Unter den URLs

- `http://www.nkode.io/assets/programming/volume.csv`
- `http://www.nkode.io/assets/programming/volume.xml`

finden Sie Umsatzdaten, die mit der Musterlösung zu Aufgabe 11.2 erzeugt wurden. Entwickeln Sie nun ein Programm, dass aus

- der CSV oder der XML Datei (ihre Entscheidung) für jede Region den akkumulierten Umsatz über alle Produkte

berechnet und den Umsatz der jeweiles **besten Region**, des **besten Produkts** in folgender Form auf der Konsole ausgibt:

```
Beste Region(en) ist/sind [???] mit ??? USD Umsatz.
Beste(s) Produkt(e) ist/sind [???] mit ??? USD Umsatz.
```

Die ??? bezeichnen dabei hoffentlich die durch Sie korrekt berechneten Werte. Beachten Sie bitte ferner, dass es durchaus sein kann, dass mehrere Regionen, mehrere Produkte denselben Umsatz erzeugen können. Es wären in diesen Fällen mehr als ein Produkt, mehr als eine Region auszugeben.

Hinweis: *Volle Punkte bekommen Sie nur dann, wenn Sie die Dateien direkt von der URL einlesen und nicht erst herunterladen und lokal von Ihrem Rechner einlesen.*

11.4. Aufgabe: Buchstabenzählen in Internet-Texten

Unter der URL

- `http://www.nkode.io/assets/programming/countmychars.txt`

finden Sie eine Textdatei. Sie sollen diese hinsichtlich Buchstabenhäufigkeit mittels einer Methode `countChars()` folgender Signatur

- `Map<Character, Integer> countChars(URL url) throws IOException`

auswerten. Ähnliche Probleme haben Sie bereits in Aufgabe 4.3 und 5.5 gelöst. Beachten Sie bitte, dass sich hinter der URL beliebige Texte verbergen können. Die Auswertung soll case-insensitive erfolgen, d.h. 'a' ist wie 'A' zu zählen. Die Auswertung soll nur die Zeichen 'a' bis 'z' und 'A' bis 'Z' umfassen, d.h. keine Interpunktionszeichen, Sonderzeichen, Umlaute oder sonstige (nicht druckbare) Zeichen.

Die Ausführung folgender Zeilen Code

```
1 URL url = new URL("http://www.nkode.io/assets/programming/countmychars.txt");
2 Map<Character, Integer> countedChars = countChars(url);
3
4 for (char c : countedChars.keySet()) {
5   System.out.println("- " + c + ": " + countedChars.get(c));
6 }
```

soll eine **alphabetisch sortierte Ausgabe** erzeugen, die **nur** die Vorkommen im Text vorhandener Zeichen umfasst. Eine denkbare Ausgabe könnte bspw. wie folgt aussehen:

```
- a: 339
- b: 69
- c: 163
- d: 188
```

```
- e: 656
- f: 63
- g: 76
- h: 186
- i: 373
- k: 23
- l: 204
- m: 151
.
.
.
```

Hinweis: *Volle Punkte bekommen Sie nur dann, wenn Sie die Datei direkt von der URL einlesen und nicht erst herunterladen und lokal von Ihrem Rechner einlesen.*

Teil V.

Aufgaben zu Unit 5
Rekursive Programmierung, rekursive Datenstrukturen und Lambdas

12. Rekursive Verarbeitung von Datenstrukturen

12.1. Aufgabe: Rekursives Durchlaufen eindimensionaler Arrays

Sie sollen eine rekursive Methode `zeichenweise` mit folgender Signatur

```
String zeichenweise(int[] xs, String sep)
```

entwickeln, um ein Array über Integers (`int[]`) derart in eine Zeichenkette zu wandeln, dass jedes Element des Arrays durch eine Trennzeichenkette separiert wird.

Das Array

```
int[] a = { 1, 2, 3, 4, 5, 6, 7 };
```

soll also durch Ihre Methode in folgende Zeichenkette gewandelt werden, wenn der Methode als Trennzeichenkette "-" übergeben wurde. D.h.

```
System.out.println(zeichenweise(a, "-"))
```

soll folgendes ausgeben.

```
1-2-3-4-5-6-7
```

Hinweis: *Die Methode `zeichenweise` darf eine überladene Methode gleichen Namens aufrufen.*

12.2. Aufgabe: Rekursives Durchlaufen zweidimensionaler Arrays

Sie sollen nun eine weitere rekursive Methode `zeilenweise` mit folgender Signatur

```
String zeilenweise(int[][] xs)
```

entwickeln, um ein zweidimensionales Array über Integers (`int[][]`) derart in eine Zeichenkette zu wandeln, dass jede Zeile durch ein Linefeed-Zeichen ('\n' = Zeilenumbruch) getrennt wird. Jede Zeile `xs[i]` ($0 < i < xs.length$) soll durch Nutzung der Methode (siehe Aufgabe 12.1)

```
zeichenweise(xs[i], " ")
```

erzeugt werden.

Das Array

```
int[][] xs = {
  { 1, 2, 3 },
  { 4, 5 },
  { 6 }
};
```

soll in eine Zeichenkette gewandelt werden, so dass der Aufruf

```
System.out.println(zeilenweise(xs));
```

folgende Ausgabe für oben stehendes Array erzeugt:

```
1-2-3
4-5
6
```

Hinweis: *Die Methode `zeilenweise` darf eine überladene Methode gleichen Namens aufrufen.*

12.3. Aufgabe: Rekursives Durchlaufen von `Strings`

Sie sollen nun eine rekursive Methode `foo` mit folgender Signatur

```
String foo(String s)
```

schreiben, die folgendes Verhalten zeigt.

```
System.out.println(foo("Hello World"));
```

Konsolenausgabe:

```
H-e-l-l-o- -W-o-r-l-d
```

12.4. Aufgabe: Rekursives Spiegeln von `Strings`

Sie sollen nun eine rekursive Methode `mirror` mit folgender Signatur

```
String mirror(String s)
```

schreiben, die folgendes Verhalten zeigt.

```
System.out.println(mirror("Spiegel"));
```

Konsolenausgabe:

```
SpiegellegeipS
```

12.5. Aufgabe: Rekursives Umdrehen von `Strings`

Sie sollen nun eine rekursive Methode `backward` mit folgender Signatur

```
String backward(String s)
```

schreiben, die folgendes Verhalten zeigt.

```
1 System.out.println(backward("olleH"));
```

Konsolenausgabe:

```
Hello
```

12.6. Aufgabe: Rekursives Durchlaufen einer `Liste`

Sie sollen nun zwei rekursive Methoden namens `join()` und `sum()` schreiben, die folgende Signaturen

```
String join(List vs, String sep)
int sum(List<Integer> vs)
```

haben und folgendes Verhalten zeigen:

```
1 List<Integer> vs = new LinkedList<Integer>();
2 for (int i = 1; i <= 5; i++) vs.add(i);
3 System.out.println(join(vs, "+") + "=" + sum(vs));
```

Konsolenausgabe:

```
1+2+3+4+5=15
```

Hinweis: Achten Sie darauf, dass die Methoden `join()` und `sum()` die Liste nicht verbrauchend verarbeiten. Denken sie ggf. über den Einsatz eines `Iterators` und überladener Methoden nach.

12.7. Aufgabe: Rekursives Zusammenführen zweier `Listen`

Sie sollen nun eine rekursive Methode `zip` schreiben, die folgende Signatur

```
List zip(List vs, List ws)
```

hat und folgendes Verhalten zeigt:

```
1 List as = new LinkedList();
2 for (int i = 7; i >= 1; i -= 2) as.add(i);
3 List bs = new LinkedList();
4 for (int i = 2; i <= 6; i += 2) bs.add(i);
5 System.out.println(serialize(zip(as, bs), " + "));
```

Konsolenausgabe:

```
7 + 2 + 5 + 4 + 3 + 6 + 1
```

12.8. !!! NEU !!! Aufgabe: Rekursives Maximum

Implementieren sie eine rekursive Methode `max()`, die auf einer gegebenen Liste ganzer Zahlen, das Maximum bestimmt. Sollte kein Maximum bestimmbar sein, soll die Methode `null` zurückgeben.

Prüfen Sie ihre Methode mittels einer rekursiven Methode `List<Integer> generateRandoms(int start, int end, int n)` die eine zufällige Liste ganzer gleichverteilter Zahlen zwischen `start` und `end` liefert und `n` Einträge hat.

```
System.out.println(max(generateRandoms(1, 10, 100)));
// => vermutl. 10
System.out.println(max(generateRandoms(1, 10, 0)));
// => null
```

Hinweis: *Sie dürfen alle Methoden verwenden, die die List Schnittstelle zur Verfügung stellt. Insbesondere sei auf `addAll()` hingewiesen. Mit dieser Methode können sie eine Liste an eine andere Liste anhängen.*

12.9. !!! NEU !!! Aufgabe: Kürzeste Wörter bestimmen

Implementieren sie nun bitte eine Methode `getShortestWords()` die nur auf rekurisven Methoden basiert und selbst keine Schleifen verwendet und nicht von globalen Variablen abhängt. Die Methode soll auf einer gegebenen Zeichenkette die kürzesten Wörter bestimmen. Wörter sind dabei alle Zeichenketten, die durch ein oder meherere Leerzeichen voneinander getrennt sind.

```
List<String> words = getShortestWords("Dies ist nur ein dummes Beispiel");
System.out.println(words);
// => [ist, nur, ein]
System.out.println(getShortestWords("Dies ist so ein dummes Beispiel"));
// => [so]
System.out.println(getShortestWords(""));
// => []
```

Hinweis: *Es bietet sich an, dass Problem in Teilprobleme zu gliedern und diese einzeln rekursiv zu loesen. Z.B: erst einmal die Länge der kürzesten Wörter rekurisv zu bestimmen und anschließend die Wörter anhand ihrer Länge aus einer Liste von Wörtern rekursiv herauszufiltern.*

Tipp: Aus einem `Array` lässt sich eine Liste mittels der Methode `Arrays.asList()` machen.

12.10. !!! NEU !!! Aufgabe: Substrings zählen

Implementieren Sie eine rekursive Methode `countSubstring()` die zählt wie häufig eine Zeichenkette in einer anderen Zeichenkette vorkommt.

```
System.out.println(countSubstring("Hello World", "Hello"));
// => 1
System.out.println(countSubstring("Hello World", "l"));
// => 3
System.out.println(countSubstring("Hello", ""));
// => 5
System.out.println(countSubstring("aaa", "aa"));
// => 2
```

Hinweis: Das Problem lässt sich tatsächlich als rekursiver Einzeiler lösen.

13. Bäume

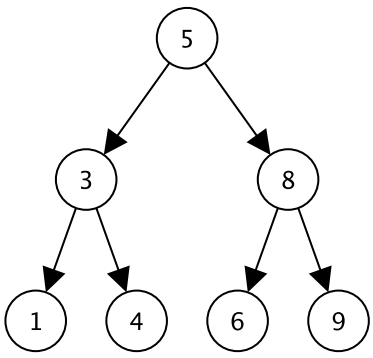
13.1. Aufgabe: Rekursiver Binärbaum

Ein Baum besteht aus Knoten (`Node`). Ein Knoten hat einen ganzzahligen Wert (`value`) und ggf. einen linken (`left`) und einen rechten Sohn (`right`).

Bauen Sie mit folgenden Codezeilen

```
1  Node tree = new Node(5,
2    new Node(3,
3      new Node(1, null, null),
4      new Node(4, null, null),
5    ),
6    new Node(8,
7      new Node(6, null, null),
8      new Node(9, null, null)
9    )
10 );
```

folgenden sortierten Binärbaum auf



und geben Sie diesen in inorder, preorder und postorder Reihenfolge aus.

Implementieren Sie hierzu die Klasse `Node` sowie die folgenden rekursiven Methoden

- `public static String inorder(Node n)`
- `public static String preorder(Node n)`
- `public static String postorder(Node n)`

13.2. Aufgabe: Bestimmen von Baumeigenschaften

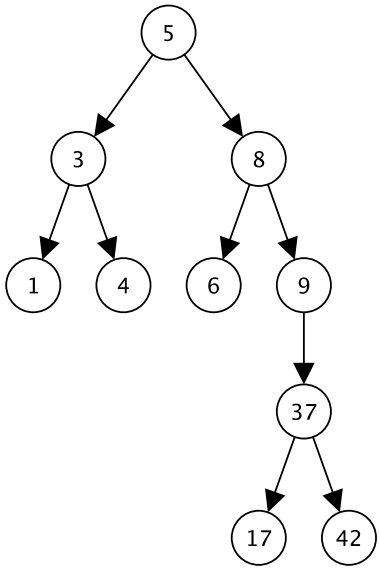
Nutzen Sie für diese Aufgabe ihren Datentyp für einen rekursiven Binärbaum aus Aufgabe 13.1.

Implementieren Sie nun folgende Methoden zur Bestimmung von Baumeigenschaften:

- `public static int countNodes(Node n)` zum Zählen von Knoten in einem Binärbaum
- `public static int countEdges(Node n)` zum Zählen von Kanten in einem Binärbaum
- `public static int height(Node n)` zum Bestimmen der Höhe eines Binärbaums

Testen Sie ihre Routinen anhand des folgenden Binärbaums.

Hinweis: *Für die Methode `height` benötigen Sie vermutlich eine Maximumbestimmung. In JAVA existiert hierzu die Methode `Math.max(int v, int w)` die für zwei Werte den größeren zurückgibt.*



13.3. Aufgabe: `insert()` für sortierte Binärbäume

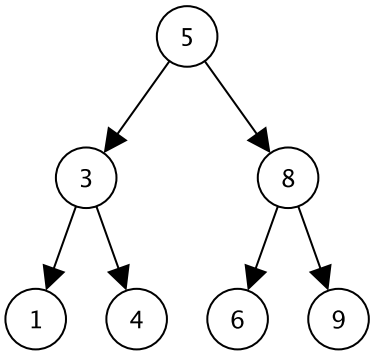
Nutzen Sie für diese Aufgabe ihren Datentyp für einen rekursiven Binärbaum aus Aufgabe 13.1.

Implementieren Sie nun folgende Methode

```
public static void insert(int v, Node tree)
```

zum Einfügen eines neuen Wertes in einen sortierten Binärbaum. Orientieren Sie sich dabei an den Vorlesungsunterlagen.

Für den Baum



sollten die Anweisungen

```
Node tree = new Node(5,
  new Node(3,
    new Node(1, null, null),
    new Node(4, null, null),
  ),
  new Node(8,
    new Node(6, null, null),
    new Node(9, null, null)
  )
);

insert(2, tree);
insert(7, tree);
insert(10, tree);

System.out.println(inorder(tree));
```

die folgende inorder Ausgabe (siehe Aufgabe 13.1) erzeugen.

```
1 2 3 4 5 6 7 8 9 10
```

14. Sortieren

14.1. Aufgabe: `bubbleSort()`

Schreiben Sie nun ein Programm, dass mehrere Eingaben von einem Benutzer einliest und diese sortiert ausgibt. Gibt der Nutzer die Zahl 0 ein, wird die Eingabe abgebrochen. Gibt der Nutzer einer andere Zahl als 0 ein, wird nach einer weiteren Zahl als Eingabe gefragt. Implementieren Sie hierzu die folgenden Methoden

- `public static List<Integer> eingabe()`
- `public static List<Integer> bubbleSort(List<Integer> vs)`
- `public String ausgabe(List<Integer> vs)`

um die zu lösenden Probleme Eingabe, Sortieren und Ausgabe in einzelnen Methoden zu realisieren.

Nachfolgend ein Programm-Ablauf-Beispiel zum besseren Verständnis:

```
Bitte geben Sie eine weitere Zahl ein: 5
Bitte geben Sie eine weitere Zahl ein: 17
Bitte geben Sie eine weitere Zahl ein: -23
Bitte geben Sie eine weitere Zahl ein: 0
Sortierte Ausgabe: -23, 0, 5, 17
```

Hinweis: *Da Sie nicht wissen wieviele Eingaben ein Nutzer machen wird, ist es ggf. sinnvoll anstatt einer semidynamischen Datenstruktur `Array` (wie in der Vorlesung vorgestellt) eine volldynamische Datenstruktur `List` zu nutzen.*

14.2. Aufgabe: `binSort()`

Nutzen Sie nun die Datenstruktur Binärbaum, die Sie in Aufgabe 13.1 (Referenztyp `Node`) entwickelt haben und die `insert()` Operation auf einem Binärbaum (vgl. Aufgabe 13.3), um ebenfalls einen Sortieralgorithmus zu entwickeln.

```
public static List<Integer> binSort(List<Integer> vs)
public static List<Integer> inorder(Node tree)
```

`binSort()` soll eine `List` von vorne nach hinten auslesen und aus diesem einen Binärbaum mittels wiederholten `insert()` Operationen erzeugen. Anschließend soll dieser Binärbaum mittels des inorder Algorithmus durchlaufen werden und jedes Element in eine Rückgabeliste geschrieben werden. Diese `List` ist dann automatisch sortiert. Die Kernzeilen ihres `binSort` Algorithmus könnten bspw. wie folgt aussehen:

```
1 Node tree = new Node(vs.remove(0), null, null);
2 for(int i : vs) {
3   insert(i, tree);
4 }
5 return inorder(tree);
```

Mit folgendem Code können Sie sich 1000 Zufallszahlen zwischen 0 und 100000 erzeugen und per binSort sortieren und auf der Konsole ausgeben lassen, um zu prüfen, ob ihr binSort() korrekt funktioniert.

```
List<Integer> liste = new LinkedList<Integer>();
for (int j = 1; j <= 1000; j++) {
  liste.add((int)(100000 * Math.random()));
}
for (int i : binSort(liste)) {
  System.out.println(i);
}
```

14.3. Aufgabe: Laufzeitvergleich zwischen bubbleSort() und binSort()

Sie sollen nun ein Programm schreiben, dass das Laufzeitverhalten zwischen den beiden Sortieralgorithmen bubbleSort() und binSort() vergleicht.

Das Programm soll wie folgt arbeiten:

1. Der Nutzer wird gefragt, wieviele Zufallszahlen zum Sortieren erzeugt werden sollen.
2. Die Zufallszahlen werden durch das Programm erzeugt und
   a) durch den bubbleSort() Algorithmus sortiert (dabei wird die Laufzeit für den Sortiervorgang gestoppt)
   b) durch den binSort() Algorithmus sortiert (dabei wird ebenfalls die Laufzeit für den Sortiervorgang gestoppt)
3. Anschließend werden die Laufzeiten in für beide Sortieralgorithmen ausgegeben
4. Der Nutzer wird erneut gefragt, wieviele Zufallszahlen zum Sortieren erzeugt werden sollen (bei Eingabe von 0 wird das Programm abgebrochen)

Hinweis: *Um die Laufzeit von einem Programmteil zu bestimmen, können Sie wie folgt vorgehen:*

```
long start = System.currentTimeMillis();
// Hier stehen diverse Anweisungen deren Laufzeit gemessen werden soll
// Z.B. der Aufruf ihres Sortieralgorithmus
long end = System.currentTimeMillis();
long laufzeit = end - start; // gemessene Laufzeit in Milisekunden
```

Nachfolgend ein Programm-Ablauf-Beispiel zum besseren Verständnis:

```
Wieviele Zahlen sollen sortiert werden (0 für Programmende)? 10
Laufzeit für binSort bei 10 Zufallszahlen: 2 ms
Laufzeit für bubbleSort bei 10 Zufallszahlen: 4 ms

Wieviele Zahlen sollen sortiert werden (0 für Programmende)? 100
Laufzeit für binSort bei 100 Zufallszahlen: 5 ms
Laufzeit für bubbleSort bei 100 Zufallszahlen: 21 ms

Wieviele Zahlen sollen sortiert werden (0 für Programmende)? 1000
Laufzeit für binSort bei 1000 Zufallszahlen: 6 ms
```

```
Laufzeit für bubbleSort bei 1000 Zufallszahlen: 1610 ms

Wieviele Zahlen sollen sortiert werden (0 für Programmende)? 0
Ende
```

Vergleichen Sie die Laufzeitverhalten von `binSort` und `bubbleSort` für

- 10
- 20
- 40
- 80
- 160
- 320
- 640
- 1280
- 2560
- 5120
- 10240 und (Geduld mitbringen)
- 100000 Zufallszahlen (bitte sehr viel Geduld mitbringen)

Was fällt Ihnen auf? Welchen Algorithmus würden Sie bevorzugen? `binSort` ist übrigens ein rein rekursiver Algorithmus. Behaupten Sie immer noch, dass rein rekursive Programme grundsätzlich langsamer sind?

Kapitel 14A Lambda-Ausdrücke

Für alle Aufgaben dieses Kapitels gilt. Sie dürfen keinerlei Kontrollanweisungen wie `for()`, `while()`, `if()` etc. verwenden.

14A.1 Aufgabe: Definiere Lamba-Ausdrücke

Definieren sie jeweils Lambda-Ausdrücke und weisen sie diesen geeigneten Referenzvariablen zu. Anschließend wenden sie ihre Lambda-Ausdrücke auf geeignete Parameter an.

Beispielsweise:

- Definieren Sie einen Lambda-Ausdruck zum Quadrieren ganzzahliger Werte. Wenden sie diesen auf den Wert 7 an.

```
1 UnaryOperator<Integer> q = x -> x * x;
2 System.out.println(q.apply(7));
```

Führen Sie dies für folgende Beispiele durch. Achten Sie darauf geeignete Funktionstypen inklusive Typisierung des `java.util.function` Package zu nutzen.

Lambda-Ausdruck ...	**anwenden auf die Werte**
zum n-maligen hintereinander hängen eines Strings (z.B. 3 x "Hello" => "HelloHelloHello"	`(3, ‘‘Hello’’), (-9, ‘‘Hello’’)`
zur Längenbestimmung eines Strings	`‘‘’, ‘‘Hello’’, null, ‘‘World’’`
zum Prüfen, ob eine Zeichenkette eine gerade Anzahl an Zeichen hat	`‘‘Hi’’, ‘‘Hello’’`
zum Prüfen, ob eine Zeichenkette eine ungerade Anzahl an Zeichen hat	`‘‘’, ‘‘Fünf’’, ‘‘Fuenf’’`
zum Durchschnittsbildung zweier Fließkommazahlen	`(3.0, 5.0), (2.7, -1.9), (0.0, 0.0)`
zur Wahl der kürzeren zweier Zeichenketten != null	`(‘‘Hello’’, ‘‘’), (‘‘’, ‘‘Hello’’), (‘‘Hello’’, ‘‘Hi’’), (‘‘Drei’’, ‘‘Vier’’) , (‘‘Vier’’, ‘‘Drei’’)`

14A.2 Aufgabe: Finde gerade Werte

Definieren sie einen Lambda-Ausdruck, der feststellt, ob ein ganzzahliger numerischer Wert gerade ist (durch 2 teilbar ist).

Wenden Sie diesen Lambda-Ausruck auf die folgende Liste von Werten mittels eines Streams an

```
List<Integer> is = Arrays.asList(1, 3, 4, 7, 11, 18, 29, 47, 76);
```

um alle geraden Werte aus dieser Liste zu bestimmen und anschließend auszugeben. Die Ausgabe auf der Konsole sollte etwa wie folgt aussehen:

```
Ursprüngliche Liste:
- 1
- 3
- 4
- 7
- 11
- 18
- 29
- 47
- 76
Gefilterte Liste:
- 4
- 18
- 76
```

14A.3 Aufgabe: Erzeuge eine HTML Liste

Die folgende Liste (und beliebige andere gleich typisierte Listen)

```
1 List<String> strings = Arrays.asList("Dies", "ist", "ein", "Beispiel");
```

sollen sollen sie mittels einer Funktion `String toHtmlList(List<String>)` in einem HTML-Liste konvertieren.

D.h. für den Aufruf von

```
System.out.println(toHtmlList(strings));
```

sollte die Ausgabe etwa wie folgt aussehen:

```
<ul>
<li>Dies</li>
<li>ist</li>
<li>ein</li>
<li>Beispiel</li>
</ul>
```

Implementieren sie die Methode `String toHtmlList(List<String>)` nur mittels Lambda Ausdrücken und Streams.

14A.4 Aufgabe: Erzeuge eine lexikographisch geordnete HTML Liste

Sie sollen die folgende Liste nun in eine lexikographisch aufsteigend geordnete HTML-Liste konvertieren.

```
1 List<String> strings = Arrays.asList("Dies", "ist", "ein", "Beispiel");
```

D.h. für den Aufruf von

```
System.out.println(toOrderedHtmlList(strings));
```

sollte die Ausgabe etwa wie folgt aussehen:

```
<ul>
<li>Beispiel</li>
<li>Dies</li>
<li>ein</li>
<li>ist</li>
</ul>
```

Implementieren sie die Methode `String toOrderedHtmlList(List<String>)` nur mittels Lambda Ausdrücken und Streams.

14A.5 Aufgabe: Bestimme Buchstabenhäufigkeiten in einer Textdatei

Sie sollen nun für eine beliebige Textdatei eine statistische Auswertung über die relavtive Häufigkeit von Zeichen in der Textdatei durchführen. Das **Leerzeichen** soll dabei nicht in die Statistik einfließen (d.h. sie sollen das Leerzeichen ' ' werten, als ob dieses Zeichen nicht existent ist).

Für den Aufruf von

```
1 System.out.println(charStat("http://www.nkode.io/assets/programming/countmychars.txt"));
```

sollte die Statistik etwa wie folgt aussehen.

```
Häufigste Zeichen: (13.872590108968986%)
- 'e'
Seltenste Zeichen: (0.02095557418273260 8%)
- '-'
- ':'
- 'ß'
Zeichen mit abnehmender Häufigkeit:
- 'e': 13.872590108968986%
- 'i': 7.753562447611065%
- 'n': 7.334450963956413%
- 't': 7.229673093042749%
- 'a': 6.999161777032691%
- 's': 6.999161777032691%
...
```

Hinweis: Auch in dieser Aufgabe dürfen sie keinerlei Kontrollstrukturen verwenden. Sie können diese Aufgabe nur mit Rekursion, Streams und Lambda Funktionen lösen. Dies gilt auch für das Einlesen der Textdatei von der URL!

Hilfreich für diese Aufgabe ist es ferner, wenn sie sich das Interface `Map` und die Methoden `entrySet()` sowie die Klasse `Entry` in der Java API ansehen.

Sie benötigen ferner sicher die Klasse `Stream`, und die auf `Stream` arbeitenden Methoden `stream()`, `filter()`, `map()`, `reduce()`.

14A.6 Aufgabe: Bestimme Primzahlen nur mittels Streams und Lambdas

Die folgende Methode bestimmt für einen ganzzahligen Wert n, ob dieser eine Primzahl ist oder nicht.

```java
/**
 * Methode zur Bestimmung ob n eine Primzahl ist.
 * @param n zu prüfende Zahl > 0
 * @return true, wenn n eine Primzahl ist, ansonsten false
 */
public static boolean isPrim(int n) {
  if (n == 1) return false;
  for (int i = 2; i <= n / 2; i++) {
    if (n % i == 0) return false;
  }
  return true;
}
```

Sie sollen die Methode `isPrim()` nun in eine Lambdafunktion `Predicate<Integer> isPrim` umwandeln und `isPrim` in einer anderen Lambdafunktion `Function<Integer, List<Integer>> prims` nutzen, um alle Primzahlen bis zu einer variablen Zahl zu bestimmen.

Die Lambdafunktion `prims` soll bspw. in folgendem Aufruf wie folgt eingesetzt werden können, um die Primzahlen bis 500 tabellarisch in 7 Spalten auszugeben.

```java
int n = 0;
for (int prim : prims.apply(500)) {
  System.out.print(prim + "\t");
  System.out.print(++n % 7 == 0 ? "\n" : "");
}
```

Dieser Aufruf soll folgende Ausgabe auf der Konsole erzeugen.

```
2   3   5   7   11  13  17
19  23  29  31  37  41  43
47  53  59  61  67  71  73
79  83  89  97  101 103 107
109 113 127 131 137 139 149
151 157 163 167 173 179 181
191 193 197 199 211 223 227
229 233 239 241 251 257 263
269 271 277 281 283 293 307
311 313 317 331 337 347 349
353 359 367 373 379 383 389
397 401 409 419 421 431 433
439 443 449 457 461 463 467
479 487 491 499
```

14A.7 !!! NEU !!! Aufgabe: Bestimme vollkommene Zahlen nur mittels Streams und Lambdas

Eine natürliche Zahl n wird vollkommene Zahl (auch perfekte Zahl) genannt, wenn sie gleich der Summe aller ihrer (positiven) Teiler außer sich selbst ist.

Vgl. Wikipedia: `https://de.wikipedia.org/wiki/Vollkommene_Zahl`.

Die kleinsten drei vollkommenen Zahlen sind

- $6 = 1 + 2 + 3 = 6$
- $28 = 1 + 2 + 4 + 7 = 28$
- $496 = 1 + 2 + 4 + 8 + 16 + 31 + 62 + 124 + 248$.

Entwickeln Sie eine Lambdafunktion `perfectNumbers`, die alle vollkommenen Zahlen bis zu einer gegegebenen Schranke bestimmt und bspw. wie folgt aufgerufen werden kann.

```
1 perfectNumbers.apply(500).forEach(n -> System.out.println("- " + n));
```

```
- 6
- 28
- 496
```

Gehen Sie dabei am besten so vor, dass sie das Problem in Teilprobleme gliedern.

1. Entwickeln sie eine Lambdafunktion `divisors`, die einen Stream aller Teiler (ohne sich selber) einer Zahl bestimmt.
2. Entwickeln sie ein Prädikat `perfect`, dass `divisors` nutzt und bestimmt ob eine Zahl eine vollkommene Zahl ist.
3. Entwickeln sie eine Lambdafunktion `perfectNumber`, die `perfect` nutzt, um einen Stream aller vollkommenen Zahlen bis zu einer gegebenen Schranke bestimmt.

Hinweis: *Die Musterlösung inkl. Kommentaren ist nicht länger als 15 Zeilen.*

14A.8 !!! NEU !!! Aufgabe: Noch mehr Lambdas

Für alle folgenden Aufgaben, dürfen sie nur Streams und Lambdafunktionen nutzen. Innerhalb von Lambdafunktionen sind jegliche Kontrollanweisungen wie Schleifen oder bedingte Anweisungen untersagt. Sie dürfen aber natürlich Lambdafunktionen oder Prädikate in anderen Lambdafunktionen oder Prädikaten nutzen.

Beispiel: Bestimme eine Primzahl

Sie sollen bestimmen, ob eine Zahl eine Primzahl ist. Dies ist ein Prädikat. Eine Zahl ist eine Primzahl oder nicht. D.h. wir können den Typ des Lambdas bestimmen.

```
1 Predicate<Integer> isPrim = p -> ...;
```

Eine Primzahl p ist eine Zahl, die nur durch sich selbst und 1 teilbar ist. D.h. wir müssen die Reste aller möglichen Divisionen von 2 bis $p - 1$ bestimmen und prüfen dass keiner dieser Zahlen p ganzzahlig teilen kann.

```
1 Predicate<Integer> isPrim = p -> Stream.iterate(2, n -> n + 1)
2                                  .limit(p)
3                                  .filter(i -> i < p)
4                                  .matchNone(i -> p % i == 0);
5
6 System.out.println(isPrim.test(7)); // => true
7 System.out.println(isPrim.test(10)); // => false
```

Aufmerksamen Lesern wird auffallen, dass 0 und 1 fälschlicherweise als Primzahlen erkannt werden. Da wir also Fälle $p \leq 1$ gesondert behandeln müssen, erweitern wir unser Prädikat letztlich zu:

```
Predicate<Integer> isPrim =
    p -> p <= 1 ? false :
         Stream.iterate(2, n -> n + 1)
               .limit(p)
               .filter(i -> i < p)
               .noneMatch(i -> p % i == 0);

System.out.println(isPrim.test(0)); // => false
System.out.println(isPrim.test(1)); // => false
System.out.println(isPrim.test(2)); // => true
System.out.println(isPrim.test(7)); // => true
System.out.println(isPrim.test(10)); // => false
```

(1) Entferne aus einer Liste von Zeichenketten eine vorgegebene Zeichenkette

Entwickeln Sie eine Lambdafunktion `without`, die wie folgt aufgerufen werden kann.

```
System.out.println(without.apply(Arrays.asList("a", "b", "c", "a"), "b"));
// => [a, c, a]
System.out.println(without.apply(Arrays.asList("a", "b", "c", "a"), "x"));
// => [a, b, c, a]
System.out.println(without.apply(Arrays.asList(), "x"));
// => []
```

(2) Bestimme die Anzahl gleicher Werte in zwei Listen von Zeichenketten

Entwickeln Sie eine Lambdafunktion `common`, die wie folgt aufgerufen werden kann.

```
System.out.println(common.apply(
  Arrays.asList("a", "c", "x"),
  Arrays.asList("b", "c", "d", "x")
)); // => 2

System.out.println(common.apply(
  Arrays.asList("a", "b", "c", "x", "z"),
  Arrays.asList("a", "c", "x")
)); // => 3

System.out.println(common.apply(
  Arrays.asList("a", "b", "c"),
  Arrays.asList("a", "b", "c")
)); // => 3
```

(3) Bestimme die n. Ziffer einer ganzen Zahl

Entwickeln Sie eine Lambdafunktion `nthDigit`, die wie folgt aufgerufen werden kann.

```
System.out.println(nthDigit.apply(128, 0)); // => 1
System.out.println(nthDigit.apply(128, 1)); // => 2
System.out.println(nthDigit.apply(128, 2)); // => 8
System.out.println(nthDigit.apply(-128, 0)); // => 1
```

```
System.out.println(nthDigit.apply(-128, 1)); // => 2
System.out.println(nthDigit.apply(-128, 2)); // => 8
```

Hinweis: Ihre Lambdafunktion muss keine Fälle abfangen, in denen unmögliche Ziffern abgefragt werden (bspw. die -1. Ziffer oder die 3. Ziffer von 128).

(4) Bestimme ob eine Zahl eine sich selbst teilende Zahl ist.

Ein sich selbstteilende Zahl ist eine Zahl die ganzzahlig durch jede Ihrer Ziffern geteilt werden kann. Bspw.:

- 128 / 1 == 128
- 128 / 2 == 64
- 128 / 8 == 16

aber nicht

- 57 / 5 == 11.4 (geht nicht)
- 57 / 7 == 8.142... (geht nicht)

oder

- 120 / 1 == 120 (geht)
- 120 / 2 == 60 (geht)
- 120 / 0 (undefiniert, geht also nicht)

Entwickeln Sie ein Prädikat `dividesSelf`, das wie folgt aufgerufen werden kann.

```
System.out.println(dividesSelf.test(128)); // => true
System.out.println(dividesSelf.test(57)); // => false
System.out.println(dividesSelf.test(101)); // => false
System.out.println(dividesSelf.test(33)); // => true
```

(5) Bestimme alle sich selbst teilenden Zahlen bis zu einer oberen Schranke

Entwickeln Sie eine Lambdafunktion `selfDividers`, die wie folgt aufgerufen werden kann.

```
selfDividers.apply(20).forEach(i -> System.out.print(i + " "));
// => 1 2 3 4 5 6 7 8 9 11 12 15
```

(6) Formatiere eine Liste von Werten als Zeichenkette zu n Spalten.

Entwickeln Sie eine Lambdafunktion `formatAsRows`, die eine Liste von Integer-Werten in eine Zeichenkette zu n Spalten formatiert und bspw. wie folgt aufgerufen werden kann.

```
System.out.println(formatAsRows.apply(selfDividers.apply(1000), 7));
```

Dies sollte folgende Ausgabe erzeugen.

```
  1   2   3   4   5   6   7
  8   9  11  12  15  22  24
 33  36  44  48  55  66  77
 88  99 111 112 115 122 124
126 128 132 135 144 155 162
168 175 184 212 216 222 224
244 248 264 288 312 315 324
333 336 366 384 396 412 424
432 444 448 488 515 555 612
624 636 648 666 672 728 735
777 784 816 824 848 864 888
936 999
```

Teil VI.

Aufgaben zu Unit 6
Einführung in die Objektorientierte Programmierung

15. Autohaus

15.1. Aufgabe: Entwickeln einer Auto-Klasse

Für ein Autohaus ist eine Klasse `Auto` zu entwickeln. Folgende Datenfelder sind bei einem Auto zu verwalten:

- Hersteller vom Typ `String`
- Laufleistung in km vom Typ `long`
- Preis in EUR vom Typ `double`
- Farbe vom Typ `String`
- Unfallwagen vom Typ `boolean`
- Kraftstoff vom Typ `String`
- Leistung in PS vom Typ `double`

Ein Objekt vom Typ `Auto` soll mit folgender Codezeile erzeugt werden können:

```
1 Auto a = new Auto("Ford", 125000, 7999.99, "silber metallic", false, "Diesel", 101.0);
```

Der Aufruf

```
1 System.out.println(a);
```

sollte in etwa folgendes Ergebnis ausgeben:

```
---
Hersteller: Ford
Preis: 7999.99 EUR
Motor: 101.0 PS (Diesel)
KM-Stand: 125000 km
Farbe: silber metallic
unfallfrei
---
```

Der Zusatz "unfallfrei" soll nur ausgegeben werden, wenn das Datenfeld "Unfallwagen" den Wert `false` hat.

Entwickeln Sie nun bitte diese Klasse `Auto` und testen Sie diese mit den folgenden Codezeilen.

```
1 Auto a1 = new Auto("Ford", 125000, 7999.99, "silber metallic", false, "Diesel", 101.0);
2 Auto a2 = new Auto("Mercedes", 63000, 22999.99, "blue silver", true, "Super", 137.0);
3 Auto a3 = new Auto("Daihatsu", 12000, 3099.99, "green dynamite", true, "Benzin", 75.0);
4 System.out.println(a1);
5 System.out.println(a2);
6 System.out.println(a3);
```

Ihre Ausgabe sollte etwa wie folgt aussehen:

```
---
Hersteller: Ford
Preis: 7999.99 EUR
Motor: 101.0 PS (Diesel)
KM-Stand: 125000 km
Farbe: silber metallic
unfallfrei
---
---
Hersteller: Mercedes
Preis: 22999.99 EUR
Motor: 137.0 PS (Super)
KM-Stand: 63000 km
Farbe: blue silver
---
---
Hersteller: Daihatsu
Preis: 3099.99 EUR
Motor: 75.0 PS (Benzin)
KM-Stand: 12000 km
Farbe: green dynamite
---
```

15.2. Aufgabe: Sortieren von Autos

Sie kennen aus der Vorlesung noch den BubbleSort Algorithmus. Dieser Algorithmus kann dazu genutzt werden, eine Menge von Werten auf oder absteigend zu sortieren. Sollten Sie sich nicht mehr erinnern, hier ist noch einmal die Implementierung von `bubbleSort`.

```
1  public static void bubbleSort(int[] xs) {
2    boolean unsorted=true;
3    while (unsorted) {
4      unsorted = false;
5      for (int i=0; i < xs.length-1; i++) {
6        if (!(xs[i] <= xs[i+1])) {
7          int dummy = xs[i];
8          xs[i] = xs[i+1];
9          xs[i+1] = dummy;
10         unsorted = true;
11       }
12     }
13   }
14 }
```

Der Verkaufsleiter wünscht eine Liste aller Autos, die nach Preis absteigend sortiert ist.

Erstellen Sie dem Verkaufsleiter diese Liste, indem Sie den bubbleSort Algorithmus nutzen und entsprechend anpassen, um Autos nach den Wünschen des Verkaufsleiters zu sortieren. Passen Sie bubbleSort daher an den Datentyp (die Klasse) Auto an.

Testen Sie Ihren Code mit folgenden Codezeilen:

```
Auto[] autos = {
  new Auto("Ford", 125000, 7999.99, "silber metallic", false, "Diesel", 101.0),
  new Auto("Mercedes", 63000, 22999.99, "blue silver", true, "Super", 137.0),
  new Auto("Daihatsu", 12000, 3099.99, "green dynamite", true, "Benzin", 75.0)
};
bubbleSort(autos);
for(Auto a : autos) {
  System.out.println(a);
}
```

Sie sollten nun in etwa folgende nach Preis absteigend sortierte Ausgabe erhalten:

```
---
Hersteller: Mercedes
Preis: 22999.99 EUR
Motor: 137.0 PS (Super)
KM-Stand: 63000 km
Farbe: blue silver
---
---
Hersteller: Ford
Preis: 7999.99 EUR
Motor: 101.0 PS (Diesel)
KM-Stand: 125000 km
Farbe: silber metallic
unfallfrei
---
---
Hersteller: Daihatsu
Preis: 3099.99 EUR
Motor: 75.0 PS (Benzin)
KM-Stand: 12000 km
Farbe: green dynamite
---
```

15.3. Aufgabe: Berechnungen auf dem Wagenbestand

Der Verkaufsleiter fragt sich nun:

1. Wie hoch ist eigentlich der Verkaufserlös aller unserer Autos ohne Nachlässe?
2. Wieviel Prozent unserer Autos sind eigentlich Unfallwagen?
3. Wieviel Prozent unserer Autos sind Diesel betrieben?
4. Wieviel Prozent unserer Autos sind elektrobetrieben?
5. Wie hoch ist unser erwarteter Verkaufserlös, wenn im Schnitt 10% Nachlass auf einen unfallfreien Wagen und 25% Nachlass auf einen Unfallwagen gegeben werden?

Beantworten Sie dem Verkaufsleiter diese Fragen für den folgenden Auto-Bestand

```
Auto[] autobestand = {
  new Auto("Ford", 125000, 7999.99, "silber metallic", false, "Diesel", 101.0),
  new Auto("Mercedes", 63000, 22999.99, "blue silver", true, "Super", 137.0),
  new Auto("Daihatsu", 12000, 3099.99, "green dynamite", false, "Benzin", 75.0),
  new Auto("Ford", 1700, 17999.99, "silber metallic", false, "Diesel", 101.0),
  new Auto("Mercedes", 63000, 22999.99, "blue silver", true, "Elektro", 37.0),
  new Auto("Daihatsu", 12000, 3099.99, "green dynamite", true, "Benzin", 75.0),
  new Auto("Ford", 12500, 12999.99, "silber metallic", false, "Super", 121.0),
  new Auto("Mercedes", 6300, 32999.99, "blue silver", false, "Super", 137.0),
  new Auto("Daihatsu", 12000, 3099.99, "green dynamite", true, "Benzin", 75.0),
  new Auto("Ford", 1700, 17999.99, "silber metallic", false, "Diesel", 101.0),
  new Auto("Mercedes", 63000, 22999.99, "blue silver", true, "Elektro", 37.0),
  new Auto("Daihatsu", 12000, 3099.99, "green dynamite", true, "Benzin", 75.0)
}
```

indem Sie entsprechende Auswertemethoden implementieren. Ihre Ausgabe für den Verkaufsleiter sollte etwa wie folgt aussehen. Die ??? kennzeichnen die durch Sie hoffentlich korrekt berechneten Werte.

```
Erlös ohne Nachlässe: ??? EUR
Anteil an Unfallwagen: ?? %
Anteil an Dieselwagen: ?? %
Anteil an Elektrowagen: ?? %
Erlös mit Nachlässen: ??? EUR
```

16. Thinking in Objects

16.1. Aufgabe: Gekapselte Auto-Klasse

In der letzten Übung haben Sie eine Auto-Klasse entwickelt. Mittlerweile haben Sie in der Vorlesung erfahren, dass ein objektorientiertes Prinzip, dass Prinzip der Kapselung ist. Wenn Sie Ihre Implementierung der `Auto`-Klasse genau ansehen, stellen Sie fest, dass diese nicht dem Prinzip der Kapselung entspricht. Alle Datenfelder sind `public` und damit voll zugreifbar. In der OO-Programmierung versucht man so etwas so weit wie möglich zu vermeiden.

Sie haben mittlerweile die Zugriffsmodifikatoren `public`, `protected` und `private` kennengelernt, um eine Kapselung zu implementieren.

Ihr Verkaufsleiter plant nun nicht nur Autos sondern auch Ihre Software für andere Auto-Häuser zu vermarkten, um so ein zweites Standbein für das Unternehmen auzubauen und ein Spezial-Software-Haus für Auto-Häuser zu werden. Sie sind nun gezwungen ihre Quick-And-Dirty Lösung einem sogenannten Refactoring zu unterziehen.

Ihre erste Aufgabe ist es nun, ausgehend von der `Auto`-Klasse der Übung 1, die `Auto`-Klasse zu kapseln. Hierbei gehen Sie bitte wie folgt vor.

1. Deklarieren Sie alle Datenfelder der Klasse Auto als `private`, so dass auf diese nur innerhalb der Auto-Klasse zugegriffen werden kann. Dadurch gewährleisten Sie, dass Sie Kontrolle über Ihre Daten behalten, auch wenn andere Ihre Software als Basis-Bibliothek nutzen wollen.
2. Wird auf ein Datenfeld von außerhalb der Klasse zugegriffen, so sehen Sie bitte eine sogenannte `getter` Methode hierfür vor (übliche Konvention in der OO-Programmierung). `Getter` Methoden geben einfach nur den Wert eines zugehörigen Datenfeldes zurück.
3. Handelt es sich um ein Datenfeld vom Typ `boolean`, dann sehen Sie bitte eine sogenannte `is` Methode vor (übliche Konvention in der OO-Programmierung). is Methoden geben einfach nur den Wert eines zugehörigen boolean Datenfeldes zurück.

Refactoren Sie nun bitte die Klasse `Auto` wie oben angegeben und stellen Sie sicher, dass die ursprüngliche Funktionalität erhalten bleibt.

Testen Sie die refactorte Klasse `Auto` wie auch in Übung 1 mit den folgenden Codezeilen.

```
1 Auto a1 = new Auto("Ford", 125000, 7999.99, "silber metallic", false, "Diesel", 101.0);
2 Auto a2 = new Auto("Mercedes", 63000, 22999.99, "blue silver", true, "Super", 137.0);
3 Auto a3 = new Auto("Daihatsu", 12000, 3099.99, "green dynamite", true, "Benzin", 75.0);
4 System.out.println(a1);
5 System.out.println(a2);
6 System.out.println(a3);
```

Ihre Ausgabe sollte wieder wie folgt aussehen:

```
---
Hersteller: Ford
```

```
Preis: 7999.99 EUR
Motor: 101.0 PS (Diesel)
KM-Stand: 125000 km
Farbe: silber metallic
unfallfrei
---
---
Hersteller: Mercedes
Preis: 22999.99 EUR
Motor: 137.0 PS (Super)
KM-Stand: 63000 km
Farbe: blue silver
---
---
Hersteller: Daihatsu
Preis: 3099.99 EUR
Motor: 75.0 PS (Benzin)
KM-Stand: 12000 km
Farbe: green dynamite
---
```

16.2. Aufgabe: Automatisierte Bestandsverwaltung aller angelegten Autos

Bei weiterer Betrachtung Ihres Codes fällt Ihnen auf, dass sie zwar Autos anlegen können, alle angelegten Autos jedoch immer umständlich in einem Array gespeichert werden müssen. Wenn ein Programmierer dies einmal "vergisst", ist ein Auto "futsch".

Sie entschließen sich den Bestand aller Autos automatisiert in einer Klassenvariable `bestand` vom Typ `List` zu speichern. Jedes neu angelegte Auto soll diesem `bestand` hinzugefügt werden. Aus der Vorlesung wissen Sie, dass sie dazu vermutlich Ihren Konstruktor anpassen müssen.

Ferner soll die `Auto` Klasse noch folgende Klassenmethoden anbieten:

- `getBestand()` soll die Liste mit allen bislang angelegten Autos liefern
- `getAnzahl()` soll einen `int` Wert liefern, der angibt wieviele Autos bislang angelegt worden sind

Testen Sie bitte Ihre neuen Klassenmethoden mit folgenden Codezeilen

```
1 new Auto("Ford", 125000, 7999.99, "silber metallic", false, "Diesel", 101.0);
2 new Auto("Mercedes", 63000, 22999.99, "blue silver", true, "Super", 137.0);
3 new Auto("Daihatsu", 12000, 3099.99, "green dynamite", true, "Benzin", 75.0);
4
5 for(Auto a : Auto.getBestand()) System.out.println(a);
6 System.out.println("Insgesamt " + Auto.getAnzahl() + " Autos im Bestand.");
```

Sie sollten nun in etwa folgende Ausgabe erhalten:

```
---
Hersteller: Ford
```

```
Preis: 7999.99 EUR
Motor: 101.0 PS (Diesel)
KM-Stand: 125000 km
Farbe: silber metallic
unfallfrei
---
---
Hersteller: Mercedes
Preis: 22999.99 EUR
Motor: 137.0 PS (Super)
KM-Stand: 63000 km
Farbe: blue silver
---
---
Hersteller: Daihatsu
Preis: 3099.99 EUR
Motor: 75.0 PS (Benzin)
KM-Stand: 12000 km
Farbe: green dynamite
---
Insgesamt 3 Autos im Bestand.
```

16.3. Aufgabe: Weitere Funktionen für die Bestandsverwaltung

Sie entschließen sich ferner die folgenden Klassenmethoden der ersten Übung (die dort in der Klasse AufgabeNr1 außerhalb der Klasse `Auto` implementiert waren)

- `public static double erloes_inkl_nachlass(Auto[], double, double)`
- `public static double anteil_unfallwagen(Auto[])`
- `public static double anteil_kraftstoffart(Auto[], String)`
- `public static void bubbleSort(Auto[])`

in die Klasse `Auto` zu integrieren und direkt auf dem Autobestand der Klassenvariable `bestand` arbeiten zu lassen. Denn all diese Methoden arbeiten auf dem Auto Bestand und gehören daher auch logisch zu der Klasse `Auto`. Die neue Signatur der Methoden sieht dann wie folgt aus (da man das Auto-Array nicht mehr benötigt).

- `public static double erloes_inkl_nachlass(double, double)`
- `public static double anteil_unfallwagen()`
- `public static double anteil_kraftstoffart(String)`
- `public static void bubbleSort()`

Testen Sie Ihr Refactoring mit folgenden Code Zeilen

```
1 new Auto("Ford", 125000, 7999.99, "silber metallic", false, "Diesel", 101.0);
2 new Auto("Mercedes", 63000, 22999.99, "blue silver", true, "Super", 137.0);
3 new Auto("Daihatsu", 12000, 3099.99, "green dynamite", false, "Benzin", 75.0);
4 new Auto("Ford", 1700, 17999.99, "silber metallic", false, "Diesel", 101.0);
5 new Auto("Mercedes", 63000, 22999.99, "blue silver", true, "Elektro", 37.0);
```

```
6
7  Auto.bubbleSort();
8
9  for(Auto a : Auto.getBestand()) System.out.println(a);
10
11 System.out.println("Dieselautos: " + Auto.anteil_kraftstoffart("Diesel") + "%");
12 System.out.println("Elektroautos: " + Auto.anteil_kraftstoffart("Elektro") + "%");
13 System.out.println("Unfallwagen: " + Auto.anteil_unfallwagen() + "%");
14 System.out.println("Erlös ohne Nachlass: " + Auto.erloes_inkl_nachlass(0, 0) + " EUR");
15 System.out.println("Erlös mit Nachlass: " + Auto.erloes_inkl_nachlass(0.1, 0.25) + " EUR");
```

16.4. Aufgabe: Berechnungen mit der neuen Autoverwaltung

Ihr Verkaufsleiter hat endlich Überblick über seinen Wagenbestand und fragt sich nun:

- Um wieviel Prozent könnten wir unseren Erlös eigentlich steigern, wenn wir behaupten würden, dass wir unsere Unfallwagen unfallfrei wären?
- Ein Unfallwagen erhält üblicherweise 25% Ermäßigung
- Alle anderen Wagen etwa 10% Ermäßigung

Beantworten Sie dem Verkaufsleiter diese Fragen für den folgenden Auto-Bestand

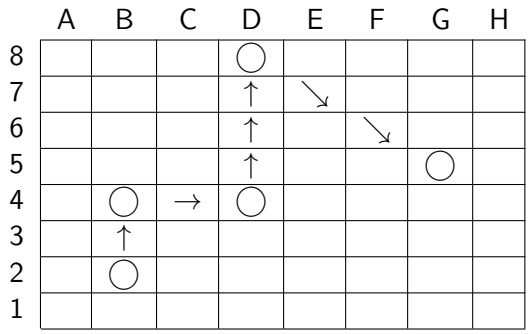
```
1  new Auto("Ford", 125000, 7999.99, "silber metallic", false, "Diesel", 101.0);
2  new Auto("Mercedes", 63000, 22999.99, "blue silver", true, "Super", 137.0);
3  new Auto("Daihatsu", 12000, 3099.99, "green dynamite", false, "Benzin", 75.0);
4  new Auto("Ford", 1700, 17999.99, "silber metallic", false, "Diesel", 101.0);
5  new Auto("Mercedes", 63000, 22999.99, "blue silver", true, "Elektro", 37.0);
6  new Auto("Daihatsu", 12000, 3099.99, "green dynamite", true, "Benzin", 75.0);
7  new Auto("Ford", 12500, 12999.99, "silber metallic", false, "Super", 121.0);
8  new Auto("Mercedes", 6300, 32999.99, "blue silver", false, "Super", 137.0);
9  new Auto("Daihatsu", 12000, 3099.99, "green dynamite", true, "Benzin", 75.0);
10 new Auto("Ford", 1700, 17999.99, "silber metallic", false, "Diesel", 101.0);
11 new Auto("Mercedes", 63000, 22999.99, "blue silver", true, "Elektro", 37.0);
12 new Auto("Daihatsu", 12000, 3099.99, "green dynamite", true, "Benzin", 75.0);
```

indem Sie entsprechende Auswertungen implementieren. Ihre Ausgabe für den Verkaufsleiter sollte etwa wie folgt aussehen. Die ??? kennzeichnen die durch Sie hoffentlich korrekt berechneten Werte.

```
Prozentuale Steigerung des Erlöses: ??? %
Absolute Steigerung des Erlöses: ?? EUR
```

16.5. Aufgabe: Possible Chessmen

Ein Schachbrett ist wie folgt aufgebaut.



Auf diesem Schachbrett ist eine mögliche Zugabfolge gekennzeichnet:

'B2' - 'B4' - 'D4' - 'D8' - 'G5'

Gem. den Schachregeln kann diese Zugabfolge nicht jede Figur ziehen.

Sie sollen nun bestimmen, welche Schachfiguren (König, Dame, Turm, Läufer, Springer, weißer Bauer, schwarzer Bauer) in der Lage sind, eine **beliebig vorgegebene Zugfolge** zu spielen. Folgende Codezeilen

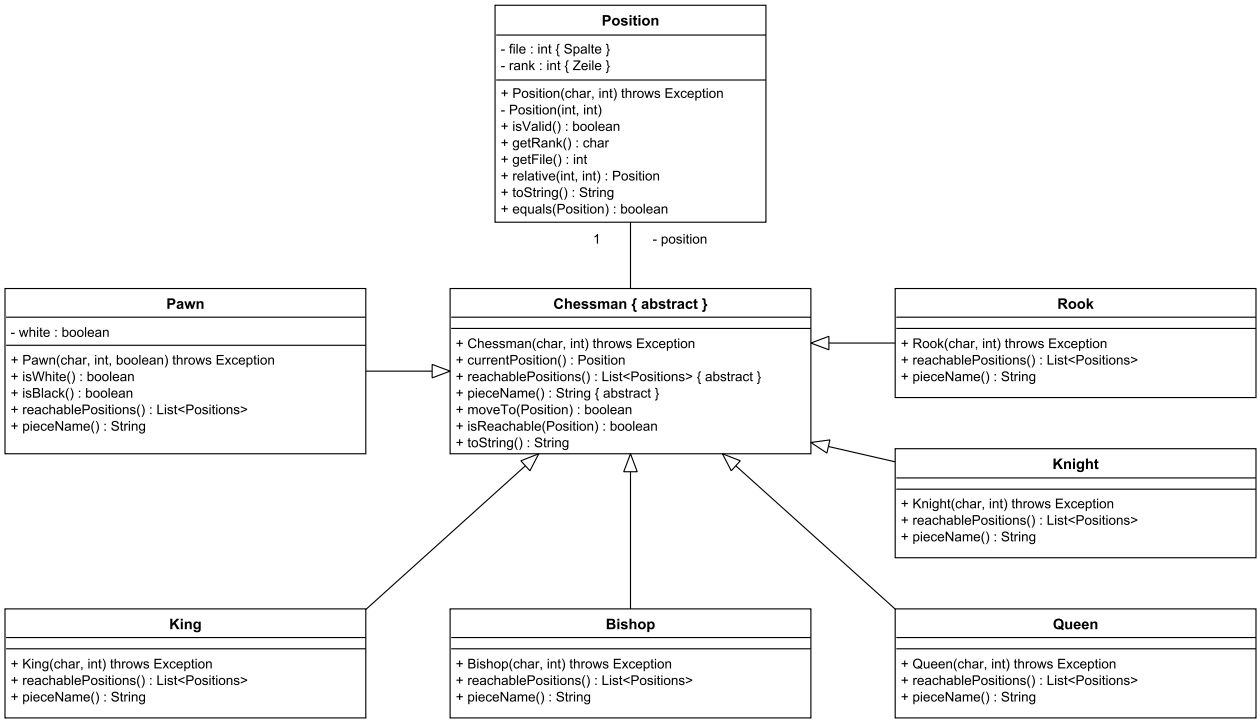
```
List<Position> moves = Arrays.asList(
  new Position('B', 2),
  new Position('B', 4),
  new Position('D', 4),
  new Position('D', 8),
  new Position('G', 5)
);

for (int i = 0; i < moves.size(); i++) {
  List<Position> submoves = moves.subList(0, i + 1);
  String mans = possibleChessmen(submoves).stream()
                                          .map(cm -> cm.pieceName())
                                          .collect(Collectors.joining(", "));
  System.out.println(moves.get(i) + ": " + mans);
}
```

sollen folgende Ausgabe erzeugen

```
B2: König, Dame, Turm, Läufer, Springer, Weißer Bauer, Schwarzer Bauer
B4: Dame, Turm, Weißer Bauer
D4: Dame, Turm
D8: Dame, Turm
G5: Dame
```

und stellen so für die oben exemplarisch angegebene Zufabfolge dar, wie die möglichen Figuren Zug für Zug sinken. Zur Hilfe sei ihnen folgendes UML-Diagramm an die Hand gegeben, mit denen das Problem objektorientiert strukturiert werden kann.



Implementieren Sie die Methode

- `possibleChessmen()` und
- das UML-Diagramm

um das Problem zu lösen.

Teil VII.

Aufgaben zu Unit 7
Objektorientierte Programmierung

17. Objektkommunikation

17.1. Aufgabe: Eine Autobestandsverwaltung auf Basis von Objektkommunikation

Neben den Prinzpien Kapselung, Polymorphie und Vererbung bestimmt ein weiteres Prinzip der Objektorientierung wesentlich die Struktur objektorientierter Software: Objektkommunikation. In der Objektorientierung lösen mehrere Objekte im Zusammenspiel ein Problem/eine Aufgabe. Hierzu müssen diese Objekte – wie auch im echten Leben – miteinander kommunizieren zu können. Während in den ursprünglichsten objektorientierten Sprachen (wie bspw. Smalltalk) tatsächlich noch Messages hierfür eingesetzt wurden, wird dies heutzutage üblicherweise mittels Methoden (d.h. Routinenaufrufen) abgebildet. Jeder Routinenaufruf kann aber wie eine Nachricht von einem Objekt an ein anderes Objekt aufgefasst werden.

So kann bspw. das Objekt a eine Kenntnisbeziehung zu einem Objekt b haben. Objekt b kann ferner eine Methode $foo()$ haben. Das Objekt a kann dann (aufgrund seiner Kenntnisbeziehung zum Objekt b) die Methode $foo()$ des Objekts b aufrufen. Üblicherweise geschieht das in etwa in folgender Form.

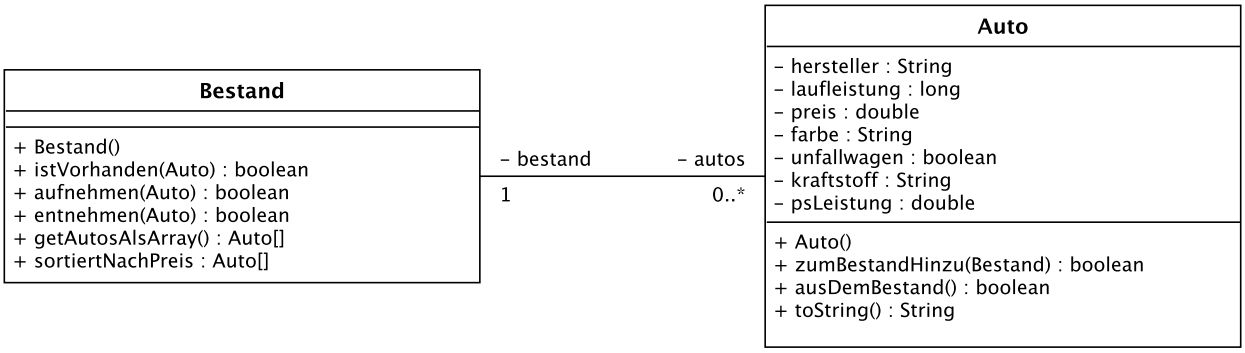
```
1  class A {
2    B b;
3
4    [...]
5
6    void function bar() {
7      b.foo();
8      /* Objekte der Klasse A rufen an dieser Stelle
9      * die Methode foo() eines Objekts der Klasse B
10     * auf.
11     */
12   }
13
14   [...]
15 }
```

Der Kommentar in oben stehendem Listing gibt eine methodenorientierte Interpretation der Interaktion zwischen die beiden Objekten a und b an. Genausogut kann man aber auch sagen, dass das Objekt a die Nachricht foo an ein Objekt b sendet. Dies ist die objektkommunikationsorientierte Interpretation, die in den ursprünglichen objektorientierten Sprachen wie Smalltalk definiert wurde. Beide Formen sind gleichwertig. Nachrichten können genauso wie Methoden Parameter "übertragen".

In der Objektorientierung macht man sich dieses Prinzip zu nutze, um Aufgaben durch mehrere Objekte bearbeiten zu lassen (d.h. auf mehrere Schultern zu verteilen). Die Abstimmung zwischen den Objekten erfolgt dann auf Basis eines Nachrichtenaustauschs/Methodenaufrufen zwischen den Objekten.

Wesentlich hierfür ist eine Kenntnisbeziehung zwischen den Objekten. In UML Klassendiagrammen können diese z.B. durch Assoziationen (Kenntnisrelationen) zum Ausdruck gebracht werden (auch Aggregationen und Kompositionen sind solche Kenntnisrelationen).

Sie sollen auf Basis dieser neuen Kenntnisse Ihre Bestandsverwaltung etwas objektorientierter umbauen. Orientieren Sie sich bitte an folgendem UML Klassendiagramm, dass eine neue Klasse Bestand einführt. Objekte der Klasse Bestand haben die Aufgabe, eine Menge von Autos zu verwalten.



Ein Autobestand wird mit dieser Vorgabe, dann wie folgt angelegt und nach Preis sortiert ausgegeben:

```
1  Bestand bestand = new Bestand();
2  Auto a = new Auto("Ford", 125000, 7999.99, "silber metallic", false, "Diesel", 101.0);
3  a.zumBestandHinzu(bestand);
4
5  a = new Auto("Mercedes", 63000, 22999.99, "blue silver", true, "Super", 137.0);
6  a.zumBestandHinzu(bestand);
7
8  a = new Auto("Daihatsu", 12000, 3099.99, "green dynamite", false, "Benzin", 75.0);
9  a.zumBestandHinzu(bestand);
10
11 a = new Auto("Ford", 1700, 17999.99, "silber metallic", false, "Diesel", 101.0);
12 a.zumBestandHinzu(bestand);
13
14 a = new Auto("Mercedes", 63000, 22999.99, "blue silver", true, "Elektro", 37.0);
15 a.zumBestandHinzu(bestand);
16
17 for(Auto auto : bestand.sortiertNachPreis()) System.out.println(auto);
```

Folgende Hinweise seien Ihnen noch zu den Methoden der Klasse `Bestand` mit erläuternd auf den Weg gegeben.

- `Bestand.istVorhanden(Auto) : boolean` Diese Methode prüft, ob ein Auto bereits im Bestand ist. Liefert `true` wenn ja, andernfalls `false`.
- `Bestand.aufnehmen(Auto) : boolean` Diese Methode fügt ein Auto-Objekt zum Bestand hinzu. Der Parameter referenziert das Auto-Objekt, das dem Bestand hinzugefügt werden soll. Die Methode liefert `false`, wenn Auto-Objekt schon im Bestand vorhanden ist, ansonsten `true`.
- `Bestand.entnehmen(Auto) : boolean` Diese Methode entfernt das angegebene Auto-Objekt aus dem Bestand. Die Methode liefert `false`, wenn Auto nicht im Bestand vorhanden war, ansonsten `true`.
- `Bestand.getAutosAlsArray() : Auto[]` Diese Methode liefert alle Auto-Objekte des Bestands als Array.
- `Bestand.sortiertNachPreis() : Auto[]` Diese Methode liefert ein nach Preis absteigend sortiertes Array von Auto-Objekten des Bestands.

Folgende Hinweise seien Ihnen noch zu den Methoden der Klasse `Auto` mit erläuternd auf den Weg gegeben.

- `Auto.zumBestandHinzu(Bestand) : boolean` Diese Methode fügt ein Auto-Objekt in einen Bestand ein. Der Parameter bezeichnet dabei den das Bestand-Objekt, in das das Auto eingefügt werden soll. Die Methode liefert `true`, wenn das Einfügen erfolgreich war. Sie liefert `false`, wenn das Auto-Objekt bereits im Bestand vorhanden ist (Referenzvergleich).
- `ausDemBestand() : boolean` Diese Methode entfernt das Auto-Objekt (sich selbst) aus dem im Objekt gespeicherten Bestand. Sie liefert `false`, wenn das Auto keinem Bestand zugeordnet war, ansonsten `true`.
- `toString() : String` Diese Methode soll dieselbe String Repräsentation eines Auto-Objekts erzeugen, wie auch in Aufgabe 15.1.

Implementieren Sie nun bitte die Klassen `Auto` und `Bestand` wie erläutert.

17.2. Aufgabe: Berechnungen mit der neuen objektorientierten Bestandsverwaltung

Mit der neuen Bestandsverwaltung sollen Sie dem Verkaufsleiter exakt dieselben Fragen, wie in Aufgabe 16.4 gestellt, beantworten können. Sie haben hierzu bereits einige Hilfsmethoden in der Klasse `Auto` implementiert. Sind diese in der Klasse `Auto` noch sinnvoll platziert? Welche Objekte sollten in dieser geänderten Gliederung eigentlich die aus Aufgabe 16.4 bekannten Fragen beantworten. Objekte der Klasse `Auto` oder Objekte der Klasse `Bestand`?

Bitte schreiben Sie Ihren Code bitte dahingehend um, dass die Fragen nun ebenfalls mit der voll objektorientierten Bestandsverwaltung beantwortet werden können. Testen Sie Ihren Code auf folgendem Bestand.

```
1  Bestand bestand = new Bestand();
2  Auto a = new Auto("Ford", 125000, 7999.99, "silber metallic", false, "Diesel", 101.0);
3  a.zumBestandHinzu(bestand);
4
5  a = new Auto("Mercedes", 63000, 22999.99, "blue silver", true, "Super", 137.0);
6  a.zumBestandHinzu(bestand);
7  a = new Auto("Daihatsu", 12000, 3099.99, "green dynamite", false, "Benzin", 75.0);
8  a.zumBestandHinzu(bestand);
9
10 a = new Auto("Ford", 1700, 17999.99, "silber metallic", false, "Diesel", 101.0);
11 a.zumBestandHinzu(bestand);
12
13 a = new Auto("Mercedes", 63000, 22999.99, "blue silver", true, "Elektro", 37.0);
14 a.zumBestandHinzu(bestand);
15
16 a = new Auto("Daihatsu", 12000, 3099.99, "green dynamite", true, "Benzin", 75.0);
17 a.zumBestandHinzu(bestand);
18
19 a = new Auto("Ford", 12500, 12999.99, "silber metallic", false, "Super", 121.0);
20 a.zumBestandHinzu(bestand);
21
22 a = new Auto("Mercedes", 6300, 32999.99, "blue silver", false, "Super", 137.0);
23 a.zumBestandHinzu(bestand);
24
25 a = new Auto("Daihatsu", 12000, 3099.99, "green dynamite", true, "Benzin", 75.0);
26 a.zumBestandHinzu(bestand);
27
```

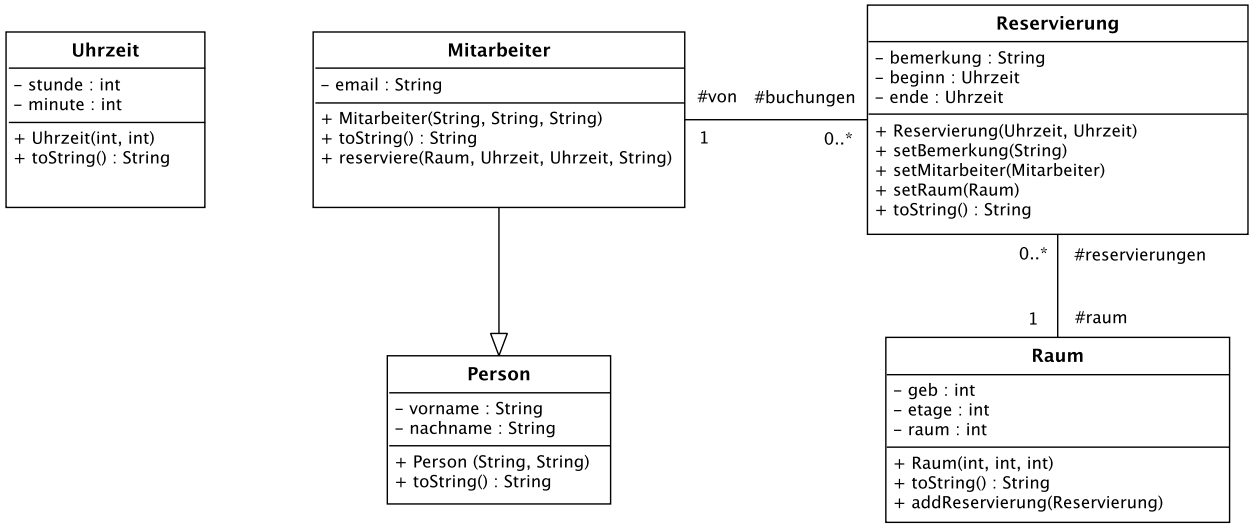
```
28 a = new Auto("Ford", 1700, 17999.99, "silber metallic", false, "Diesel", 101.0);
29 a.zumBestandHinzu(bestand);
30
31 a = new Auto("Mercedes", 63000, 22999.99, "blue silver", true, "Elektro", 37.0);
32 a.zumBestandHinzu(bestand);
33
34 a = new Auto("Daihatsu", 12000, 3099.99, "green dynamite", true, "Benzin", 75.0);
35 a.zumBestandHinzu(bestand);
```

Ihre Ausgabe für den Verkaufsleiter sollte etwa wie folgt aussehen. Die ??? kennzeichnen die durch Sie hoffentlich korrekt berechneten Werte.

```
Prozentuale Steigerung des Erlöses: ???%
Absolute Steigerung des Erlöses: ?? EUR
```

17.3. Aufgabe: UML Beispiel I (Raumverwaltung)

Gegeben ist folgendes UML Diagramm.



Ergänzend sind Ihnen folgende Hinweise für die Implementierung gegeben.

Eine Uhrzeit soll wie folgt angelegt werden können.

```
1 Uhrzeit u = new Uhrzeit(12, 54);
```

und mittels `System.out.println(u)` wie folgt ausgegeben werden können.

```
12:54 Uhr
```

Ein Mitarbeiter soll wie folgt angelegt

```
1 Mitarbeiter m = new Mitarbeiter("Max", "Mustermann", "mustermann@beispiel.com");
```

und mittels `System.out.println(m)` wie folgt ausgegeben werden können.

```
Max Mustermann (mustermann@beispiel.com)
```

Ein Raum soll wie folgt angelegt

```
1 Raum r = new Raum(18, 0, 1);
```

und mittels `System.out.println(r)` wie folgt ausgegeben werden können:

```
Raum 18-0.1
```

Mitarbeiter können Räume wie folgt reservieren. Sind beispielsweise die folgenden Mitarbeiter und Räume gegeben

```
1 Mitarbeiter m1 = new Mitarbeiter("Max", "Mustermann", "mm@ex.com");
2 Mitarbeiter m2 = new Mitarbeiter("Tessa", "Loniki", "loniki@ex.com");
3 Raum r1 = new Raum(18,0,1);
4 Raum r2 = new Raum(2,1,9);
5 Raum r3 = new Raum(2,1,10);
```

und werden zusätzlich die folgenden Anweisungen ausgeführt

```
1 m1.reserviere(r1, new Uhrzeit(12, 30), new Uhrzeit(14, 30), "VOOP");
2 m1.reserviere(r2, new Uhrzeit(14, 30), new Uhrzeit(16, 30), "WebTech");
3 m2.reserviere(r2, new Uhrzeit(12, 30), new Uhrzeit(13, 30), "Prog II");
4 m2.reserviere(r3, new Uhrzeit(9, 30), new Uhrzeit(11,30), "ITM");
```

dann soll die Raumausgabe von `System.out.println(r1)` nun zusätzliche Reservierungsinformationen beinhalten:

```
Raum 18-0.1
gebucht von Max Mustermann (mm@ex.com) von 12:30 Uhr bis 14:30 Uhr für VOOP
```

Auch die Ausgabe von `System.out.println(r2)` soll dann diese Ausgabe erzeugen:

```
Raum 2-1.9
gebucht von Max Mustermann (mm@ex.com) von 14:30 Uhr bis 16:30 Uhr für WebTech
gebucht von Tessa Loniki (loniki@ex.com) von 12:30 Uhr bis 13:30 Uhr für Prog II
```

Gleiches gilt für die Ausgabe von `System.out.println(r3)`:

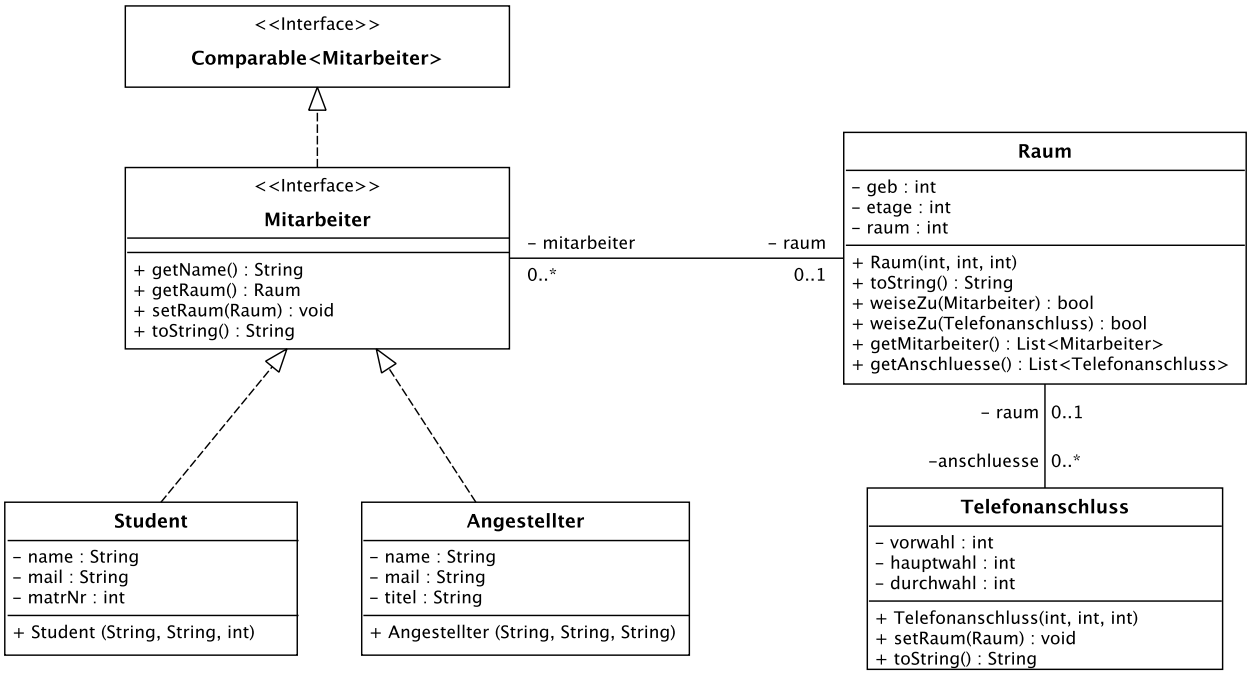
```
Raum 2-1.10
gebucht von Tessa Loniki (loniki@ex.com) von 9:30 Uhr bis 11:30 Uhr für ITM
```

Setzen Sie dieses Diagramm unter Berücksichtigung der Implementierungshinweise sowie der folgenden ergänzenden Hinweise bitte in Java Code um.

1. Beachten Sie bitte, dass Sie offenbar über Assoziationen von Räumen auf Mitarbeiter schließen können müssen. Diese Beziehungen müssen also mittels geeigneter Methoden verwaltet werden.
2. Das UML Diagram ist vollständig. Sie müssen keine Methoden/Datenfelder hinzufügen oder anpassen, um oben gezeigtes Verhalten realisieren zu können!
3. Sie müssen keine Doppelbelegungen erkennen oder zeitlichen Sortierungen bei den Ausgaben vornehmen.
4. Sie müssen ebenso keine Reservierungsrücknahmen, Umbuchungen oder ähnliches berücksichtigen.

17.4. Aufgabe: UML-Beispiel II (Telefonbuch)

Gegegeben ist folgendes Klassendiagramm.



Ergänzend sind Ihnen folgende Hinweise für die Implementierung gegeben.

Ein Telefonanschluss soll wie folgt angelegt werden können.

```
1 Telefonanschluss t = new Telefonanschluss(451, 300, 5549);
```

und mittels `System.out.println(t)` wie folgt ausgegeben werden können.

```
0451-300 5549
```

Ein Angestellter soll wie folgt angelegt

```
1 Mitarbeiter m = new Angestellter("Prof. Dr.", "Musterfrau", "musterfrau@fhl.de");
```

und mittels `System.out.println(m)` wie folgt ausgegeben werden können.

```
Prof. Dr. Musterfrau (musterfrau@fhl.de)
```

Ein Student soll wie folgt angelegt

```
1 Mitarbeiter s = new Student("Max Mustermann", "max.mustermann@stud.fhl.de",
      123456);
```

und mittels `System.out.println(s)` wie folgt ausgegeben werden können.

```
Max Mustermann (max.mustermann@stud.fhl.de)
```

Ein Raum soll wie folgt angelegt

```
Raum r = new Raum(18, 0, 1);
```

und mittels `System.out.println(r)` wie folgt ausgegeben werden können:

```
18-0.1
```

Mitarbeiter können einen Raum zugewiesen bekommen. Bspw. wie folgt:

```
Mitarbeiter m = new Angestellter("Prof. Dr.", "Musterfrau", "musterfrau@fhl.de");
Mitarbeiter s = new Student("Max Mustermann", "max.mustermann@stud.fhl.de",
    123456);
Raum r = new Raum(18, 0, 1);
r.weiseZu(m);
r.weiseZu(s);
```

Räume können mehrere Telefonanschlüsse haben. Bspw. wie folgt:

```
Telefonanschluss t1 = new Telefonanschluss(451, 300, 4321);
Telefonanschluss t2 = new Telefonanschluss(451, 300, 1234);
Raum r = new Raum(18, 0, 1);
r.weiseZu(t1);
r.weiseZu(t2);
```

Beachten Sie, dass Mitarbeiter und Telefonanschlüsse einem Raum nur einmal zugewiesen werden können. Wird ein Mitarbeiter oder ein Telefonanschluss (irrtümlich) mehrmals demselben Raum zugewiesen, ist nur die erste Zuweisungsoperation erfolgreich.

Wenn ihnen eine Liste von Mitarbeitern gegeben ist, sollen sie nun ein **alphabetisch sortiertes Telefonbuch** mittels folgender Methode ausgeben können. Beachten Sie, dass im vorliegenden Beispiel Mitarbeiter durchaus über mehrere Telefonanschlüsse erreichbar sind.

```
String telefonbuch(List<Mitarbeiter> mas)
```

Im oben gezeigten Beispiel sollte folgende Ausgabe erfolgen:

```
Max Mustermann (max.mustermann@stud.fhl.de)
- 0451-300 4321 (18-0.1)
- 0451-300 1234 (18-0.1)
Prof. Dr. Musterfrau (musterfrau@fhl.de)
- 0451-300 4321 (18-0.1)
- 0451-300 1234 (18-0.1)
```

Das muss natürlich auch mit mehr als einem Raum funktionieren.

18. Klassenhierarchien

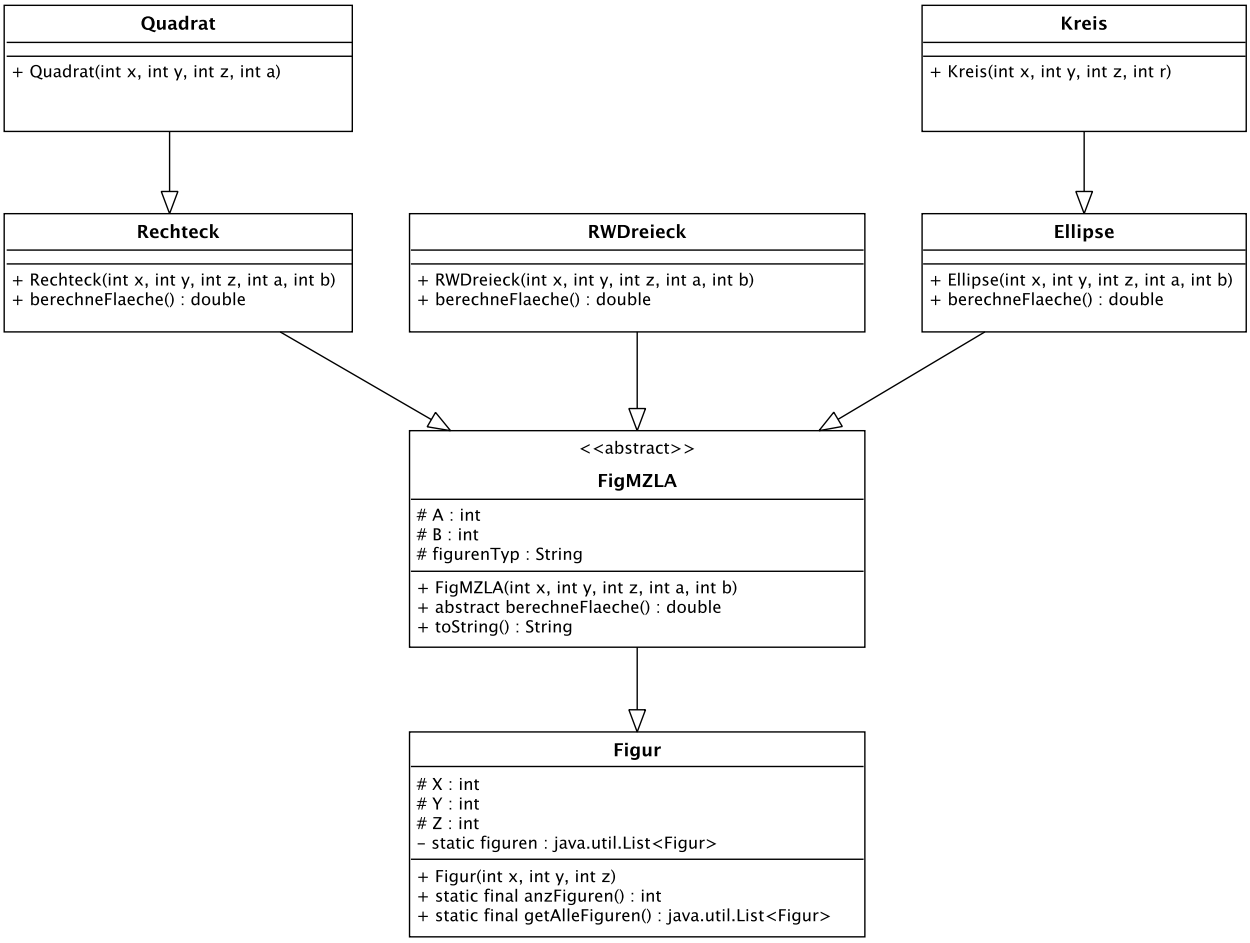
18.1. Aufgabe: Figurenhierachie

Sie sollen nun eine Basisbibliothek für ein Vektor-Grafikprogramm entwickeln. Das Grafikprogramm kennt als Grundformen Rechtecke, Quadrate, Ellipsen, rechtwinklige Dreiecke und Kreise.

Ihre Grafikbibliothek soll in einer Druckerei genutzt werden. Unter anderem ist es nun erforderlich zu bedruckende Flächeneinheiten zu berechnen, um die erforderliche Menge an Toner im Vorfeld großer Druckaufträge berechnen und bestellen zu können.

Ferner soll zusätzlich zu diesen Grundformen neben den x- und y-Angaben noch ein z-Index gespeichert werden, der es ermöglicht Figuren auf unterschiedliche Ebenen zu legen.

Zur Lösung dieser Aufgabe wird ihnen folgendes UML-Diagramm bereitgestellt.



Die folgenden Codezeilen

```
new Rechteck(5, 5, 3, 10, 20);
new RWDreieck(30, 2, 4, 3, 4);
new Ellipse(4, 10, 5, 10, 20);
new Kreis(10, 10, 6, 5);
new Quadrat(100, 200, -10, 35);

for (Figur f : Figur.getAlleFiguren()) {
  System.out.println(f);
}
```

sollten dann folgendes Ergebnis zeigen.

```
Rechteck an Position (?, ?, ?) mit einer Fläche von ??? Flächeneinheiten.
Rechtwinkliges Dreieck an Position (?, ?, ?) mit einer Fläche von ??? Flächeneinheiten.
Ellipse an Position (?, ?, ?) mit einer Fläche von ??? Flächeneinheiten.
Kreis an Position (?, ?, ?) mit einer Fläche von ??? Flächeneinheiten.
Quadrat an Position (?, ?, ?) mit einer Fläche von ??? Flächeneinheiten.
```

Die ? bezeichnen die durch Sie hoffentlich korrekt berechneten und/oder ausgegebenen Werte.

18.2. Aufgabe: Ebenensortierung von Figuren

In der Druckerei hat man festgestellt, dass die Ausdrucke der Figuren immer wieder unterschiedlich sind. Die Überlagerungen sich überlappender Figuren hängen von der Reihenfolge ab, wie sie durch das Programm erzeugt wurden. Dies ist nicht gewollt. Grafiker brauchen mehr Kontrolle über Ihr Layout. Daher soll nun der z-Index herangezogen werden, um die Druckreihenfolge der Figuren festzulegen. Es soll zuerst die Figur mit dem kleinsten z-Index gedruckt werden, anschließend alle anderen aufsteigend nach ihrem z-Index sortiert.

Denken Sie bei Ihrer Lösung über den Einsatz des Ihnen bereits bekannten `bubbleSort` Algorithmus nach.

```
static void bubbleSort(int[] xs) {
  boolean unsorted=true;
  while (unsorted) {
    unsorted = false;
    for (int i=0; i < xs.length-1; i++) {
      if (!(xs[i] <= xs[i+1])) {
        int dummy = xs[i];
        xs[i] = xs[i+1];
        xs[i+1] = dummy;
        unsorted = true;
      }
    }
  }
}
```

Nutzen Sie die JAVA API, um herauszufinden, wie man aus einer `List` ein `Array` über `Object` macht und bedenken Sie erforderliche explizite Down-Casts.

`http://download.oracle.com/javase/6/docs/api/java/util/List.html#toArray()`

Bringen Sie Ihre Erweiterungen nun in die Musterlösung der letzten Übung ein.

Die folgenden Codezeilen

```
new Rechteck(5, 5, 3, 10, 20);
new RWDreieck(30, 2, 4, 3, 4);
new Ellipse(4, 10, 10, 10, 20);
new Kreis(10, 10, 6, 5);
new Quadrat(100, 200, -10, 35);

for (Figur f : Figur.getZsortierteFiguren()) {
  System.out.println(f);
}
```

sollten dann folgendes Ergebnis zeigen.

```
Quadrat an Position (?, ?, -10) mit einer Fläche von ??? Flächeneinheiten.
Rechteck an Position (?, ?, 3) mit einer Fläche von ??? Flächeneinheiten.
Rechtwinkliges Dreieck an Position (?, ?, 4) mit einer Fläche von ??? Flächeneinheiten.
Kreis an Position (?, ?, 6) mit einer Fläche von ??? Flächeneinheiten.
Ellipse an Position (?, ?, 10) mit einer Fläche von ??? Flächeneinheiten.
```

Die ? bezeichnen die durch Sie hoffentlich korrekt berechneten und/oder ausgegebenen Werte.

18.3. Aufgabe: Gefilterte Figurenausgaben

Ihre Grafikbibliothek wird in einer Druckerei genutzt. Ferner soll es nun ergänzend möglich sein, frei konfigurierbar den Ebenenbereich (z-Index) anzugeben, innerhalb dessen ausgedruckt werden soll. So ist es möglich Ebenen für den Ausdruck auszublenden, z. B. eine Draft- und Wasserzeichenebene (höchste und tiefste Ebene).

Entwickeln Sie nun bitte eine statische Methode `filterZ` für die Klasse Figur, so dass folgende Codezeilen

```
new Rechteck(5, 5, -10, 10, 20);
new RWDreieck(30, 2, 80, 3, 4);
new Ellipse(4, 10, 31, 10, 20);
new Kreis(10, 10, 31, 5);
new Quadrat(100, 200, 13, 35);
new Rechteck(5, 5, 7, 10, 20);
new RWDreieck(30, 2, -11, 3, 4);
new Ellipse(4, 10, -17, 10, 20);
new Kreis(10, 10, 50, 5);
new Quadrat(100, 200, 30, 35);
new Rechteck(5, 5, 25, 10, 20);
new RWDreieck(30, 2, 7, 3, 4);
new Ellipse(4, 10, 4, 10, 20);
new Kreis(10, 10, 10, 5);
new Quadrat(100, 200, 5, 35);

for (Figur f : Figur.filterZ(0, 50, Figur.getZsortierteFiguren())) {
  System.out.println(f);
}
```

in etwa die folgende Ausgabe produzieren:

```
Ellipse an Position (4, 10, 4) mit einer Fläche von 628.3185307179587 Flächeneinheiten
Quadrat an Position (100, 200, 5) mit einer Fläche von 1225.0 Flächeneinheiten
Rechteck an Position (5, 5, 7) mit einer Fläche von 200.0 Flächeneinheiten
Rechtwinkliges Dreieck an Position (30, 2, 7) mit einer Fläche von 6.0 Flächeneinheiten
Kreis an Position (10, 10, 10) mit einer Fläche von 78.53981633974483 Flächeneinheiten
Quadrat an Position (100, 200, 13) mit einer Fläche von 1225.0 Flächeneinheiten
Rechteck an Position (5, 5, 25) mit einer Fläche von 200.0 Flächeneinheiten
Quadrat an Position (100, 200, 30) mit einer Fläche von 1225.0 Flächeneinheiten
Ellipse an Position (4, 10, 31) mit einer Fläche von 628.3185307179587 Flächeneinheiten
Kreis an Position (10, 10, 31) mit einer Fläche von 78.53981633974483 Flächeneinheiten
Kreis an Position (10, 10, 50) mit einer Fläche von 78.53981633974483 Flächeneinheiten
```

18.4. Aufgabe: Flächenberechnung für gefilterte Figurenausgaben

Bestimmen Sie für die Figuren

```
1  new Rechteck(5, 5, -10, 10, 20);
2  new RWDreieck(30, 2, 80, 3, 4);
3  new Ellipse(4, 10, 31, 10, 20);
4  new Kreis(10, 10, 31, 5);
5  new Quadrat(100, 200, 13, 35);
6  new Rechteck(5, 5, 7, 10, 20);
7  new RWDreieck(30, 2, -11, 3, 4);
8  new Ellipse(4, 10, -17, 10, 20);
9  new Kreis(10, 10, 50, 5);
10 new Quadrat(100, 200, 30, 35);
11 new Rechteck(5, 5, 25, 10, 20);
12 new RWDreieck(30, 2, 7, 3, 4);
13 new Ellipse(4, 10, 4, 10, 20);
14 new Kreis(10, 10, 10, 5);
15 new Quadrat(100, 200, 5, 35);
```

mit einem **z-Index** zwischen **0 und 50** bitte

1. die zu bedruckende Gesamtfläche (es sind keine Überlappungen zu berücksichtigen, d.h. mehrfach bedruckte Flächen sind auch mehrfach zu zählen)
2. die durchschnittliche Fläche pro Figur.

Sie sollten in etwa folgende Ausgabe produzieren:

```
Die Gesamtfläche aller ? Figuren beträgt ??? Flächeneinheiten.
Die durchschnittliche Fläche pro Figur beträgt ??? Flächeneinheiten.
```

Teil VIII.

Aufgaben zu Unit 8
Testen (objektorientierter) Software

19. Unittesting und Codecoverage

Um diese Aufgaben lösen zu können, sollten sie sich folgende Tools für Ihre Eclipse Programmierumgebung installieren:

- EclEmma (`http://www.eclemma.org`, auch installierbar über den Eclipse Marketplace [Help → Eclipse Marketplace])
- JUnit4 (Standardmäßig bei Eclipse installiert, aber auch installbierbar über den Eclipse Marketplace)

19.1. Aufgabe: Buchstabenzählen testen

Gegeben ist folgende Methode `countChars`:

```
1  /**
2   * Zaehlt die Vorkommensanzahl von Zeichen innerhalb eines Strings.
3   * @param insensitive Soll case insensitive (true) gezaehlt werden oder nicht (false)?
4   * @param str Zu durchsuchender String.
5   * @param cs Zu zaehlende Zeichen innerhalb von str.
6   * @return Map<Character, Integer> Gezaehlte Vorkommen als
7   *       Mapping von Zeichen auf Haeufigkeit.
8   */
9  public static Map<Character, Integer> countChars(
10   boolean insensitive,
11   String str,
12   char... cs
13 ) {
14   Map<Character, Integer> ret = new HashMap<Character, Integer>();
15   for (char c : cs) {
16     ret.put(c, 0);
17   }
18
19   if (insensitive) {
20     str = str.toLowerCase();
21   }
22
23   int l = str.length();
24   for (int i = 1; i < l - 1; i++) {
25     for (char c : cs) {
26       if (!insensitive) {
27         if (c == str.charAt(i)) {
28           ret.put(c, ret.get(c) + 1);
29         }
30       } else {
31         if (c == str.charAt(i)) {
32           ret.put(c, ret.get(c) + 1);
33           ret.put(Character.toUpperCase(c), ret.get(c));
34         }
```

```
35        }
36      }
37    }
38    return ret;
39  }
```

Diese zählt Zeichenvorkommen in Zeichenketten. Über einen Parameter kann eingeschränkt werden, ob dies case senstive erfolgen soll, oder nicht.

Es wurden bereits folgende Unittests entwickelt:

```
1  import static org.junit.Assert.*;
2  import java.util.Map;
3  import org.junit.Test;
4
5  /**
6   * Testfaelle, die die Implementierung abpruefen sollen.
7   * Vorgabe der Aufgabe 19.1
8   * @author Nane Kratzke
9   *
10  */
11 public class Aufgabe19_1TestIncomplete {
12
13    /**
14   * Pruefe ob countChars bei einem typischen Aufruf funktioniert.
15   */
16   @Test
17   public void testCaseInsensitive() {
18     Map<Character, Integer> m = Aufgabe19_1WithErrors.countChars(false, "Hello World", 'o',
         'e');
19
20     // Pruefe die Struktur der Rueckgabe
21     assertTrue(m.keySet().size() == 2);
22     assertTrue(m.keySet().contains('o'));
23     assertTrue(m.keySet().contains('e'));
24
25     // Pruefe Inhalt der Rueckgabe
26     assertTrue(m.get('o') == 2);
27     assertTrue(m.get('e') == 1);
28   }
29
30   /**
31    * Pruefe ob countChars case sensitive korrekt verarbeitet.
32    */
33   @Test
34   public void testCaseSensitive() {
35     Map<Character, Integer> m = Aufgabe19_1WithErrors.countChars(true, "Hello World", 'o', '
         e', '!');
36
37     assertTrue(m.get('o') == 2);
38     assertTrue(m.get('O') == 2);
39     assertTrue(m.get('e') == 1);
40     assertTrue(m.get('E') == 1);
41     assertTrue(m.get('!') == 0);
42   }
43
44 }
```

1. Führen Sie diese beiden Testfälle mit JUnit durch und stellen Sie die Statement Coverage für die oben stehende Methode mittels des Eclipse Plugins EclEmma (installieren Sie dieses ggf. über den Eclipse Marketplace nach) fest. Glauben Sie den Testfällen bei der angezeigten Statement Coverage?
2. Entwickeln sie nun weitere Unittests, um herauszufinden, ob diese Methode tatsächlich korrekt funktioniert.
   - Die Methode hat hinsichtlich ihrer Eingabeparameter keinerlei Einschränkungen, d.h. alle syntaktisch korrekten und denkbaren Eingaben erfüllen die **Vorbedingung**.
   - Als **Nachbedingung** soll die Methode eine Map liefern, die die in einem `String str` zu zählenden Zeichen `a, b, ...` auf ihre Häufigkeit abbildet und hinsichtlich ihres `keySets` der folgenden Bedingung für `Strings str` und `char a, b, ...` genügt (nachfolgender Code ist Pseudocode, kein lauffähiges Java).

     ```
     Map<Character, Integer> result_cs = countChars(false, str, a, b, ...)
     assertTrue(result_cs.keySet().equals(new HashSet<Character>(a, b, ...)))
     Map<Character, Integer> result_ncs = countChars(true, str, a, b, ...)
     assertTrue(result_ncs.keySet().equals(new HashSet<Character>(
         Character.toLowerCase(a), Character.toLowerCase(b), ...,
         Character.toUpperCase(a), Character.toUpperCase(b), ...,
     )))
     ```
3. Korrigieren Sie die Methode, so dass Sie tatsächlich korrekt funktioniert. Beachten Sie dabei vor allem primitive und triviale Sonderfälle. Geben Sie bitte als **Abgabe** ein Eclipse Archiv mit Ihren ergänzten Unittests und ihrer korrigierten `countChars` Methode ab. Kommentieren Sie dabei deutlich, die geänderten Quelltextpassagen. Kommentieren Sie die Originalquelltextpassage aus und setzen sie darunter ihren korrigierten Quelltext.

19.2. Aufgabe: Testfallerstellung zur Erhöhung von Coverages

Gegeben ist folgende Klasse zum Aufbau eines Binärbaums:

```
1  import java.util.LinkedList;
2  import java.util.List;
3  import java.util.Random;
4
5  /**
6   * Ein BugTree Objekt dient zur Repraesentation von Binaerbaeumen.
7   * Aus mehreren Knoten kann ein Binaerbaum gebildet werden.
8   * Ein Knoten in einem Binaerbaum hat einen Wert value, sowie einen
9   * linken Ast left und einen rechten Ast right.
10  * Der Wert eines Knotens kann von beliebigem Typ sein,
11  * muss aber mittels der Schnittstelle Comparable verglichen
12  * werden koennen.
13  * Binaerbaeume koennen mit den Methoden ascending und descending
14  * in Listen serialisiert werden, in dem der Binaerbaum inorder
15  * nach dem Schema LKR bzw. RKL durchlaufen wird.
16  * @author Nane Kratzke
17  */
18 public class BugTree<T extends Comparable<T>> {
```

```
19
20   /**
21    * Wert des Knotens.
22    */
23   private T value;
24
25   /**
26    * Verweis auf den linken Ast des Knotens.
27    */
28   private BugTree<T> left;
29
30   /**
31    * Verweis auf den rechten Ast des Knotens.
32    */
33   private BugTree<T> right;
34
35   /**
36    * Konstruktor zum Erzeugen eines Wurzelnotens mit dem Wert v.
37    * Der linke und rechte Teilast dieses Binaerbaums sind immer null.
38    * @param v Wert des Knotens
39    */
40   public BugTree(final T v) { this.value = v; }
41
42   /**
43    * Konstruktur zum Erzeugen eines sortierten Binaerbaums aus einer Liste.
44    * @param list Liste von Werten aus denen der sortierte Binaerbaum gebildet werden soll
45    *             (list != null und list.size() >= 1)
46    */
47   public BugTree(final List<T> list) {
48     Random rand = new Random();
49
50     List<T> insert = list;
51     this.value = insert.remove(0);
52
53     // Wir durchlaufen die Liste in zufaelliger Reihenfolge,
54     // um ggf. bereits vorsortierte Listen zu durchbrechen
55     // und so die Leistungsfaehigkeit des BinSort (im Mittel) zu erhalten.
56     while (!insert.isEmpty()) {
57       T v = insert.remove(rand.nextInt(insert.size()));
58       this.insert(v);
59     }
60   }
61
62   /**
63    * Liefert den linken Ast des Knotens.
64    * @return Linker Ast
65    */
66   public BugTree<T> getLeft() { return this.left; }
67
68   /**
69    * Haengt einen Baum unter den linken Ast.
70    * @param tree Baum der unter den linken Ast gehaengt werden soll
71    */
72   public void setLeft(final BugTree<T> tree) { this.left = tree; }
73
74   /**
```

```
75     * Liefert den rechten Ast des Knotens.
76     * @return Rechter Ast
77     */
78    public BugTree<T> getRight() { return this.right; }
79
80    /**
81     * Haengt einen Baum unter den rechten Ast.
82     * @param tree Baum der unter den rechten Ast gehaengt werden soll
83     */
84    public void setRight(final BugTree<T> tree) { this.right = tree; }
85
86    /**
87     * Liefert den Wert des Knotens.
88     * @return Wert des Knotens.
89     */
90    public T value() { return this.value; }
91
92    /**
93     * Serialisiert einen Binaerbaum in eine lineare Liste
94     * in dem der Binaerbaum inorder (LKR) durchlaufen wird.
95     * @return Aus Binaerbaum serialisierte Liste von Werten nach dem LKR Durchlauf
96     */
97    public List<T> ascending() {
98      List<T> ret = new LinkedList<T>();
99      if (this.getLeft() != null) { ret.addAll(this.getLeft().ascending()); }
100     ret.add(this.value());
101     if (this.getRight() != null) { ret.addAll(this.getRight().ascending()); }
102     return ret;
103   }
104
105   /**
106    * Serialisiert einen Binaerbaum in eine sortierte lineare Liste
107    * in dem der Binaerbaum inorder (RKL) durchlaufen wird.
108    * @return Aus Binaerbaum serialisierte Liste von Werten nach dem RKL Durchlauf
109    */
110   public List<T> descending() {
111     List<T> ret = new LinkedList<T>();
112     if (this.getRight() != null) { ret.addAll(this.getRight().descending()); }
113     ret.add(this.value());
114     if (this.getLeft() != null) { ret.addAll(this.getLeft().descending()); }
115     return ret;
116   }
117
118   /**
119    * Fuegt den Wert v in den Binaerbaum ein.
120    * @param v Einzufuegender Wert
121    */
122   public void insert(final T v) {
123     if (v.compareTo(this.value()) < 0) {
124       if (this.getLeft() == null) { this.setLeft(new BugTree<T>(v)); }
125       else { this.getLeft().insert(v); }
126     }
127
128     if (v.compareTo(this.value()) > 0) {
129       if (this.getRight() == null) { this.setRight(new BugTree<T>(v)); }
130       else { this.getRight().insert(v); }
```

```
    }
  }
}
```

und folgende Klasse zum Sortieren von Listen:

```
import java.util.List;

/**
 * Die Klasse BugSort soll diverse Sortieralgorithmen bereitstellen.
 * Aktuell wird nur der Binaersort Algorithmus angeboten.
 * @author Nane Kratzke
 */
public class BugSort<T extends Comparable<T>> {

  /**
   * Sortiert eine Liste von Werten nach dem Binaersort Algorithmus.
   * @param list Zu sortierende Liste (list != null)
   * @return Aufsteigend sortierte Liste
   */
  public List<T> binSort(final List<T> list) { return binSort(list, true); }

  /**
   * Sortiert eine Liste von Werten nach dem Binaersort Algorithmus
   * aufsteigend oder absteigend.
   * @param list Zu sortierende Liste (list != null)
   * @param asc Soll aufsteigend oder absteigend sortiert werden
   * @return Aufsteigend sortierte Liste (wenn asc == true)
   *         Absteigend sortierte Liste (wenn asc == false)
   */
  public List<T> binSort(final List<T> list, boolean asc) {
    BugTree<T> bintree = new BugTree<T>(list);
    return asc ? bintree.ascending() : bintree.descending();
  }
}
```

Gegeben sind ferner folgende Testfälle:

```
import static org.junit.Assert.*;
import java.util.LinkedList;
import java.util.List;
import org.junit.Test;

/**
 * Testet den in der Klasse Sort befindlichen BinSort Algorithmus.
 * Vorgabe fuer Aufgabe 19.2
 * @author Nane Kratzke
 */
public class BugSortTest {

  /**
   * Testet aufsteigenden BinSort mit einer Liste von Strings.
   */
  @Test
  public void testBinSortWithStrings() {
    List<String> strs = new LinkedList<String>();
```

```
19      strs.add("Hello World");
20      strs.add("Mein Name ist Hase");
21      strs.add("Das wird schon klappen");
22
23      BugSort<String> stringSorter = new BugSort<String>();
24      List<String> sorted = stringSorter.binSort(strs);
25
26      assertTrue(sorted.get(0).equals("Das wird schon klappen"));
27      assertTrue(sorted.get(1).equals("Hello World"));
28      assertTrue(sorted.get(2).equals("Mein Name ist Hase"));
29    }
30
31    /**
32     * Testet aufsteigenden BinSort mit einer Liste von Integern.
33     */
34    @Test
35    public void testBinSortWithIntegers() {
36      List<Integer> is = new LinkedList<Integer>();
37      for (int i = 0; i <= 5; i++) is.add(i);
38
39      BugSort<Integer> integerSorter = new BugSort<Integer>();
40      List<Integer> sorted = integerSorter.binSort(is);
41
42      // Wir pruefen die aufsteigende Sortierung
43      for (int i = 0; i < sorted.size() - 1; i++) {
44        assertTrue(sorted.get(i) <= sorted.get(i + 1));
45      }
46    }
47  }
```

1. Bestimmen Sie die Codecoverage mittels EclEmma. Entwickeln Sie möglichst wenig weitere Testfälle, um zu einer 100% Testabdeckung (Statement und Branch Coverage) für die Klassen `BugTree` und `BugSort` zu gelangen.
2. Beantworten Sie so die Frage, ob die Methode `binSort` tatsächlich korrekt funktioniert.
3. Korrigieren Sie ggf. die Sortierlogik, so dass `binSort` tatsächlich korrekt funktioniert. Beachten Sie dabei vor allem Sonderfälle. Geben Sie bitte als **Abgabe** ein Eclipse Archiv mit Ihren ergänzten Unittests und ihrer korrigierten `BugTree` oder `BugSort` Klasse ab. Kommentieren Sie dabei deutlich, die geänderten Quelltextpassagen. Kommentieren Sie die Originalquelltextpassage aus und setzen sie darunter ihren korrigierten Quelltext.

Teil IX.

Aufgaben zu Unit 9
Generische Datentypen

20. Generische Datentypen

20.1. Aufgabe: Entwickeln einer generischen Warteschlange

Lesen Sie sich bitte in das Konzept der Datenstruktur Warteschlange ein

- `http://de.wikipedia.org/wiki/Warteschlange_(Datenstruktur)`

und entwickeln Sie eine generische Warteschlange `MyQueue` mit dem generischen Typparameter `T`. Die Datenstruktur `MyQueue` soll die folgenden Methoden implementieren:

- `boolean enter(T)` fügt ein neues Element der Warteschlange hinzu. Liefert `true`, wenn das Element der Warteschlange hinzugefügt werden konnte, andernfalls `false`.
- `T leave() throws NoSuchElementException` entnimmt das erste Element der Warteschlange. Ist die Warteschlange leer wirft die Methode eine `java.util.NoSuchElementException`.
- `boolean isEmpty` prüft ob die Warteschlange leer ist
- `T front()` liest das erste Element der Warteschlange, belässt es aber in der Warteschlange. Liefert `null`, wenn sich kein Element in der Warteschlange befindet.
- `String toString()` die eine eine textuelle Repräsentation der Warteschlange in folgender Form liefert: `[e1, e2, e3, e4]` wenn
  - `e1`, ..., `e4` Elemente der Warteschlange sind und
  - `e1` das hinterste Element und
  - `e4` das vorderste Element der Warteschlange ist.

Testen Sie bitte ihre generische Datenstruktur `MyQueue` mittels der folgenden Codezeilen. Diese bauen eine initiale Warteschlange auf und simulieren den zufälligen Eintritt eines Kunden in die Warteschlange und die zuverlässige Abarbeitung des vordersten Kunden in der Warteschlange – solange bis diese leer ist.

```
1  import java.util.Random;
2
3  public class MyQueueTest {
4
5    public static void main(String[] args) {
6      MyQueue<String> myq = new MyQueue<String>();
7
8      myq.enter("Max Mustermann");
9      myq.enter("Maren Musterfrau");
10     myq.enter("Tessa Loniki");
11     myq.enter("Hans Hampelmann");
12
13     System.out.println("Initiale Warteschlange: " + myq);
14     System.out.println("Erster in der Warteschlange: " + myq.front());
15     System.out.println("");
16
```

```
17      int i = 1;
18
19      while(!myq.isEmpty()) {
20        System.out.println("Verarbeite Kunde: " + myq.leave());
21        System.out.println("Verbleibende Warteschlange: " + myq);
22        System.out.println("");
23
24        if ((new Random()).nextInt(10) < 4) {
25          String newKunde = "Karl König " + i++;
26          System.out.println("Ankunft eines neuen Kunden: " + newKunde);
27          myq.enter(newKunde);
28        }
29      }
30
31      System.out.println("Noch jemand in der Warteschlange? " + myq.front());
32    }
33  }
```

Diese sollten in etwa folgende Ausgabe erzeugen. Da das Hinzukommen von neuen Kunden zufällig bestimmt wird, ist die nachfolgende Ausgabe natürlich rein exemplarisch. Ihre Ausgabe wird mit hoher Wahrscheinlichkeit anders sein und sich von Applikationsaufruf zu Applikationsaufruf unterscheiden.

```
Initiale Warteschlange: [Hans Hampelmann, Tessa Loniki, Maren Musterfrau, Max Mustermann]
Erster in der Warteschlange: Max Mustermann

Verarbeite Kunde: Max Mustermann
Verbleibende Warteschlange: [Hans Hampelmann, Tessa Loniki, Maren Musterfrau]

Ankunft eines neuen Kunden: Karl König 1
Verarbeite Kunde: Maren Musterfrau
Verbleibende Warteschlange: [Karl König 1, Hans Hampelmann, Tessa Loniki]

Ankunft eines neuen Kunden: Karl König 2
Verarbeite Kunde: Tessa Loniki
Verbleibende Warteschlange: [Karl König 2, Karl König 1, Hans Hampelmann]

Verarbeite Kunde: Hans Hampelmann
Verbleibende Warteschlange: [Karl König 2, Karl König 1]

Ankunft eines neuen Kunden: Karl König 3
Verarbeite Kunde: Karl König 1
Verbleibende Warteschlange: [Karl König 3, Karl König 2]

Verarbeite Kunde: Karl König 2
Verbleibende Warteschlange: [Karl König 3]

Ankunft eines neuen Kunden: Karl König 4
Verarbeite Kunde: Karl König 3
Verbleibende Warteschlange: [Karl König 4]

Verarbeite Kunde: Karl König 4
```

```
Verbleibende Warteschlange: []
Ankunft eines neuen Kunden: Karl König 5
Verarbeite Kunde: Karl König 5
Verbleibende Warteschlange: []

Noch jemand in der Warteschlange? null
```

Schreiben Sie nun den oben angegebenen Testcode so um, dass auch die Abarbeitung eines Kunden in der Warteschlange dem Zufall überlassen bleibt.

- Spielen Sie ein wenig mit den Ankunfts- und Abarbeitungswahrscheinlichkeiten der Kunden herum.
- Sie haben nun zwei Zufallskomponenten in ihr Programm eingebaut. Ist es nun sicher, dass das Programm terminiert?
- Welche Bedingung für die Ankunfts- und Abarbeitungswahrscheinlichkeit muss gelten, damit das Programm höchstwahrscheinlich terminiert? Versuchen Sie sich die Sachverhalte an der Kasse eines Supermarkts mit ankommenden und abgehenden Kunden zu veranschaulichen.

20.2. Aufgabe: Entwickeln eines generischen Binärsorts

In der Vorlesung wurde das Prinzip des Sortierens mit einem Binärbaum behandelt. Sie sollen nun auf Basis dieser Kenntnisse eine generische BinSort Methode mit folgender Signatur

```
<T extends Comparable<T>> List<T> bsort(List<T> is)
```

entwickeln. Die in der Vorlesung (Semester 1) gezeigte Variante eines Binärbaums ging davon aus, dass keine doppelten Elemente in einem Binärbaum sind. In dieser Aufgabe sollen Sie jedoch auch Listen sortieren können, in denen Elemente mehrmals vorkommen können.

Sie müssen hierzu vermutlich

- eine generische Klasse `BTree<T extends Comparable<T>>` entwickeln, um Binärbäume aufbauen zu können (und etwas von der in der Vorlesung behandelten Implementierung abwandeln),
- eine rekursive Methode `void insert(T element)` definieren, um Elemente in einen Binärbaum einfügen zu können, sowie eine
- rekursive Methode `List<T> buildInOrderList(BTree<T> tree)` definieren, um einen Binärbaum in einem Inorder Durchlauf durchlaufen und so aus den Knoten des Baums eine Liste bilden zu können.

Für eine Testklasse `MyObject`

```
1 class MyObject implements Comparable<MyObject> {
2   int value;
3   public MyObject(int v) { this.value = v; }
4   public int compareTo(MyObject o) { return this.value - o.value; }
5   public String toString() { return "" + value; }
6 }
```

könnte folgender Testcode

```
1  import java.util.Collection;
2  import java.util.LinkedList;
3  import java.util.List;
4  import java.util.Random;
5
6  [...]
7
8  public class TestBinSort {
9
10   public static void main(String[] args) {
11     List<MyObject> values = new LinkedList<MyObject>();
12
13     Random r = new Random();
14     for (int i = 20; i >= 1; i--) {
15       values.add(new MyObject(r.nextInt(i)));
16     }
17
18     System.out.println("Unsortiert: " + values);
19     System.out.println("Sortiert: " + BTree.bsort(values)); }
20   }
21 }
```

dann folgende Ausgabe erzeugen:

```
Unsortiert: [8, 15, 15, 11, 5, 10, 1, 10, 9, 1, 9, 4, 6, 1, 3, 4, 1, 0, 1, 0]
Sortiert:   [0, 0, 1, 1, 1, 1, 1, 3, 4, 4, 5, 6, 8, 9, 9, 10, 10, 11, 15, 15]
```

Teil X.

Aufgaben zu Unit 10 Objektorientierter Entwurf und objektorientertes Design (am Beispiel von Tic Tac Toe)

21. Tic Tac Toe

21.1. Aufgabe: How to Play TIC TAC TOE

Im Rahmen der Vorlesung werden Ihnen objektorientierte Entwurfsprinzipien am Beispiel des Spiels TIC TAC TOE veranschaulicht. Nutzen Sie hierzu auch die Zeit, die Ihnen im Rahmen des Tutoriums ergänzend ermöglicht wird.

Befolgen Sie bitte die folgenden Schritte:

1. Laden Sie sich die TIC TAC TOE Engine als Eclipse Projektarchiv aus dem Moodle Kurs herunter (`T3Engine.zip`).
2. Importieren Sie dieses Projektarchiv wie gewohnt in Ihre Eclipse Programmierumgebung
3. Fügen Sie in ihrem Projekt, welches Sie für dieses Aufgabenblatt angelegt haben, nun die TIC TAC TOE Engine zu Ihrem Buildpath hinzu.
   a) Selektieren Sie Ihr Projekt im Package Explorer anschließend Rechte Maustaste -> Properties
   b) Selektieren Sie JAVA Build Path -> Projects
   c) Klicken Sie anschließend auf Add und fügen Sie ihr TIC TAC TOE Project hinzu

In der T3 Engine gibt es eine sogenannte `T3Starter` Klasse im Paket `de.fhl.oop.tictactoe.engine`. Diese dient dazu ein Spiel zwischen zwei Tic Tac Toe Spielern zu starten. Die T3 Engine bietet im Paket `de.fhl.oop.tictactoe.player` weiterhin die folgenden Spieler an:

- `CrazySpieler` - dieser Spieler sucht zeilenweise ein leeres Feld und setzt auf das erste das er findet
- `Katastrophenspieler` - dieser teilt bei jedem Spielzeug durch null und verliert daher immer (wurde nur für Testzwecke benötigt)
- `Zufallsspieler` - dieser setzt zufällig auf freie Felder
- und der abstrakte Spieler `T3VersierterSpieler` erweitert den `T3Spieler`, um zwei zusätzliche Methoden `leere_felder` und `gewinnfelder`, die es ihm ermöglichen, leere Felder und solche Felder zu bestimmen, die im nächsten Zug für einen der Spieler den Gewinn bringen würden. Diese beide Methoden machen es Ihnen einfacher intelligentere Strategien zu entwickeln.

In dieser Übung sollen Sie vorerst nur herausfinden, welche von den beiden Strategien des `CrazySpieler` und des `ZufallsSpieler` die bessere ist. Lassen Sie hierzu beide Spieler eine Partie mit jeweils 1000 Spielen durchspielen und schauen Sie, welche Spieler mehr Spiele gewonnen hat.

```
1 import de.fhl.oop.tictactoe.engine.T3Starter;
2 import de.fhl.oop.tictactoe.player.*;
3
4 public class Testspiel {
5   public static void main(String[] args) {
6     T3Starter.starte_partie(
7       1000,
```

```
8       new CrazySpieler("Crazy"),
9       new ZufallsSpieler("Zufall")
10    );
11  }
12 }
```

Welches ist die bessere Strategie?

- Auf das erste leere Feld zu setzen?
- Oder zufällig zu setzen?

Entwickeln Sie nun zu Hause und im Tutorium Ihre eigene Strategie. Entwickeln Sie Ihre eigene Spielklasse, in dem Sie diese von `T3VersierterSpieler` ableiten.

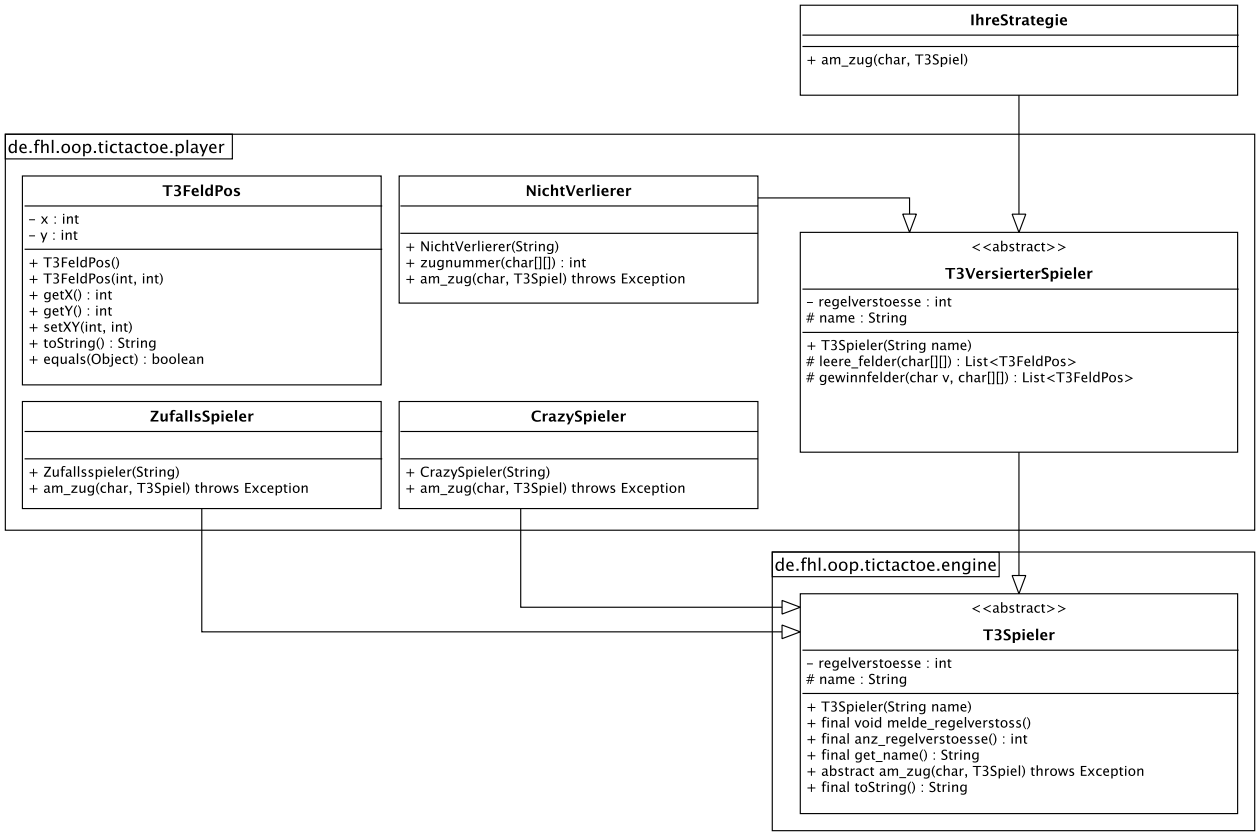
21.2. Aufgabe: Entwickeln einer eigenen TIC TAC TOE Strategie

Informieren Sie sich bitte unter

`http://www.wikihow.com/Win-at-Tic-Tac-Toe`

über optimale Spielstrategien für TIC TAC TOE.

Nachfolgendes UML Diagramm zeigt die Beziehungen zwischen bereits existierenden Spielstrategien.



Sie sollen nun Ihre eigene Spielstrategie entwickeln. Studieren Sie hierzu die Spielstrategieimplementierungen der Spieler

- `ZufallsSpieler` (setzt zufällig auf ein freies Feld)
- `CrazySpieler` (setzt auf das erste freie Feld beginnend oben rechts)
- `NichtVerlierer` (implementiert die in dem Link angegebene optimale Strategie zu einem Teil, d.h. diese Strategie ist immer noch schlagbar)

um ihre eigene Strategie zu implementieren. Leiten Sie hierzu ihre Strategie aus der Klasse `T3VersierterSpieler` ab, die ihnen die Methoden `leere_felder` und `gewinnfelder` anbietet. Mit der Methode

- `leere_felder` können Sie leere Felder bestimmen,
- mit der Methode `gewinnfelder` können Sie Felder bestimmen, in denen im nächsten Zug X oder O gewinnen würde, wenn es diese gibt.

Versuchen Sie zu verstehen, wie die Klasse `NichtVerlierer` diese zusätzlichen Felder nutzt.

Implementieren Sie nun eine Strategie, die besser ist, als

- `ZufallsSpieler`
- `CrazySpieler`

Testen Sie Ihre Strategie mit folgenden Codezeilen:

```
1  import de.fhl.oop.tictactoe.engine.T3Starter;
2  import de.fhl.oop.tictactoe.player.*;
3
4  public class Testspiel {
5
6    public static void main(String[] args) {
7      T3Starter.starte_partie(
8        1000,
9        new CrazySpieler("Crazy"),
10       new IhreStrategie("Ihr Name")
11     );
12     T3Starter.starte_partie(
13       1000,
14       new ZufallsSpieler("Zufall"),
15       new IhreStrategie("Ihr Name")
16     );
17   }
18 }
```

Erweitern Sie diese Strategie derart, dass Sie versuchen, die Strategie ihres Profs zu schlagen (`NichtVerlierer`). Da die `NichtVerlierer` Strategie nicht alle Aspekte der optimalen Strategie implementiert, ist sie zu schlagen!

Testen Sie Ihre Strategie mit folgenden Codezeilen:

```
1  import de.fhl.oop.tictactoe.engine.T3Starter;
2  import de.fhl.oop.tictactoe.player.*;
3
4  public class Testspiel {
5
6    public static void main(String[] args) {
7      T3Starter.starte_partie(
8        1000,
```

```
9       new NichtVerlierer("Prof. Kratzke"),
10      new IhreStrategie("Ihr Name")
11    );
12  }
13 }
```

Um zu prüfen, ob ihre Strategie auch wirklich die optimale Strategie ist, prüfen Sie, ob ihre Strategie **NIEMALS** gegen den Zufallsspieler verliert.

```
1  import de.fhl.oop.tictactoe.engine.T3Starter;
2  import de.fhl.oop.tictactoe.player.*;
3
4  public class Testspiel {
5
6    public static void main(String[] args) {
7      T3Starter.starte_partie(
8        100000, // Nehmen Sie hier ruhig einen großen Wert ...
9        new Zufallsspieler("Zufall"),
10       new IhreStrategie("Ihr Name")
11     );
12   }
13 }
```

Teil XI.

Aufgaben zu Unit 11
Graphical User Interfaces

22. GUI Programmierung

22.1. Aufgabe: Taschenrechner

Implementieren Sie den Taschenrechner wie in der Vorlesung vorgestellt. Nutzen Sie das Model-View-Controller Pattern.

Das Modell des Taschenrechners soll vier Register haben, sowie die in der Vorlesung gezeigten Operatoren zur Interaktionen mit den Taschenrechnermodell anbieten (`berechne`, `getOperand`, `setOperand`, `getOperator`, `setOperator`, `getError`, `clear`):

- **Result** zum Speichern von Rechenergebnissen
- **Operand** zum Speichern einer Eingabe (Operanden).
- **Operator** zum Speichern eines eingegeben Operators. Ein Operator kann +, -, * und / annehmen.
- **Error** zum Speichern des letzten aufgetretenen Fehlers

Der View des Taschenrechners soll in etwa folgendes Aussehen haben:

Beachten Sie bitte, dass fehlerhafte Eingaben durch den Bediener erfolgen können (z.B. durch Null teilen), aber das Programm nicht zum Absturz bringen dürfen.

22.2. Aufgabe: Bouncing Bears

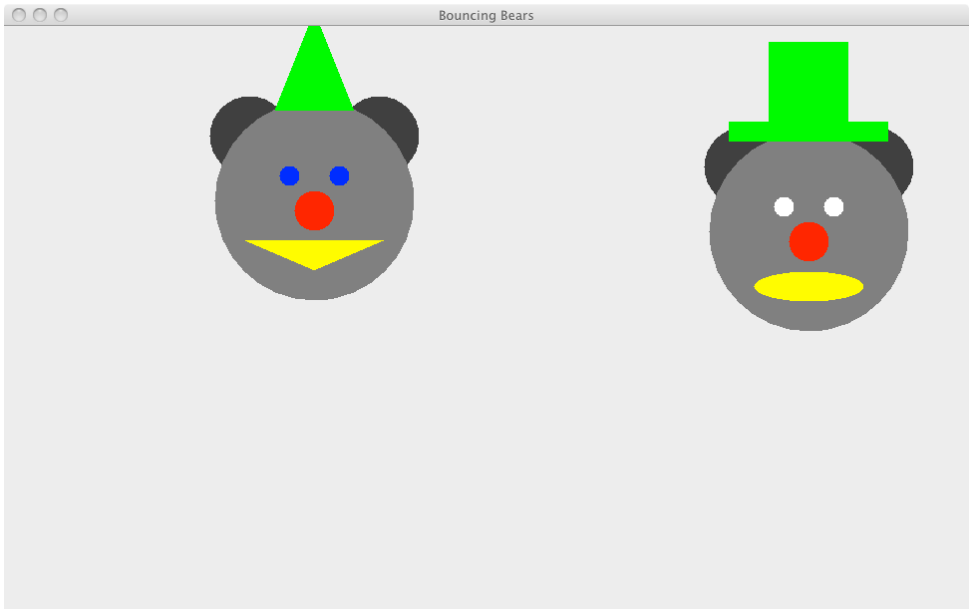
Diese Programmierübung demonstriert Ihnen nebenbei die wesentlichen Prinzipien eines jeden Vektorgrafikprogramms. Gleichzeitig nutzen wir diese Prinzipien, um Ihnen anhand einer einfachen Animation die Prinzipien der objektorientierten Programmierung und etwas Grafikprogrammierung an einem etwas komplexeren Problem zu veranschaulichen. Die Aufgabe ist eine grafische Erweiterung zu der Aufgabe 18.1.

Unser Problem soll "Bouncing Bears" sein – eine Abwandlung einer häufig im Rahmen der Programmierausbildung genutzten Problemstellung namens "Bouncing Balls".

Sie sollen nun die durch Sie programmierte Figurenhierarchie nutzen, um "Bouncing Bears" im Rahmen einer grafischen SWING-Oberfläche zu implementieren. Bei "Bouncing Bears" bewegen sich mehrere Bären[1] wie nachfolgend beschrieben auf einem leeren Canvas (ein Bildschirmbereich, der mittels Grafikbibliotheken "bemalt" werden kann):

- Ein Bär bewegt sich solange in die Richtung seines Bewegungsvektors (DX, DY) bis er an eine Begrenzung seines Canvas stößt.
- Die Bewegungsvektoren bewegen sich zufällig in folgendem Zahlenbereich $DX \in \{-10..-1, 1..10\} \wedge DY \in \{-10..-1, 1..10\}$ sind jedoch niemals null $DX \neq 0 \wedge DY \neq 0$.
- Stößt ein Bär an die obere Begrenzung eines Canvas so wird DY mit einem negativen Y-Bewegungsanteil zwischen -10 und -1 zufällig gesetzt und die Bewegung des Bärs mit dem neuen Bewegungsvektor fortgesetzt.
- Stößt ein Bär an die untere Begrenzung eines Canvas so wird DY mit einem positiven Y-Bewegungsanteil zwischen 1 und 10 zufällig gesetzt und die Bewegung des Bärs mit dem neuen Bewegungsvektor fortgesetzt.
- Stößt ein Bär an die linke Begrenzung eines Canvas so wird DX mit einem positiven X-Bewegungsanteil zwischen 1 und 10 zufällig gesetzt und die Bewegung des Bärs mit dem neuen Bewegungsvektor fortgesetzt.
- Stößt ein Bär an die rechte Begrenzung eines Canvas so wird DX mit einem negativen X-Bewegungsanteil zwischen -1 und -10 zufällig gesetzt und die Bewegung des Bärs mit dem neuen Bewegungsvektor fortgesetzt.

Auf diese Weise sieht es so aus, als ob die Bären zwischen den unteren und oberen sowie linken und rechten Begrenzungen eines Canvas hin- und her abprallen und sich so zufällig auf dem Canvas bewegen.



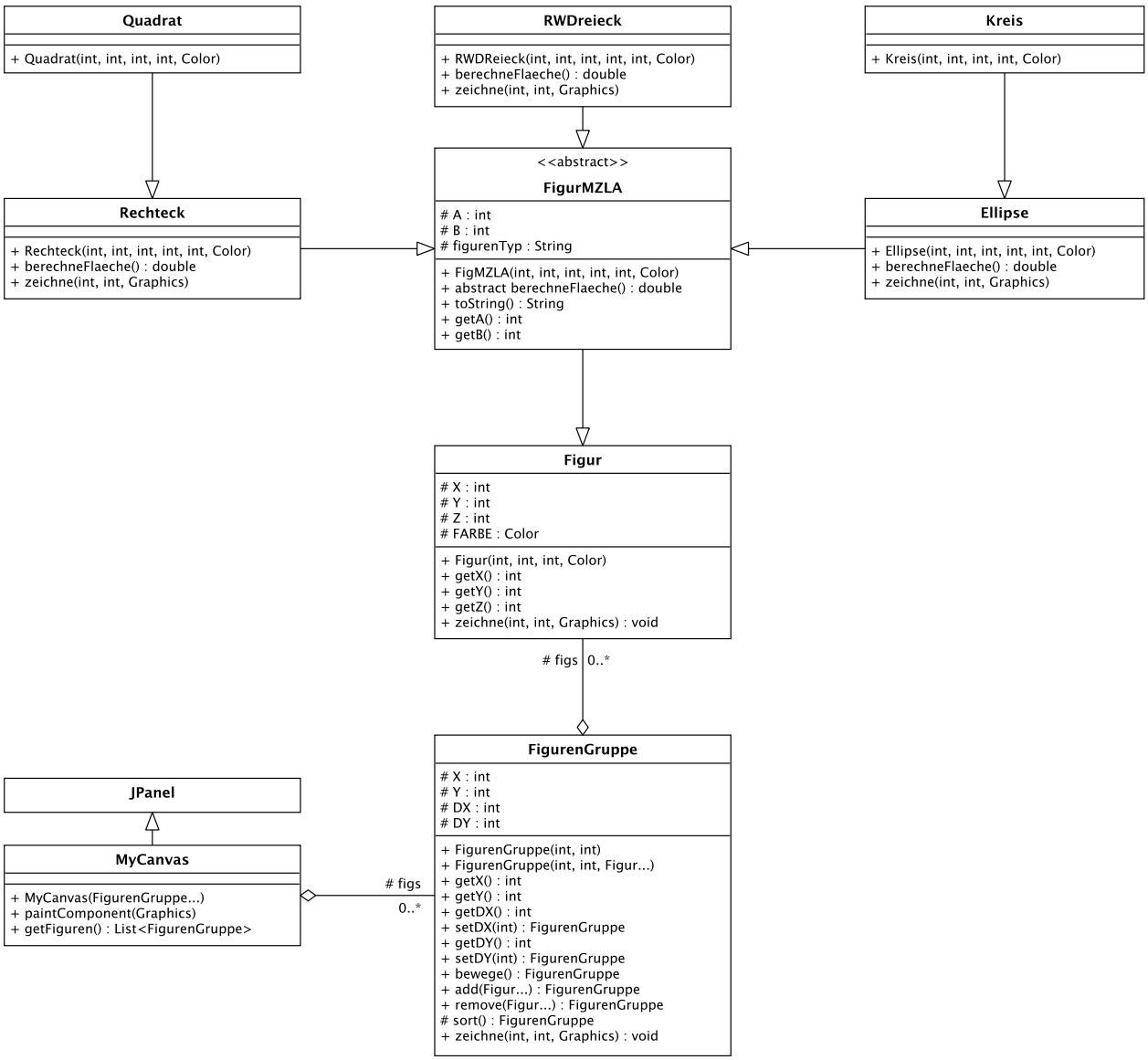
Einen Bären (wie oben dargestellt) können Sie dabei als Gruppe von Figuren auffassen, die aus den Ihnen mittlerweile bekannten Grundfiguren Kreis, Rechteck, Quadrat, Ellipse und rechtwinkliges Dreieck aufgebaut ist.

[1] Bei "Bouncing Balls" bewegen sich einfach nur Kreise bzw. Bälle – aber das ist uns zu langweilig ;-)

Sie müssen hierzu ihre Figurenhierarchie noch etwas erweitern (z.B. um Farben und die Fähigkeit, dass eine Figur sich selber zeichnen kann). Haben Sie dies bewältigt, können Sie z.B. mit unten stehenden Zeilen, eine Gruppe von Figuren erzeugen, die den im oberen Bild rechts zu sehenden "Bären" entspricht.

```
1  FigurenGruppe gesicht1 = new FigurenGruppe(0, 0,
2    new Kreis(0, 0, 0, 100, Color.GRAY),        // Gesicht
3    new Kreis(0, -10, 1, 20, Color.RED),        // Nase
4    new Kreis(65, 65, -1, 40, Color.DARK_GRAY), // Linkes Ohr
5    new Kreis(-65, 65, -1, 40, Color.DARK_GRAY), // Rechtes Ohr
6    new Kreis(-25, 25, 1, 10, Color.WHITE),     // Linkes Auge
7    new Kreis(25, 25, 1, 10, Color.WHITE),      // Rechtes Auge
8    new Ellipse(0, -55, 1, 55, 15, Color.YELLOW), // Mund
9    new Rechteck(-80, 90, 1, 160, 20, Color.GREEN), // Hut (Krempe, unten)
10   new Rechteck(-40, 90, 1, 80, 100, Color.GREEN) // Hut (oben)
11 );
```

Die Erweiterung der Figurenhierarchie sehen Sie in folgendem Klassendiagramm.



Im Wesentlichen sind dabei folgende Erweiterungen zu Ihrer bislang bereits implementierten Figurenhierarchie vorgenommen worden:

1. `Figur`**`en`** wurden um folgende Aspekte erweitert:

   a) Figuren können zusätzlich zur Z-Ebene nun auch noch eine Farbe vom Typ `Color` haben. Die Farbe wird im jeweiligen Konstruktor der Figuren als letzter Parameter übergeben. Schlagen Sie bitte hierzu die Klasse `java.awt.Color` nach[2].

   b) `Figuren` können sich selber relativ zu einer gegebenen X,Y Position mittels eines der Methode `zeichne()` übergebenen `Graphics` Objekts zeichnen. Schlagen Sie hierzu bitte die Klasse `java.awt.Graphics` nach[3]. Die folgenden Codezeilen zeigen beispielhaft eine mögliche Implementierung der `zeichne()` Methode für die Klasse Ellipse. Sie müssen noch eine vergleichbare Funktionalität für die `zeichne()` Methoden der Klassen `Rechteck` und `RWDreieck` implementieren.

```
1  /**
2   * Zeichnet eine Ellipse in einer Component mittels eines Graphics Objekts
3   * @param x X-Position zu der relativ gezeichnet werden soll
4   * @param y Y-Position zu der relativ gezeichnet werden soll
5   * @param g Graphics object das zum Zeichnen genutzt werden soll
6   */
7  public void zeichne(int x, int y, Graphics g) {
8    int x_draw = x + this.getX() - this.getA();
9    int y_draw = y - this.getY() - this.getB();
10   g.setColor(this.FARBE);
11   g.fillOval(x_draw, y_draw, this.getA() * 2, this.getB() * 2);
12 }
```

2. `Figuren` können einer `FigurenGruppe` zugeordnet werden.

   a) Eine `FigurenGruppe` hat eine Position (X, Y) zu der relativ alle zugeordneten Figuren gezeichnet werden. Die X, Y Position der `FigurenGruppe` kann im Konstruktor angegeben werden, ebenso wie die der `FigurenGruppe` zuzuordnenden `Figuren`.

   b) Alle einer `FigurenGruppe` zugeordnete `Figuren` werden anhand ihrer Z-Ebene sortiert. Wird eine weitere Figur der `FigurenGruppe` hinzugefügt, so wird diese Figur entsprechend Ihrer Z-Ebene in die Liste der Figuren der `FigurenGruppe` mittels der `sort()` Methode einsortiert

   c) Eine `FigurenGruppe` hat einen Bewegungsvektor (DX, DY) mit der sich alle der `FigurenGruppe` zugeordneten Figuren auf dem Canvas bewegen. Der Bewegungsvektor kann mittels der `setDX()` und `setDY()` Methoden gesetzt werden. Die Methode `bewege()` verschiebt dann die Figuren-Gruppe um die in DX und DY angegebenen Werte. Die Werte von `DX`, `DY` können auch negativ sein (Bewegung nach links bzw. unten).

   d) `FigurenGruppen` können sich selber relativ zu einer gegebenen X,Y Position mittels eines der Methode `zeichne()` übergebenen `Graphics` Objekts zeichnen. Dies geschieht derart, dass alle der `FigurenGruppe` zugeordneten Figuren in der Reihenfolge Ihrer Z-Ebene gezeichnet werden (mittels der Methode `zeichne()` des `Figuren` Objekts). Die Implementierung der `zeichne()` Methode ist Ihnen für die Klasse `FigurenGruppe` der Einfachheit halber vorgeschlagen.

```
1  /**
2   * Zeichnet eine Gruppe von Figuren mittels eines Graphics Objekts
```

[2] `http://docs.oracle.com/javase/7/docs/api/java/awt/Color.html`
[3] `http://docs.oracle.com/javase/7/docs/api/java/awt/Graphics.html`

```
 * @param x X-Position zu dem die Figuren der Gruppe relativ gezeichnet werden
 * @param y Y-Position zu dem die Figuren der Gruppe relativ gezeichnet werden
 * @param g Graphics object das zum Zeichnen genutzt werden soll
 */
public void zeichne(int x, int y, Graphics g) {
  for (Figur f : this.figs) {
    f.zeichne(x + this.X, y - this.Y, g);
  }
}
```

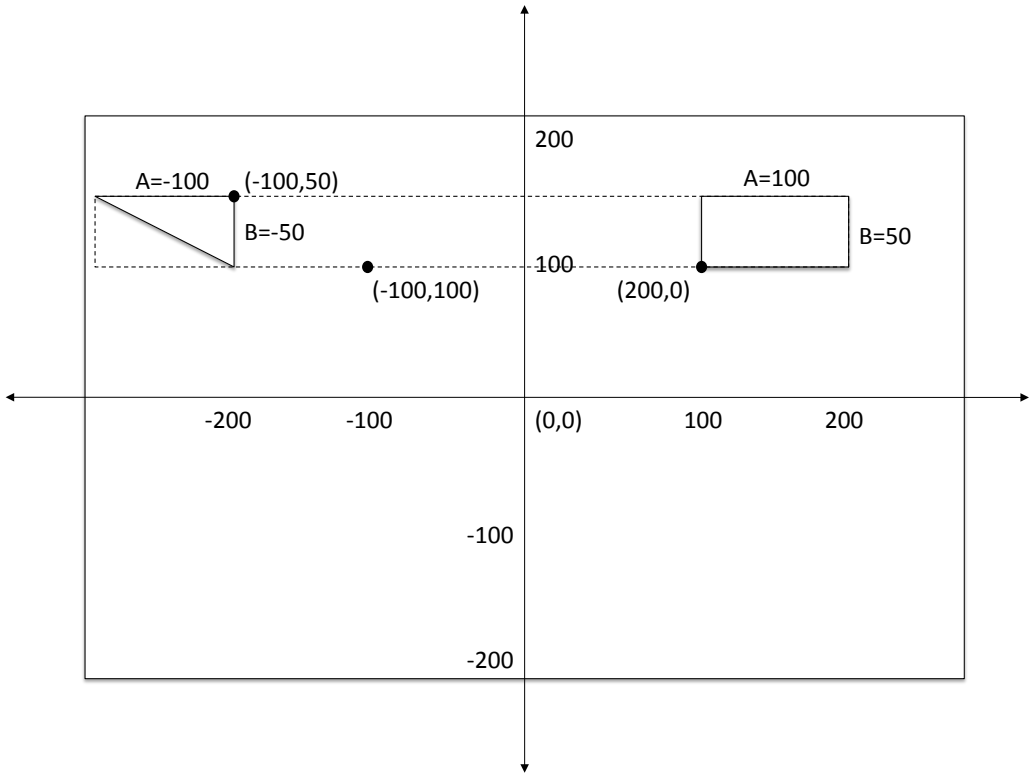
3. Mehrere `FigurenGruppen` können einem Canvas zugeordnet werden. Als Canvas wird eine GUI Komponente bezeichnet, die mittels einfacher Operationen (Kreis, Oval, Rechteck, Linie, Polygonzug, etc.) bemalt und gefüllt werden kann. Aus diesen einfachen Operationen werden komplexere Grafikanweisungen und Animationen zusammengesetzt. Unser Canvas soll `MyCanvas` heißen und ein normales `JPanel` der SWING Bibliothek erweitern.

   a) Alle `FigurenGruppen`, die einem `MyCanvas` zugewiesen werden, können im Konstruktor angegeben werden.

   b) Jede Komponente der SWING Bibliothek veranlasst mittels des Hooks `paintComponent()` ihre Darstellung. So auch `JPanel` und natürlich unser abgeleitetes `MyCanvas`. Die Implementierung der `paintComponent()` Methode für die Klasse `MyCanvas` ist Ihnen nachfolgend angegeben.

```
/**
 * Hook der zur Darstellung unserer MyCanvas Komponente aufgerufen wird
 * @param g Graphics object, dass zur Darstellung der Komponente zu nutzen ist.
 */
public void paintComponent(Graphics g) {
  super.paintComponent(g);
  int x = this.getWidth() / 2;
  int y = this.getHeight() / 2;
  for (FigurenGruppe f : this.getFiguren()) {
    f.zeichne(x, y, g);
  }
}
```

Nachfolgende Abbildung zeigt am Beispiel eines rechtwinkligen Dreiecks und eines Rechtecks wie `Figuren` innerhalb von `FigurenGruppen` auf einem `MyCanvas` positioniert werden.



Die Bezugspunkte der Figuren bzw. der FigurenGruppe sind dabei jeweils durch ein • gekennzeichnet.

- `FigurenGruppen` werden mittels Ihrer X,Y Position relativ zum Mittelpunkt des `MyCanvas` positioniert. Im unten stehenden Beispiel ist dies $X = -100$ und $Y = 100$.
- `Figuren` werden innerhalb einer `FigurenGruppe` relativ zur Position einer `FigurenGruppe` positioniert.
  - Im unten stehenden Beispiel ist das Dreieck an Position $X = -100$ und $Y = 50$ relativ zu seiner `FigurenGruppe` positioniert (beachten Sie bitte das Breite A und Höhe B negative Werte haben können, wie am Beispiel des Dreiecks gezeigt wird).
  - Im unten stehenden Beispiel ist das Rechteck an Position $X = 200$ und $Y = 0$ relativ zu seiner `FigurenGruppe` positioniert.

22.2.1. Teilaufgabe: Paint the Bears

Implementieren Sie nun das gezeigte Klassendiagramm von "Bouncing Bears" und prüfen Sie ihre Implementierung mit den folgenden Code-Zeilen. Es sollten zwei Bären in einem 800x600 großen Fenster dargestellt werden (noch ohne Animation).

```
1 FigurenGruppe baer1 = new FigurenGruppe(-150, 0,
2   new Kreis(0, 0, 0, 100, Color.GRAY), // Gesicht
3   new Kreis(0, -10, 1, 20, Color.RED), // Nase
4   new Kreis(65, 65, -1, 40, Color.DARK_GRAY), // Linkes Ohr
5   new Kreis(-65, 65, -1, 40, Color.DARK_GRAY), // Rechtes Ohr
6   new Kreis(-25, 25, 1, 10, Color.WHITE), // Linkes Auge
7   new Kreis(25, 25, 1, 10, Color.WHITE), // Rechtes Auge
8   new Ellipse(0, -55, 1, 55, 15, Color.YELLOW), // Mund
```

```
9    new Rechteck(-80, 90, 1, 160, 20, Color.GREEN), // Hut (Krempe, unten)
10   new Rechteck(-40, 90, 1, 80, 100, Color.GREEN) // Hut (oben)
11 );
12
13 FigurenGruppe baer2 = new FigurenGruppe(150, 0,
14   new Kreis(0, 0, 0, 100, Color.GRAY), // Gesicht
15   new Kreis(-25, 25, 1, 10, Color.BLUE), // Linkes Auge
16   new Kreis(25, 25, 1, 10, Color.BLUE), // Rechtes Auge
17   new Kreis(0, -10, 1, 20, Color.RED), // Nase
18   new Kreis(65, 65, -1, 40, Color.DARK_GRAY), // Linkes Ohr
19   new Kreis(-65, 65, -1, 40, Color.DARK_GRAY), // Rechtes Ohr
20   new RWDreieck(0, -40, 1, -70, -30, Color.YELLOW), // Mund (links)
21   new RWDreieck(0, -40, 1, 70, -30, Color.YELLOW), // Mund (rechts)
22   new RWDreieck(0, 90, 1, -40, 100, Color.GREEN), // Partyhut (links)
23   new RWDreieck(0, 90, 1, 40, 100, Color.GREEN) // Partyhut (rechts)
24 );
25
26 JFrame mainFrame = new JFrame("Bouncing Bears");
27 mainFrame.setSize(800, 600);
28 mainFrame.add(new MyCanvas(baer1, baer2));
29 mainFrame.setVisible(true);
```

Hinweis: Im UML Klassendiagramm finden Sie an mehreren Stellen in den Methoden/Konstruktorsignaturen Angaben wie

`Figur...`

Dies ist kein Tippfehler. Schlagen Sie im Skript des ersten Semesters noch einmal das Thema "variable Parameterlisten" in Unit 2 (insbesondere Methoden) nach. Wahlweise hilft ggf. auch dieser Link weiter:

`https://today.java.net/pub/a/today/2004/04/19/varargs.html`

22.2.2. Teilaufgabe: Animate the Bears

Sie sollen nun mindestens drei Bären in der angegebenen Art animieren. Hierzu gehen Sie am besten wie folgt vor. Programmieren Sie eine Endlosschleife, innerhalb der die Bärenpositionen aufdatiert werden und geprüft wird, ob die Position der Bären an eine der Begrenzungen des Canvas stoßen. In diesem Fall drehen Sie das Vorzeichen des Bewegungsanteils (DX oder DY) um, und weisen einen neuen Bewegungswert (zwischen 1 und 10) zu (um zufällige Bewegungen zu erzeugen). Nach Aufdatieren der Bärenpositionen aktualisieren Sie die Canvas Darstellung mittels der `repaint()` Methode.

Anschließend pausieren Sie z.B. für 10 Millisekunden[4] und setzen die Verarbeitung der Schleife wie oben angeben fort.

Der Rumpf einer solchen Endlosschleifen könnte in etwa wie folgt aussehen:

```
1 [...]
2
3 MyCanvas canvas = new MyCanvas(baer1, baer2, baer3);
4
5 [...]
6
```

[4] Schlagen Sie hierzu bitte die Klasse `Thread` inklusive der Methoden `currentThread()` und `sleep()` nach: `http://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html`

```
7  while (true) {
8    try {
9      Thread.sleep(10); // 10 ms schlafen
10     int h = canvas.getHeight() / 2; // Hoehe des Canvas bestimmen
11     int w = canvas.getWidth() / 2; // Breite des Canvas bestimmen
12
13     // Implementieren Sie etwas Intelligentes, um die Baeren zu animieren
14
15    }
16    canvas.repaint();
17   } catch (Exception ex) { System.out.println(ex); }
18 }
```

22.3. Aufgabe: Conways Game of Life

Das Spiel des Lebens (engl. Conway's Game of Life) ist ein vom Mathematiker John Horton Conway 1970 entworfenes System, basierend auf einem zweidimensionalen zellulären Automaten.

Details finden Sie unter diesem Wikipedia Link:

http://de.wikipedia.org/wiki/Conways_Spiel_des_Lebens

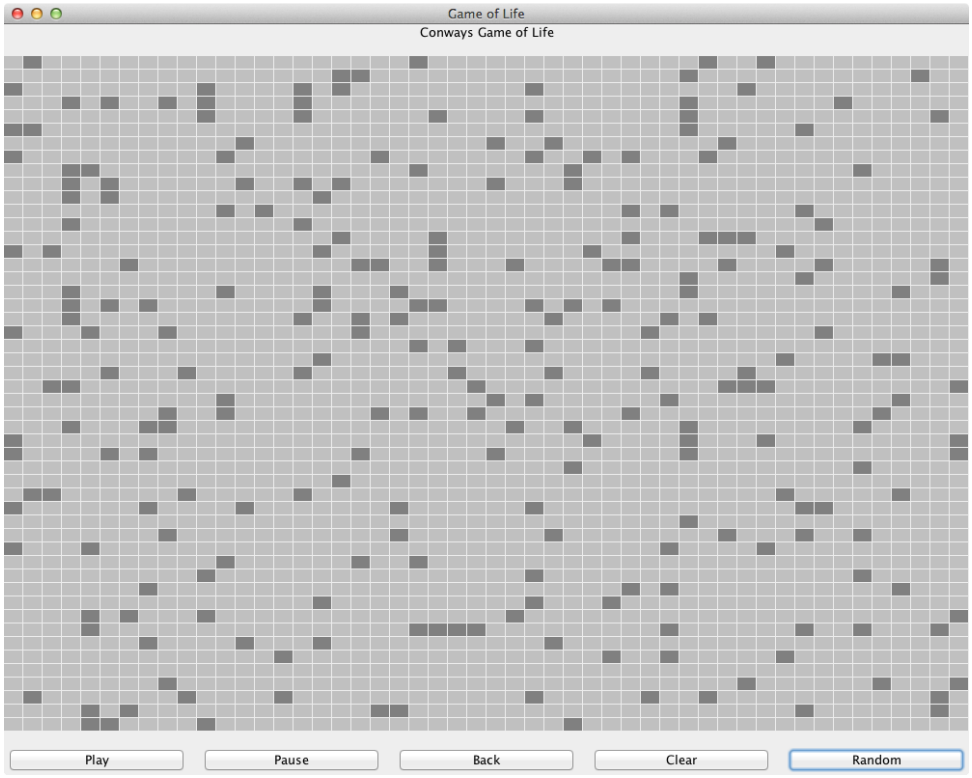
Das Spielfeld ist in Zeilen und Spalten unterteilt. Jedes Gitterquadrat ist ein Zellulärer Automat (Zelle), der einen von zwei Zuständen einnehmen kann, welche oft als lebendig und tot bezeichnet werden. Zunächst wird eine Anfangsgeneration von lebenden Zellen auf dem Spielfeld platziert. Jede lebende oder tote Zelle hat auf diesem Spielfeld genau acht Nachbarzellen, die berücksichtigt werden (Moore-Nachbarschaft). Die nächste Generation ergibt sich durch die Befolgung einfacher Regeln.

Die Folgegeneration wird für alle Zellen gleichzeitig berechnet und ersetzt die aktuelle Generation. Der Zustand einer Zelle, lebendig oder tot, in der Folgegeneration hängt nur vom Zustand der acht Nachbarzellen dieser Zelle in der aktuellen Generation ab und wird durch folgenden Regeln bestimmt (Conway Regeln).

- Regel Nr. 1: Eine tote Zelle mit genau drei lebenden Nachbarn wird in der Folgegeneration neu geboren.
- Regel Nr. 2: Lebende Zellen mit weniger als zwei lebenden Nachbarn sterben in der Folgegeneration an Einsamkeit.
- Regel Nr. 3: Eine lebende Zelle mit zwei oder drei lebenden Nachbarn bleibt in der Folgegeneration lebend.
- Regel Nr. 4: Lebende Zellen mit mehr als drei lebenden Nachbarn sterben in der Folgegeneration an Überbevölkerung.

Mit diesen vier einfachen Regeln entsteht aus bestimmten Anfangsmustern im Laufe des Spiels eine Vielfalt komplexer Strukturen. Einige bleiben unverändert, andere oszillieren und wieder andere wachsen oder vergehen. Manche Strukturen, sogenannte Gleiter, bewegen sich auf dem Spielfeld fort.

Sie sollen nun eine Applikation mit grafischer Oberfläche entwickeln, die dieses "Spiel des Lebens" implementiert und visualisiert.



Wie sie dem Screenshot entnehmen können, soll dieses Spiel:

1. Ein 50 x 50 Feld großes Spielfeld darstellen und den Lebensstatus (hellgrau == tot, dunkelgrau == lebendig) jedes einzelnen Feldes kenntlich machen. Das Spielfeld ist nicht begrenzt, d.h. an der linken Seiten setzt sich das Spielfeld an der rechten Seite fort. Genauso gilt dies für die oberen und unteren Seiten.
2. Eine initiale Belegung soll mittels eines Random Buttons veranlasst werden können (Füllungsgrad lebender Felder des Spielfeldes 10%).
3. Das Spielfeld soll gelöscht werden können (Clear). D.h. alle Feldstatus werden auf tot gesetzt.
4. Ein Spiel soll begonnen werden können mit Play. D.h. alle 250 ms soll die Folgegeneration des Spielfeldes berechnet und dargestellt werden.
5. Ein laufendes Spiel soll mittels Pause pausiert werden können. Die erneute Betätigung von Play soll das Spiel dann fortsetzen.
6. Es soll mittels Back Buttons der Spielstatus der vorhergehenden Generation wieder dargestellt werden können. Insgesamt ist die Historie von 10 Generationen im Spiel zu halten und durch (mehrfache) Back Button Betätigung wiederherzustellen.
7. Zusätzlich sollen über einen Mausklick der Status (tot oder lebend) individueller Felder auf dem Spielfeld geändert werden können, z.B. um individuelle Spielfeldbelegungen definieren zu können (z.B. einen Gleiter).

Sie können Zusatzpunkte erhalten, wenn Sie sich weitere Features für dieses Spiel ausdenken.

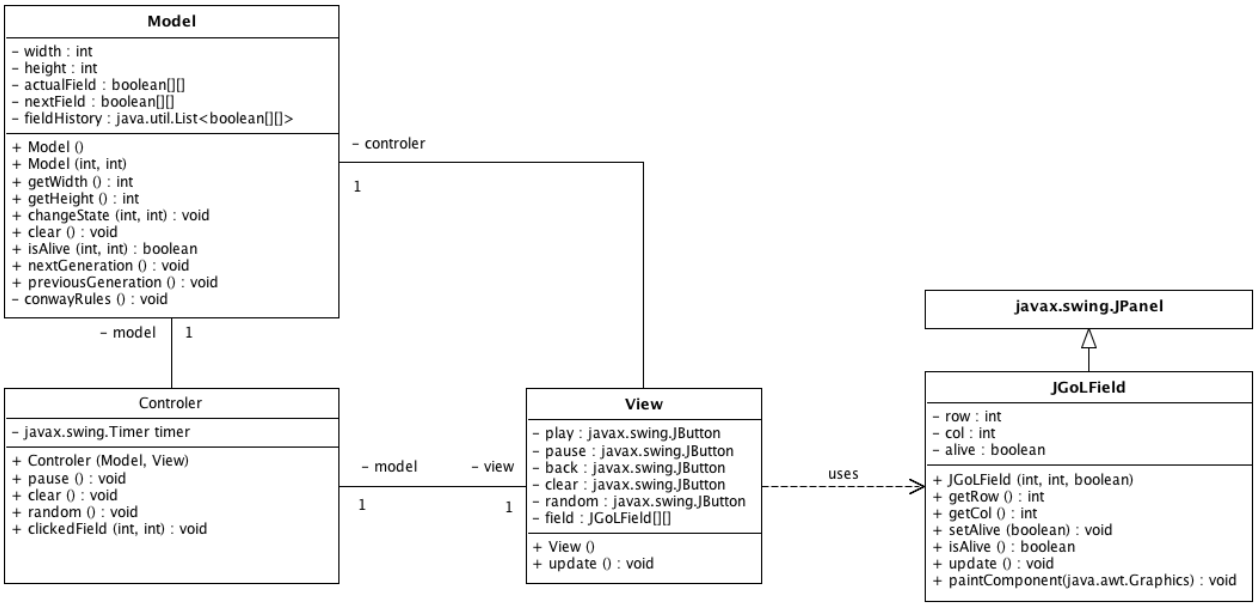
- Z.B. eine Navigation durch die eine wesentlich längere Historie (z.B. 1000 Generationen) eines Spiels mittels eines Sliders.
- Platzierung vorgefertigter Strukturen (z.B. Gleiter, etc.) auf dem Spielfeld.

Hinweise für die Implementierung:

Beachten Sie bitte das Java Swing Applikationen fehleranfällig sind, wenn Sie zusätzliche Threads starten (Swing ist nicht Thread-Safe, vgl. Unit 12). Es bietet sich daher bei wiederkehrenden und periodischen Aufgaben in Java Swing Applikationen an, diese mittels eines Swing Timer Objekts periodisch zu starten. Dies kann bspw. so erfolgen:

```
1  import java.awt.event.ActionEvent;
2  import java.awt.event.ActionListener;
3  import javax.swing.Timer;
4
5  ...
6
7  Timer timer = new Timer(250, new ActionListener() {
8    public void actionPerformed(ActionEvent e) {
9      System.out.println("250 Milisekunden sind um.");
10     // Hier koennen Sie natuerlich beliebigen Code
11     // einbetten, der periodisch aufgerufen werden
12     // soll. Z.B. Die Anwendung des naechsten Durchgangs
13     // der Conway Regeln auf die aktuelle Spielfeldbelegung.
14   }
15 });
16
17 ...
18
19 // Startet Timer und veranlasst alle 250 ms den ActionListener aufzurufen.
20 timer.start();
21
22
23 ...
24
25 // Stoppt Timer und die damit verknuepfte Periodizitaet
26 timer.stop();
27
28 ...
```

Es wird Ihnen ferner folgendes UML Klassendiagramm an die Hand gegeben, um Sie bei der Implementierung eines MVC Patterns zu unterstützen.



22.4. Aufgabe: Tic Tac Toe mit grafischer Oberfläche

Sie haben in der Vorlesung das Model-View-Controller Konzept zur Entwicklung von Applikationen mit grafischen Oberflächen gehört. Ihnen wurde das MVC Konzept am Beispiel eines Taschenrechners vermittelt:

- Model = Taschenrechnerlogik
- View = Bedienoberfläche
- Controller = Reagieren auf Bedienereingabe und Anstoßen der Logik

Sie haben ferner in der Vorlesung die TIC TAC TOE Engine kennen gelernt und in Aufgabe 21.2 haben Sie ihre eigene TIC TAC TOE Spielstrategie entwickelt. Sie sollen nun beide Konzepte derart verbinden, dass Sie eine TIC TAC TOE App (`T3App`) mit grafischer Bedienoberfläche entwickeln. Sie sollen dabei folgende Requirements umsetzen:

1. Der Bediener Ihrer App spielt immer die X Rolle in TIC TAC TOE.
2. Der Bediener Ihrer App spielt gegen Ihre eigene Strategie.
3. Die App soll in etwa wie in gezeigter Abbildung aussehen.
4. Die App soll Felder auf die ein Spieler nicht mehr setzen kann, für eine Interaktion deaktivieren.
5. Die App soll den Spielstatus ausgeben, wenn sich dieser ändert (X gewonnen, Y gewonnen, unentschieden, fehlerhafter Zug von X, fehlerhafter Zug von Y).
6. Nutzen Sie zur Implementierung Ihrer App, das MVC Pattern und die JAVA Swing Bibliothek.

    a) Ihr Model sollte die T3Engine und deren Spielzustand geeignet kapseln.

    b) Ihr View sollte das TIC TAC TOE Spielfeld darstellen

    c) Ihr Controller sollte den Spielablauf und die Bedienerinteraktionen koordinieren.

Sie müssen zur Implementierung folgendes beachten:

Die T3 Engine geht davon aus, dass zwei Automatikstrategien gegeneinander spielen. Ihre App hat jedoch einen Bediener und muss daher mit diesem interagieren. Sie müssen also eine eigene interaktive Ablauflogik vorsehen, deren einer Spieler der Bediener und deren anderer Spieler ihre Spielstrategie ist. Sie können sich dabei an der T3 Ablauflogik der T3 Engine orientieren. Die Automatiklogik finden Sie in der Klasse `T3Spiel` und dort in der Methode `leite_spiel`. Ausgehend von dieser Logik sollten sie in ihrem Controller ihre eigene Ablauflogik implementieren, die das Zugrecht wechselseitig zwischen dem Bediener und der Software zuweist.

Sie können Zusatzpunkte erhalten, wenn Sie folgende Requirements umsetzen:

1. Der Bediener der App kann wählen, ob er in der X oder der O Rolle spielen möchte.
2. Der Bediener der App kann in einem Dialog aus mehreren Strategien wählen, gegen die er spielen möchte.
3. Zur besseren Übersichtlichkeit sollen Felder die vom Spieler X belegt worden sind in grün und Felder die vom Spieler O belegt worden sind in rot markiert werden.
4. Der Bediener kann nach Ende eines Spiels ein neues Spiel starten und dabei wieder Rolle und Gegnerstrategie wählen.

22.5. !!! NEU !!! Aufgabe: Sortieren

Sie sollen nun ein kleines Spiel schreiben. Ziel ist es eine zufällig verteilte Anzahl an Zahlen auf einem n x m Raster aufsteigend zu sortieren. Das Programm soll dabei die Zeit stoppen, die dafür benötigt wird. Nachvollgehend sehen sie an einem 3 x 3 Beispiel, wie die Zahlen zu sortieren sind und welche Sortiervollständigkeit sich daraus ergibt.

2	1	9
6	7	8
3	4	5

0 von 9 Zahlen richtig sortiert
(0 % Sortiervollständigkeit)

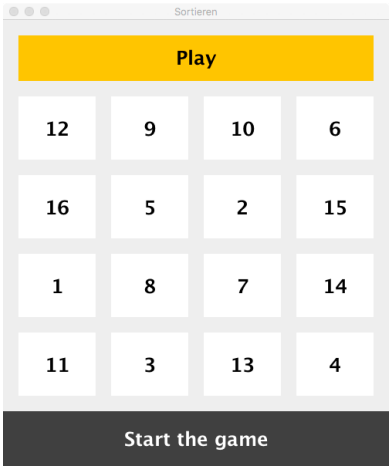
1	**2**	**3**
8	7	**6**
5	4	**9**

4 von 9 Zahlen richtig sortiert
(44,4% Sortiervollständigkeit)

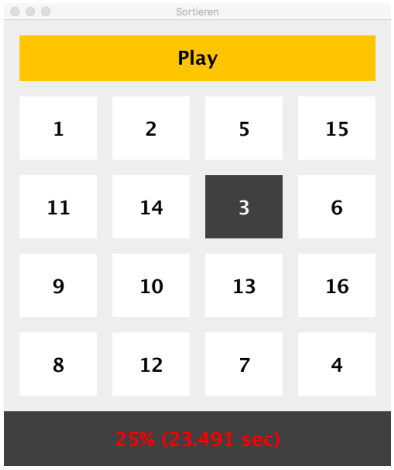
1	**2**	**3**
4	**5**	**6**
7	**8**	**9**

9 von 9 Zahlen richtig sortiert
(100% Sortiervollständigkeit)

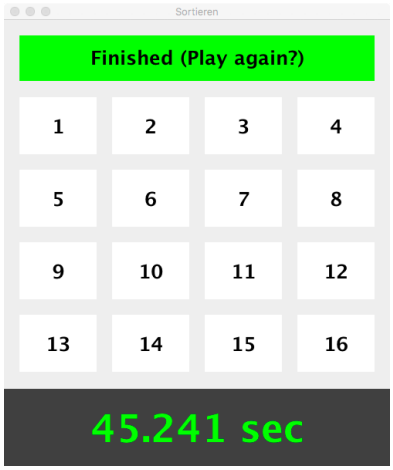
Auf solchen Spielfeldern, können zwei Felder getauscht werden, indem erst ein Feld angeklickt (und markiert) wird. Wird dann ein zweites Feld angeklickt, so werden diese beiden Felder getauscht. Durch eine Sequenz solcher Tauschoperationen kann das Spielfeld vollständig sortiert werden. Screenshots der Musterlösung sind nachfolgend zur besseren Orientierung angegeben.



Spielbeginn:
Das Spiel kann mit **Play** gestartet werden. Ab dann läuft die Zeit.



Spiel während Laufzeit
Es wird die Sortiervollständigkeit angegeben und die bislang abgelaufene Zeit. Felder die zum Tauschen selektiert wurden, werden markiert.



Spielende:
Es wird das Spielende erkannt und die benötigte Zeit angezeigt. Ein neues Spiel kann mit **Play again** begonnen werden.

Entwickeln bitte eine MVC-basierte Lösung für dieses Spiel. D.h.

- ein Model, welches das Problem konzeptionell löst
- einen View, der die oben gezeigte Darstellung realisiert (sie müssen nicht exakt die visuelle Gestaltung treffen)
- einen Controller, der Nutzerinteraktionen erkennen kann.

Dokumentieren Sie bitte ihre Klassen ergänzend mit einem UML Klassendiagramms.

Tipp: Fangen Sie mit dem Model an, nicht mit dem View!

Hinweis: Controller sind nicht darauf beschränkt nur auf Nutzerinteraktionen zu reagieren. Sie können auch zeitgesteuert getriggert werden. Folgender Code gibt alle fünf Sekunden "Hello World" auf der Konsole aus.

```
1  import javax.swing.Timer;
2
3  [...]
4
5  Timer timer = new Timer(5000, trigger -> {
6    System.out.println("Hello World");
7  });
8
9  // Nachdem der Timer gestartet wurde,
10 // wird alle 5000ms Hello World auf der Konsole ausgegeben.
11 timer.start();
12
13 Thread.sleep(20000);
14
15 // Ein Timer kann mittels timer.stop() wieder gestoppt werden.
16 timer.stop();
```

Teil XII.

Aufgaben zu Unit 12
Parallele Programmierung

23. Multithread Programmierung

23.1. Aufgabe: Paralleles Brute Force Passwort Raten mittels Threads

Gegeben ist ihnen folgende CSV Datei mit nicht im Klartext gespeicherten PINs.

`http://www.nkode.io/assets/programming/pins.csv`

Ihre Aufgabe ist es nun, für jeden in dieser Datei eingetragenen Nutzer seine Klartext-PINs mittels eines Brute Force Rateansatzes zu ermitteln (alle möglichen Kombinationen zulässiger Zeichen durchprobieren). Da Brute Force Raten sehr rechenintensiv ist, sollten sie ihr Raten **mittels Threads** parallelisieren, um den Prozessor besser auszulasten und schneller zu Ergebnissen zu gelangen[1].

Sie wissen, dass die PINs genau 4 Zeichen lang sind und aus folgenden alphanummerischen Zeichen beliebiger Kombination bestehen.

`123456789abcdefghijklmnopqrstuvwxyz`

Die Passwörter sind nicht im Klartext sondern als Base64 kodierte MD5 Hashsumme gespeichert. Das MD5-Verfahren dient bspw. dazu, Passwörter nicht im Klartext speichern zu müssen.

`http://de.wikipedia.org/wiki/MD5`

> Hat ein Nutzer einen Namen $name$ (z.B. *"Max Mustermann"*) und ein Password pwd (z.B. *"ax5f"*) so wird aus beiden Bestandteilen eine Zeichenkette konkateniert (z.B. *"Max Mustermannax5f"*) und darauf die MD5 Hashsumme gebildet ($\oplus$ bezeichne dabei den Operator zur Zeichenkettenkonkatenation):
>
> $$hash_{BYTE} = md5(name \oplus pwd)$$

In unserem Fall wird die als Byte Array vorliegende Hashsumme noch einmal Base64 kodiert (damit ist sie bspw. auch in URLs einsetzbar oder in Textdateien speicherbar, da nur ASCII Zeichen genutzt werden).

$$hash_{ASCII} = base64 \circ md5(name \oplus pwd) = base64(md5(name \oplus pwd))$$

`http://de.wikipedia.org/wiki/Base64`

Eine (kryptografische) Hashfunktion hat die Eigenschaft für eine beliebige Zeichenkette eine Art Fingerabdruck zu generieren. Aus dem Fingerabdruck ist es aber nur sehr aufwändig möglich, wieder auf die ursprüngliche Zeichenkette zu schließen. Dies macht Hashfunktionen für kryptografische Anwendungen so interresant, da sie es ermöglichen Passwörter zu speichern, aber aus den gehashten Passwörtern nicht mehr das ursprüngliche Passwort rückrechnen zu können.

Die Funktion $base64 \circ md5 : String \rightarrow String$ kann in Java wie folgt implementiert werden:

[1] Jetzt wissen sie auch, was die NSA den ganzen Tag so rechnen lässt :-)

```java
public static String md5(String s) throws NoSuchAlgorithmException {
  MessageDigest md = MessageDigest.getInstance("MD5");
  return DatatypeConverter.printBase64Binary(md.digest(s.getBytes()));
}
```

Informationen zu den genutzten Klassen `MessageDigest` und `DatatypeConverter`, finden Sie bspw. hier:

- `http://docs.oracle.com/javase/7/docs/api/java/security/MessageDigest.html`
- `http://docs.oracle.com/javase/7/docs/api/javax/xml/bind/DatatypeConverter.html`

Die in der Datei gebildeten Hashsummen, wurden also mit folgendem Verfahren in Java generiert:

```java
String name = "Max Mustermann";
String pwd = "4321";
String hash = md5(name + pwd);
```

Ähnlich wie in der Vorlesung sollen Sie nun folgende Zeiten bestimmen:

1. Laufzeit in ms des parallelisierten Brute Force Ratens
2. Laufzeit in ms des längsten Rateversuchs
3. Laufzeit des sequentiellen Brute Force Ratens
4. Speedup (d.h. wieviel schneller wäre das parallele Raten gegenüber dem sequentiellen Raten gewesen)

und in etwa folgende Ausgabe auf der Konsole generieren:

```
Michaela Bauer hat Passwort hryc Ratedauer: 12113 ms
Maja Gebauer hat Passwort hfea Ratedauer: 12604 ms
Maja Jackson hat Passwort zfdw Ratedauer: 18660 ms
Magda Abel hat Passwort 2cns Ratedauer: 2567 ms
[...]
Michaela Dauermann hat Passwort b6q3 Ratedauer: 9184 ms
Moritz Abel hat Passwort gh6y Ratedauer: 11956 ms
Laufzeit aller parallel laufenden Rate Threads: 18662 ms
Laufzeit des längsten Rate Threads: 18660 ms
Laufzeit sequentielles Raten: 59718 ms
Speedup: 3.1999
```

23.2. Aufgabe: Simulation von Warteschlangen an Kassen

Sie sollen nun ein Kassensystem in einem Supermarkt mittels parallel laufender Threads simulieren und die Leistung des Kassensystems analysieren. Das Kassensystem besteht aus

- n Kassen (`SupermarketCheckout`), die jeweils eine Warteschlange mit endlicher Länge (`maxQueueLength`) haben (z.B. 10 Kunden). Eine Kasse hat eine Leistung (`performance`), die in Sekunden pro abzurechnendem Produkt gemessen wird (bspw. 10 Sekunden pro Produkt).
- Kunden (`Customer`) gehen in den Supermarkt und kaufen eine zufällige Anzahl an Produkten (`products`) ein.

  - Die Verweildauer eines Kunden im Supermarkt (ohne Kassenaufenthalt) berechnet sich aus der Anzahl eingekaufter Produkte multipliziert mit einer Konstante (`TIME_PER_PRODUCT`), die angibt, wie lang der Einkauf eines Produkts in Anspruch nimmt.
  - Kunden stellen sich zufaellig an einer der ihnen bekannten Kassen (`availableQueues`) an (`enterQueue()`).

- Die n Kunden gehen einzeln und mit zufälligem zeitlichen Abstand in den Supermarkt. Der zeitliche Abstand zweier Kunden, die den Supermarkt betreten liegt zwischen 0 und einer maximalen Obergrenze (`CUSTOMER_ARRIVAL_RATE`) gemessen in Sekunden verteilt (bspw. maximaler Abstand zwischen zwei Kunden 120 Sekunden).

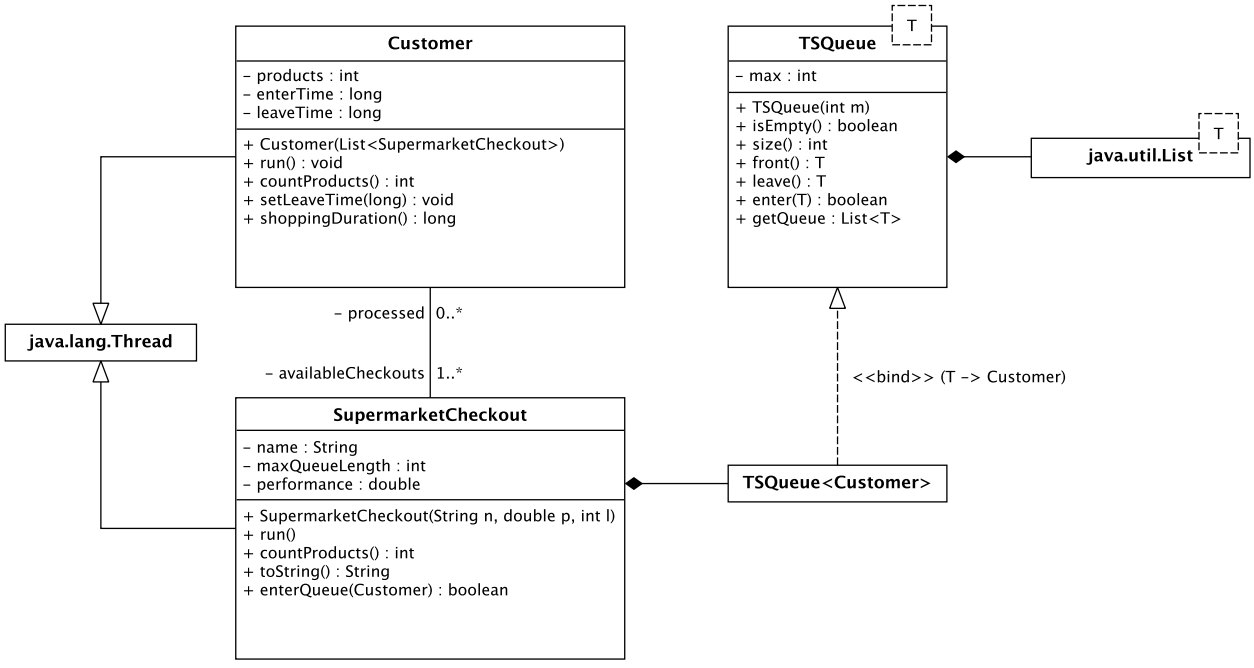
Folgende Konstanten sind definiert:

```
1  /**
2   * Klasse zur Definition aller Parameter fuer Experimente
3   * mit dem Kassensimulationsmodell.
4   * @author Nane Kratzke
5   *
6   */
7  public class Parameters {
8
9    /**
10    * Dauer, die ein Kunde benoetigt, um ein Produkt einzukaufen.
11    * Einheit: Sekunden pro Produkt
12    */
13   static final int TIME_PER_PRODUCT = 60;
14
15   /**
16    * Minimale Anzahl an Produkten, die ein Kunde kaufen kann.
17    * Einheit: Produkte
18    */
19   static final int MIN_PRODUCTS = 10;
20
21   /**
22    * Maximale Anzahl an Produkten, die ein Kunde kaufen kann.
23    * Einheit: Produkte
24    */
25   static final int MAX_PRODUCTS = 50;
26
27   /**
28    * Maximale Laenge einer Warteschlange vor einer Kasse.
29    * Einheit: Kunden
30    */
31   static final int MAX_QUEUE_LENGTH = 6;
32
33   /**
34    * Leistung einer normalen Kasse.
35    * Einheit: Sekunden pro Produkt
36    */
37   static final int NORM_QUEUE = 20;
38
39   /**
40    * Leistung einer schnellen Kasse.
41    * Einheit: Sekunden pro Produkt
42    */
43   static final int FAST_QUEUE = 10;
```

```
44
45    /**
46     * Leistung einer langsamen Kasse.
47     * Einheit: Sekunden pro Produkt
48     */
49    static final int SLOW_QUEUE = 40;
50
51    /**
52     * Anzahl an Kunden, die simuliert werden sollen.
53     * Einheit: Kunden
54     */
55    static final int CUSTOMER_AMOUNT = 100;
56
57    /**
58     * Maximale Sekundenanzahl zwischen zwei Kunden.
59     * Einheit: Ein Kunde alle n Sekunden
60     */
61    static final int CUSTOMER_ARRIVAL_RATE = 30;
62
63  }
```

Sie sollen das Kassensystem durch die beiden folgenden Kennzahlen beurteilen und für Entscheidungsfindungen heranziehen:

1. Mittlere Anzahl an Produkten pro Minute (die durch das Kassensystem gehen)
2. Minimale, Maximale und mittlere Aufenthaltszeit von Kunden im Supermarkt pro Produkt



Implementierungshinweise:

1. Für alle Zufallszahlen wird die Gleichverteilung angenommen, d.h. nutzen Sie `Math.random()`.
2. Sie sollen in 1000 facher Geschwindigkeit simulieren. D.h. Zeitangaben der Aufgabenstellung in Sekunden können mit Milisekunden im Code gleichgesetzt werden.

3. Wollen Sie Wartezeiten vor Kassen oder Aufenthaltszeiten im Supermarkt in einem Thread simulieren, nutzen sie bitte `Thread.sleep()`.
4. Die Warteschlange vor einer Kasse können Sie auf Basis der bereits von Ihnen implementiertern generischen Datenstruktur `MyQueue` realisieren (vgl. Aufgabe 20.1). Beachten Sie aber bitte, dass sie diese Implementierung noch Threadsafe machen müssen und das die Warteschlange eine begrenzte Anzahl an Einträgen hat (Stichwort Erzeuger/Verbraucher Problem).
5. Ihre Auswertung sollte auf der Konsole ausgegeben, etwa folgende Form haben:

```
Kasse 1 (32 Kunden; 841 Produkte)
Kasse 2 (37 Kunden; 1218 Produkte)
Kasse 3 (31 Kunden; 869 Produkte)
100 Kunden insgesamt
Leistung des Kassensystems: 7.054005 Produkte/Minute

Auswertung der Aufenthaltsdauer eines Kunden im Supermarkt:
Minimum: 70.0 Sekunden/Produkt
Maximum: 1297.4 Sekunden/Produkt
Durchschnitt: 288.8671 Sekunden/Produkt
```

Sie sollen nun das folgende Kassensystem

```
1  // Anlegen der Supermarktkassen
2  List<SupermarketCheckout> qs = new LinkedList<>();
3  qs.add(new SupermarketCheckout(
4    "Kasse 1",
5    Parameters.NORMAL_QUEUE,
6    Parameters.MAX_QUEUE_LENGTH
7  ));
8  qs.add(new SupermarketCheckout(
9    "Kasse 2",
10   Parameters.NORMAL_QUEUE,
11   Parameters.MAX_QUEUE_LENGT
12 )));
```

durch hinzufügen zusätzlicher Kassen im Werte zweier Normkassen (`Parameters.NORM_QUEUE`) optimieren. Ihnen stehen hierzu

- Normkassen (`Parameters.NORM_QUEUE`)
- Schnellkassen (`Parameters.FAST_QUEUE`, doppelt so schnell wie Normkassen) oder
- Bummelkassen (`Parameters.SLOW_QUEUE`, halb so schnell wie Normkassen)

zur Verfügung. Sie könnten also

- **eine** zusätzliche **Schnellkasse** oder
- **zwei** weitere **Normkassen** oder
- **vier** weitere **Bummelkassen** oder
- **eine Normkasse und zwei Bummelkassen**

den Supermarktkassen (alle vier Varianten verdoppeln rein rechnerisch die Kassenleistung) hinzufügen. Simulieren sie bitte für jede der vier Varianten 100 Kundenankünfte (`CUSTOMER_AMOUNT`).

- **Für welche Variante entscheiden Sie sich, wenn Sie die mittlere Kundenverweildauer (gemessen in Sekunden/Produkt) im Supermarkt reduzieren sollen?**

Angenommen Sie ändern die Anstellstrategie der Kunden dahingehend, dass sich Kunden immer an die Kasse mit der kürzesten Schlange anstellen.

1. **Wie müssten Sie diese geänderte Anstellstrategie im Code implementieren?**
2. **Für welche der oben genannten vier Varianten entscheiden sie sich nun?**

23.3. Aufgabe: Paralleles Brute Force Passwort Raten mittels Streams

Wie in Aufgabe 23.1 sollen Sie nun wieder Passwörter raten. Diesmal sollen sie die Parallelisierung aber **mittels Streams** vornehmen und den Speedup von parallelen zu sequentiellen Streams für die gegebene Aufgabenstellung bestimmen.

Tipp:

Eine Textdatei können Sie mittels Streams recht einfach zeilenweise verarbeiten:

```
1 URL text = new URL("http://www.nkode.io/assets/programming/pins.csv");
2 BufferedReader reader = new BufferedReader(
3   new InputStreamReader(text.openStream())
4 );
5 Stream<String> lines = reader.lines();
```

Vergleichen Sie bitte die Laufzeiten der Aufgabe 23.1 mit ihrer parallelisierten Stream-Lösung. Welche Art der Parallelisierung bevorzugen sie vor dem Hintergrund von Pragmatik, Möglichkeit Fehler zu programmieren, Anzahl an Codezeilen?

Teil XIII.

Aufgaben zu Unit 13
Funktionale Programmierung nur mit OO-Mitteln

24. Exkurs: Funktionale Programmierung auf Umwegen

Java ist eine Programmiersprache, die bis zur Version 7 per se keine funktionalen Programmierfeatures anbot. Die Demonstration funktionaler Programmierung war daher notgedrungen etwas schwierig. Dennoch lassen sich einige funktionale Programmierfeatures in rein objektorientierten Programmiersprachen nachbauen (wenn auch recht mühsam, mehr dazu in Aufgabe 24.2). Sie können auch Bibliotheken wie bspw. Functional Java (`http://functionaljava.org/`) hierfür nutzen. Letztlich lassen diese Ansätze aber alle die Eleganz funktionaler Sprachfeatures missen. Lichtblick ist hier allein der Java 7 BGGA proposal (`http://javac.info/`) der Closures als Sprachbestandteil in Java aufnehmen soll. Dieser Proposal hat dankenswerterweise in Java 8 Einzug gehalten. Insofern ist dieses Kapitel eher von "historischem Interesse" und wird nicht mehr in der regulären Vorlesung behandelt. Dennoch, wenn es sie interessiert wie man funktionale Sprachfeatures (also alles was Java 8 Streams zu tun hat) ansatzweise mit einer ganzen Batterie von objektorientierten Mitteln (Polymorphie, anonyme Klassen, Generics, Dependency Inversion Principle) nachbauen kann, sei ihnen dieses Kapitel zum Selbststudium empfohlen. Denn auch mit rein objektorientierten Sprachen kann man funktionale Prinzipien "imitieren". Die Eleganz, Kürze und Pragmatik funktionaler Sprachen bleibt dabei leider meist auf der Strecke.

In diesem Aufgabenblatt werden wir daher als erstes die Skriptsprache Groovy nutzen (um ihre funktionalen Programmierfertigkeiten wieder etwas aufzuwecken), die funktionale Programmierung out-of-the-box ermöglicht und eine Skriptsprachenerweiterung von Java ist. Diese Features von Groovy haben sie auch alle bereits mit Java 8 Streams kennengelernt.

Installieren sie sich daher in ihrer Eclipse Softwareentwicklungsumgebung bitte das

- Groovy Plugin (bitte installieren Sie das passende Plugin zu ihrer Eclipse Version)
- `http://groovy.codehaus.org/Eclipse+Plugin`

mittels des Eclipse Marketplace. Arbeiten Sie dann ferner bitte die folgenden Tutorials durch:

- `http://groovy.codehaus.org/Tutorial+1+-+Getting+started`
- `http://groovy.codehaus.org/Tutorial+2+-+Code+as+data%2C+or+closures`
- `http://groovy.codehaus.org/Differences+from+Java`
- `http://groovy.codehaus.org/Closures`
- `http://groovy.codehaus.org/Closures+-+Informal+Guide`

In diesem Aufgabenblatt sind Ihnen einige Dinge untersagt, die für sie bisher selbstverständlich beim Programmieren waren (über die sie vermutlich nicht einmal nachgedacht haben):

- Ihnen steht die If Anweisung nicht zur Verfügung, sie dürfen aber die bedingte Auswertung nutzen (?: Operator)
- Ihnen stehen keine Schleifen zur Verfügung, sie dürfen aber rekursiv programmieren oder funktionale Iteratoren wie `map`, `each`, etc. nutzen.
- Einmal mit Werten belegte Variablen dürfen sie nach der Wertebelegung nicht mehr ändern!

Sie dürfen dafür andere Dinge tun, die sie in Java bislang nicht konnten:

- Sie können Funktionen (closures) Variablen zu weisen.
- Sie können Funktionen (closures) anderen Funktionen als Parameter übergeben (das lässt sich sehr umständlich auch in Java mit generischen abstrakten Klassen annährend nachbilden, mehr dazu in Aufgabe 24.2).

24.1. Aufgabe: Funktionale Programmierung mit Groovy

Legen Sie in Eclipse ein Groovy Project an und vollziehen Sie die nachfolgenden Beispiele nach. Sie erhalten so ein Gefühl für die wesentlichsten Datenstrukturen und Programmierprinzipien funktionaler Programmiersprachen:

- Listen (Datenstruktur)
- Mappings (Datenstruktur)
- map Funktion (`collect` in Groovy)
- filter Funktion (`findAll` in Groovy)
- fold Funktion (`inject` in Groovy)
- Funktionen als Parameter für andere Funktionen

```
1  // So legen sie in Groovy eine Liste an
2  list = ["Dies", "ist", 1, "Beispiel"]
3
4  // So durchlaufen sie in Groovy diese Liste Element fuer Element
5  // und geben es auf der Console aus
6  list.each { elem -> println elem }
7
8  // Sie koennen zwei Listen aneinander haengen mit dem Plus Operator
9  l1 = [1, 2, 3]
10 l2 = [3, 4, 5]
11 println (l1 + l2)  // ergibt [1, 2, 3, 3, 4, 5]
12
13 // Sie koennen ferner pruefen, ob eine Liste einen Wert beinhaltet.
14 println ([1,2,3,4, "Schwarzes Schaf"].contains("Black Sheep")) // ergibt false
15 println ([1,2,3,4, "Black Sheep"].contains("Black Sheep")) // ergibt true
16
17 // Sie koennen in Groovy mittels Ranges auch sehr einfach Zahlenbereiche als Listen anlegen
18 // Z.B. so:
19 println (1..5)          // ergibt [1, 2, 3, 4, 5]
20 println (5..1)          // ergibt [5, 4, 3, 2, 1]
21 println ((1..10).step(2)) // ergibt [1, 3, 5, 7, 9]
22
23 // So legen sie in Groovy Variablen an und weisen ihnen Werte zu
24 // Achtung: Sie geben keine Typen an!
25 a = 1      // ganzzahlige Werte
26 b = true     // Boolesche Werte (true, false)
27 c = 3.14     // Fliesskomma Werte
28 d = "String"   // Zeichenketten mit Variableninterpolation
29 e = 'c'      // Zeichenketten ohne Variableninterpolation
30
31 // So legen sie in Groovy ein Mapping an
```

```
plz = [
  23560: "Bornkamp",
  23562: "St. Juergen",
]

// So geben sie alle Keys des Mappings aus
plz.keySet().each { key -> println key }

// So geben sie alle Values des Mappings aus
plz.values().each { value -> println value }

// So laesst sich natuerlich auch ein Mapping durchlaufen
plz.each { k, v -> println k + " -> " + v }

// Und wo wir gerade dabei sind.
// Groovy beherrscht natuerlich Variableninterpolation
// in Strings (nutzen sie einfach das $)
plz.each { k, v -> println "$k -> $v" }

// Auf die Values eines Mappings laesst sich natuerlich auch per
// Schluessel zugreifen und zwar so ...
println plz[23560] // ergibt "Bornkamp"
println plz[23562] // ergibt "St. Juergen"

// So koennen sie in Groovy eine Funktion definieren
// und einer Variablen zuweisen
q = { x -> x * x }

// So koennen sie in Groovy auf diese Funktion
// zugreifen und diese nutzen
println q(5)              // ergibt 25

// Sie koennen die Funktion auch wie folgt aufrufen
println ({x -> x * x }(5)) // ergibt ebenfalls 25

// Sie muessen die Parameter nicht einmal benennen,
// wenn sie mit dem Standardparameter it arbeiten.
// Dann wird es noch kuerzer ...
println ({ it * it }(5))   // und wieder 25

// Das geht auch mit mehrwertigen Funktionen
add = { x, y -> x + y }
println add(3, 2) // ergibt 5

// Funktionen koennen sie in Groovy auch ganz klassisch
// definieren (aber das ist eigentlich langweilig ...)
// Auf das return statement koennen sie in Groovy verzichten,
// der Wert des letzten Ausdrucks ist standardmaessig
// der return Wert einer Funktion oder einer Closure
def fac(n) {
  n == 0 ? 1 : n * fac(n - 1)
}
println fac(10)

// Jetzt wird es spannend.
// Sie koennen Funktionen anderen Funktionen als Parameter
```

```
88  // uebergeben, z.B. so.
89  facs = [1, 2, 3, 4].collect { i -> fac(i) }
90  println facs
91
92  // Einige Funktionen die auf Listen definiert sind
93  // erlauben ihnen so einen Einblick, in die Welt der funktionalen Programmierung
94  // zu erlangen.
95
96  // Die Funktion _collect_ (in vielen anderen funktionalen Sprachen zumeist _map_ genannt)
97  // laeuft ueber alle Elemente einer Liste und wendet auf die Element der Liste
98  // eine Funktion an.
99  println ([1, 2, 3, 4].collect { i -> i * i }) // ergibt [1, 4, 9, 16]
100
101 // Mittels Ranges und collect lassen sich auch sehr gut mehrdimensionale Datenstrukturen
102 // anlegen. Hier z.B. eine 3x3 Matrix.
103 println ((1..3).collect { i -> (1..3).collect { it }}) // Ergibt [[1, 2, 3], [1, 2, 3], [1,
        2, 3]]
104
105 // Die Funktion _findAll_ (in vielen anderen funktionalen Sprachen zumeist _filter_ genannt)
106 // filtert aus einer Liste von Werten diejenigen Werte, die einer Bedingung genuegen
107 println([1,2,3,4].findAll { i -> i % 2 == 0 }) // ergibt [2, 4] (alle geraden Werte)
108
109 // Die Funktion _inject_ (in vielen anderen funktionalen Sprachen zumeist _fold_
110 // oder _foldleft_ genannt) "faltet" eine Liste von Werten mittels einer Funktion zusammen,
111 // in dem nebeneinander stehende Werte miteinander ueber eine Closure verrechnet werden.
112 println([1, 2, 3, 4].inject(0, { acc, e -> acc + e }))
113 // ergibt 10, die Summe aus 1 + 2 + 3 + 4
114 // Die inject Funktion arbeitet dabei wie folgt:
115 // 0 (initialer Wert) + 1 (Listenkopf) = 1 und [2, 3, 4]
116 // 1 (Closure Ergebnis) + 2 (Listenkopf) = 3 und [3, 4]
117 // 3 (Closure Ergebnis) + 3 (Listenkopf) = 6 und [4]
118 // 6 (Closure Ergebnis) + 4 (Listenkopf) = 10 (Endergbnis) und [] (da Liste leer)
119
120 // Eine weitere hilfreiche Funktion zur Ausgabe von Listen ist die _join_ Funktion.
121 println ([1,2,3,4].join(","))   // ergibt "1,2,3,4"
```

Mehr Funktionen und Features benötigen Sie nicht, um die folgenden Aufgaben zu bearbeiten. Bitte beachten Sie, dass Ihnen nicht eine Kontrollstruktur (Schleife, Verzweigen, etc.) gezeigt wurde. Warum? Weil Sie diese nicht benötigen ... (als Multiparadigm Sprache bietet Groovy diese zwar an - aber sie sollen diese Sprachmittel bewusst nicht in diesem Aufgabenblatt nutzen).

24.1.1. Teilaufgabe A: Closures definieren

Definieren Sie eine Closure, die feststellt ob ein ganzzahliger numerischer Wert durch n teilbar ist. Wenden sie diese Closure auf alle Zahlen von 10 bis 1000 an und filtern sie alle Zahlen heraus, die durch 7 teilbar sind. Geben Sie diese in folgender Form auf der Konsole aus:

- 14
- 21
- 28
- ...

24.1.2. Teilaufgabe B: Fakultät als Closure definieren

Definieren Sie nun die Fakultätsfunktion als Closure in Grooy und wenden sie diese Closure auf alle Zahlen von 1 bis 10 an. Geben Sie dabei die Fakultät in folgender Form auf der Konsole aus:

```
fac (1) = 1
fac (2) = 2
fac (3) = 6
...
fac (10) = 3628800
```

24.1.3. Teilaufgabe C: Funktional Primzahlen bestimmen

Bestimmen Sie mittels des Siebs von Eratosthenes alle Primzahlen bis zu einer durch den Nutzer eingegebenen Zahl. Schreiben Sie hierzu eine Funktion `getAllPrims(n)`, die alle Primzahlen bis n als **absteigend** sortierte Liste liefert. Den (imperativen) Algorithmus können sie bei Wikipedia nachlesen (vgl. `http://de.wikipedia.org/wiki/Sieb_des_Eratosthenes`). Sie dürfen aber nur funktionale Sprachfeatures von Groovy nutzen. D.h.

- keine Schleifen
- keine If Anweisungen
- keine Variablen, die sie nach ihrer initialen Wertebelegung verändern

Mit folgenden Codezeilen können Sie in Groovy eine Konsoleneingabe vom Nutzer abfragen. In `N` steht dann der vom Nutzer eingegebene Wert.

```
1 System.in.withReader {
2   println "Primzahlen auflisten bis zur Zahl?"
3   N = it.readLine().toInteger()
4 }
```

Die Funktion `getAllPrims(n)` lässt sich als Dreizeiler lösen, wenn Sie folgende Hinweise beachten:

- Bestimmen Sie erst alle Zahlen von 2 bis n die keine Primzahlen sind in einer Liste über Listen.
- Fügen Sie diese Listen über Listen zu einer Liste zusammen. Diese beinhaltet dann alle Zahlen, die keine Primzahlen sind (einige Zahlen sind zwar doppelt, aber das ist letztlich egal).
- Bestimmen Sie dann alle Zahlen von 2 bis n, die nicht Bestandteil der im Schritt zwei gebildeten Liste sind.

Dieser Ansatz funktioniert vom Prinzip her genauso wie der bei Wikipedia beschriebene Ansatz, nur ändern sie nicht permanent zur Laufzeit bereits gebildete Datenstrukturen (was statusbehaftete Programmierung ist, die in der funktionalen Programmierung vermieden wird/unbekannt ist).

24.1.4. Teilaufgabe D: Funktional HTML generieren

Gegeben sei eine Liste von Werten. Wandeln Sie diese in eine HTML Liste um. Definieren Sie hierzu eine Funktion `toUnorderdList(list)`.

Bspw. soll der Aufruf von

```
println toUnorderedList(["Mein", "Name", "ist", "Hase", "!"])
```

wie folgt auf der Konsole ausgegeben werden:

```
<ul>
<li>Mein
<li>Name
<li>ist
<li>Hase
<li>!
</ul>
```

24.1.5. Teilaufgabe E: Funktional HTML Tabellen generieren

Gegeben sei eine Liste von Listen von Werten. Z.B.

```
table = [
  [ 1, 2, 3, 4, 5],
  [ 6, 7, 8, 9],
  [10, 11, 12],
  [13, 14],
  [15]
]
```

Wandeln sie beliebige derartiger Listen von Listen in HTML Tabellen um. Zweidimensionale Datenstrukturen wie `table` sollen in HTML Tabellen umgesetzt werden. Definieren Sie sich hierzu eine Funktion `toTable()`.

Bspw. soll der Aufruf von

```
println toTable(table)
```

wie folgt auf der Konsole ausgegeben werden:

```
<table>
<tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr>
<tr><td>6</td><td>7</td><td>8</td><td>9</td></tr>
<tr><td>10</td><td>11</td><td>12</td></tr>
<tr><td>13</td><td>14</td></tr>
<tr><td>15</td></tr>
</table>
```

`toTable()` soll natürlich für beliebige Listen über Listen funktionieren.

24.1.6. Teilaufgabe F: Interaktiv funktional Tabellen generieren und ausgeben

Sie sollen nun interaktiv eine Tabelle auf der Konsole ausgeben. Vom Nutzer sollen hierbei drei Werte abgefragt werden.

1. Wieviele Spalten soll die Tabelle haben?

2. Wieviele Zeilen soll die Tabelle haben?
3. Welchen Wert soll das erste Element (oben links) der Tabelle haben?

Aus diesen Angaben soll dann eine Tabelle generiert werden, deren Elemente dann zeilenweise und spaltenweise inkrementiert werden. Für die folgende Eingabe

```
1 SPALTEN = 3
2 ZEILEN = 4
3 START  = 3
```

soll ihr Programm somit folgende Tabelle auf der Konsole ausgeben:

```
3  4  5
6  7  8
9  10 11
12 13 14
```

Für die folgende Eingabe

```
1 SPALTEN = 5
2 ZEILEN = 3
3 START  = 8
```

soll ihr Programm somit folgende Tabelle auf der Konsole ausgeben:

```
8  9  10 11 12
13 14 15 16 17
18 19 20 21 22
```

Lösen sie diese Aufgabe ausschließlich mit funktionaler Programmierung. Beachten Sie dabei: Datenstrukturen im nachhinein zu verändern, ist nicht erlaubt. Es ist also nicht statthaft erst eine leere Tabelle anzulegen und diese dann schrittweise mit den Daten zu füllen. Sie müssen die Datenstruktur also mit funktionalen Sprachfeatures rein deskriptiv abhängig von der Benutzereingabe definieren!

24.2. Aufgabe: "Funktionale Programmierung" nur mit Java OO-Bordmitteln

In dieser Aufgabe sollen sie lernen, wie man mit objektorientierten Sprachmitteln anteilig funktionale Sprachfeatures "nachbauen" kann. Hierzu ist Ihnen die folgende Datenstruktur `FList<T>` gegeben, die die folgenden Methoden auf Listen definiert:

- `map` zum Anwenden einer Funktion auf alle Element einer `FList`
- `filter` zum selektieren von Elementen einer `FList` die einem zu definierendem Kriterium genügen
- `fold` zur Anwendung einer Foldfunktion, um alle Elemente einer `FList` zu einem Wert zusammenzurechnen
- `inject` um ein Element jeweils zwischen zwei Element einer `FList` einzufügen
- `zip` um zwei `FList` zu einer reißverschlussartig zusammenzuführen

```java
import java.util.Collection;
import java.util.LinkedList;

/**
 * Eine funktionale Liste (FList) ist eine Erweiterung einer normalen Liste.
 * Eine funktionale Liste hat zusaetzlich weitere Methoden, die es
 * ermoeglichen eine Liste in funktionaler Art und Weise zu verarbeiten.
 * Insbesondere Schleifen und zustandsbehaftete Programmierung sind in der
 * funktionalen Programmierung nicht vorgesehen/verboten.
 * - map   wendet eine Funktion auf jedes Element einer Liste an
 * - filter filtert Elemente aus einer Liste, die einer Filterbedingung genuegen
 * - fold  "faltet" eine Liste mittels einer Foldfunkion zu einem Wert "zusammen"
 * - inject fuegt jeweils ein Element zwischen zwei nebeneinander stehende Elemente einer
    Liste ein
 * @author Nane Kratzke
 *
 * @param <T> Typ aller Elemente der Liste
 */
public class FList<T> extends LinkedList<T> {

  /**
   * SerialVersionUID to please the compiler.
   */
  private static final long serialVersionUID = 1L;

  /**
   * Konstruktor zum Anlegen einer leeren funktionalen Liste.
   */
  public FList() { super(); }

  /**
   * Konstruktor zum Anlegen einer funktionalen Liste.
   * @param elems Elemente die der Liste beim Anlegen hinzuegefuegt werden sollen.
   */
  public FList(Collection<T> elems) {
    super();
    this.addAll(elems);
  }

  /**
   * Konstruktur zum Anlegen einer funktionalen Liste.
   * @param elems Element(e) die der Liste beim Anlegen hinzugefuegt werden sollen.
   */
  @SafeVarargs
  public FList(T... elems) {
    super();
    for (T elem : elems) { this.add(elem); }
  }

  /**
   * Map Methode.
   * Die Map Methode wendet auf jedes Element einer Liste eine Funktion an und
   * gibt diese Ergebnisse als Liste zurueck.
   * @param function Funktion die auf jedes Element der Liste angewendet werden soll
   * @return Liste von Werten
```

```
55     */
56    public <E> FList<E> map(Function<T, E> function) {
57      FList<E> ret = new FList<E>();
58      for (T e : this) { ret.add(function.call(e)); }
59      return ret;
60    }
61
62    /**
63     * Filter Methode.
64     * Die Filter Methode wendet eine boolesche Funktion auf jedes Element einer Liste an.
65     * Liefert die boolesche Funktion true, wird das Element in die Ergebnisliste aufgenommen,
66     * andernfalls wird es nicht aufgenommen.
67     * @param filter boolesche Funktion, die eine Filterbedingung definiert
68     * @return funktionale Liste von Werten, die der Bedingung der filter Funktion genuegen
69     */
70    public FList<T> filter(Function<T, Boolean> filter) {
71      FList<T> ret = new FList<T>();
72      for (T e : this) {
73        if (filter.call(e)) { ret.add(e); }
74      }
75      return ret;
76    }
77
78    /**
79     * Fold Methode.
80     * Die Fold Methode nutzt eine Foldfunktion, um eine Liste "zusammenzufalten". Dabei
81     * wird jeweils ein Element der Liste genommen und mit dem Foldergebnis der letzten
82     * Foldoperation verrechnet. Die erste Foldoperation nimmt das erste Element einer Liste
83     * und verrechnet es mit dem initialen Wert zu einem Ergebnis. Die zweite Foldoperation
84     * nimmt das zweite Element und verrechnet es mit dem ersten Foldergebnis. usw. usw.
85     * @param fold Foldfunktion
86     * @param initial Initialer Wert fuer die Foldfunktion
87     * @return aus der Liste mittels der fold Funktion errechneter Wert
88     */
89    public <E> E fold(FoldFunction<T, E> fold, E initial) {
90      E ret = initial;
91      for (T e : this) { ret = fold.call(e, ret); }
92      return ret;
93    }
94
95    /**
96     * Inject Methode.
97     * Die inject Methode fuegt jeweils zwischen zwei Elemente einer Liste ein neues ein.
98     * @param elem einzuefuegendes Element
99     * @return funktionale Liste wobei zwischen zwei Elementen der Ursprungsliste jeweils elem
           eingefuegt wurde
100    */
101   public FList<T> inject(T elem) {
102     FList<T> ret = new FList<T>();
103
104     if (this.isEmpty()) { return ret; }
105     if (this.size() == 1) {
106       ret.add(this.get(0));
107       return ret;
108     }
109
```

```
110     for (int i = 0; i < this.size() - 1; i++) {
111       ret.add(this.get(i));
112       ret.add(elem);
113     }
114     ret.add(this.get(this.size() - 1));
115     return ret;
116   }
117 
118   /**
119    * Erezugt eine neue Liste aus der gegebenen mit einer weiteren Liste
120    * in dem diese reissverschlussartig zusammen gefuegt werden.
121    * Beispielliste 1: [a, b, c]
122    * Beispielliste 2: [A, B, C, D]
123    * Erzeugt: [a, A, b, B, c, C, D]
124    * @param list "einzuzippende" Liste
125    * @return Liste die abwechselnd aus Elementen der einen und der anderen Liste besteht (
       siehe Bsp.)
126    */
127   public FList<T> zip(FList<T> list) {
128     if (list == null) { return new FList<T>(this); }
129     FList<T> tozip = new FList<T>(list);
130     FList<T> ret = new FList<T>();
131     for (T elem : this) {
132       ret.add(elem);
133       if (!tozip.isEmpty()) { ret.add(tozip.remove(0)); }
134     }
135     ret.addAll(tozip);
136     return ret;
137   }
138 }
```

Wie sie der Implementierung entnehmen können, werden zwei abstrakte generische Klassen genutzt, um einwertige Funktionen `Function <I, O>` für die `map` und `filter` Methoden

```
1  /**
2   * Eine Function ist eine abstrakte Klasse, die dazu genutzt wird
3   * eine Funktion fuer die map Methode der FunctionalList Klasse
4   * definieren zu koennen.
5   * @author Nane Kratzke
6   *
7   * @param <I> Eingabetyp
8   * @param <O> Ausgabetyp
9   */
10 public abstract class Function<I, O> {
11 
12   /**
13    * Die map Methode einer FunctionalList ruft diese Methode auf
14    * um alle Elemente einer FunctionalList mit einer Methode zu verarbeiten.
15    * Die Methode ist dabei frei durch den Programmierer definierbar.
16    * Er muss dazu nur diese abstrakte Methode mit seiner Implementierung
17    * realisieren
18    * @param input Eingabewert vom Typ O
19    * @return Ausgabewert vom Typ O
20    */
21   public abstract O call(I input);
22 
```

```
}
```

sowie zweiwertige Funktionen des Typs `FoldFunction<I, O>` für die `fold` Methode

```
/**
 * Eine FoldFunction ist eine abstrakte Klasse, die dazu genutzt wird
 * eine Funktion fuer die fold Methode der FunctionalList Klasse
 * definieren zu koennen.
 * @author Nane Kratzke
 *
 * @param <I> Input Typ
 * @param <O> Output Typ und Typ des letzten Berechnungsergebnisses
 */
public abstract class FoldFunction<I, O> {

  /**
   * Fold Funktion.
   * Diese Methode verrechnet Element fuer Element einer Liste beginnend mit einem
   * initialen Wert. Der naechste zu verrechnende Wert wird ueber den Parameter next
   * uebergeben, der letzte berechnete Wert wird mittels last uebergeben.
   * Die Methode muss die Werte fuer next und last gem. einer zu implementierenden
   * Logik miteinander verrechnen und zurueckgeben.
   * @param next neue Eingabe fuer die Foldfunktion
   * @param last letztes Ergebnis der Foldfunktion
   * @return fold(next, last) : O
   */
  public abstract O call(I next, O last);

}
```

definieren zu können.

Die Wirkungsweise dieser drei Datenstrukturen erarbeiten Sie sich bitte anhand der folgenden Beispiele.

```
/**
 * Beispiele um die Wirkungsweise der Klassen FList, Function und FoldFunction
 * deutlich zu machen.
 * @author Nane Kratzke
 */
public class Example {

  /**
   * Hauptprogramm.
   * @param args Kommandozeilenparameter (werden nicht ausgewertet)
   */
  public static void main(String[] args) {
    // Anlegen einer funktionalen Liste
    FList<Integer> is = new FList<Integer>(1, 2, 3, 4, 5, 6, 7);

    // Definition einer Quadrierungsfunktion
    Function<Integer, Integer> quadriere = new Function<Integer, Integer>() {
      public Integer call(Integer input) { return input * input; }
    };
    System.out.println(is.map(quadriere)); // sollte [1, 4, 9, 16, 25, 36, 49] ergeben

    // Definition und Anwendung der Quadrierungsfunktion auf eine Liste in einem Schritt
    FList<Integer> quadriert = is.map(new Function<Integer, Integer>() {
```

```
24      public Integer call(Integer input) {
25        return input * input;
26      }
27    });
28    System.out.println(quadriert); // sollte [1, 4, 9, 16, 25, 36, 49] ergeben
29
30    // Definition und Anwenden der "Geradefunktion" auf eine Liste (in einem Schritt)
31    FList<Integer> gerade = is.filter(new Function<Integer,Boolean>() {
32      public Boolean call(Integer input) {
33        return input % 2 == 0;
34      }
35    });
36    System.out.println(gerade);
37
38    // Definition der Summenfunktion und Anwenden dieser mittels fold (in einem Schritt)
39    Integer result = is.fold(new FoldFunction<Integer, Integer>() {
40      public Integer call(Integer next, Integer last) {
41        return last + next;
42      }
43    }, 0);
44    System.out.println("Die Summe der Liste " + is + " ist " + result);
45
46    // Funktionsweise der zip Methode
47    FList<Integer> even = gerade; // [2, 4, 6]
48    FList<Integer> uneven = is.filter(new Function<Integer, Boolean>() {
49      public Boolean call(Integer input) {
50        return input % 2 == 1;
51      }
52    }); // [1, 3, 5, 7]
53    System.out.println(even.zip(uneven)); // sollte [2, 1, 4, 3, 6, 5, 7] ausgeben
54
55    // Funktionsweise der Inject Methode
56    System.out.println(new FList<String>("A", "B", "C").inject("-"));
57    // sollte [A, -, B, -, C] ausgeben
58
59    // Inject und fold koennen bspw. auch zur komfortablen Zeichenkettenerzeugung aus Listen
          genutzt werden
60    FoldFunction<String, String> concat = new FoldFunction<String, String>() {
61      // Funktionale Definition der Zeichenkettenkonkatenation
62      public String call(String next, String last) { return last + next; }
63    };
64    System.out.println(new FList<String>("A", "B", "C").inject(" - ").fold(concat, ""));
65    // Sollte A-B-C ergeben
66  }
67 }
```

24.2.1. Teilaufgabe A: Funktionale Listenverarbeitung

Bestimmen Sie die Summe aller quadrierten geraden Zahlen einer Liste mit den Werten von 1 bis n. Für n = 5 sollen ihr Programm auf der Konsole folgendes ausgeben. Sie dürfen nur die Methoden der Klasse `FList` sowie `Function` und `FoldFunction` Objekte zur Lösung nutzen.

```
1 + 2 + 3 + 4 + 5 = 15
```

24.2.2. Teilaufgabe B: Fakultäten funktional von 1 bis n berechnen

Bestimmen Sie die Fakultät fuer die Zahlen von 1 bis n. Geben Sie diese in folgender Form auf der Konsole aus (z.B. für n = 10).

```
fac (1) = 1
fac (2) = 2
fac (3) = 6
...
fac (10) = 3628800
```

Nutzen Sie für Ihre Lösung nur die Möglichkeiten, die Ihnen die `FList` Klasse und deren vorgesehenen Erweiterungen bieten.

Hinweis: Die Fakultätsfunktion dürfen Sie rekursiv als eigene Methode definieren.

24.2.3. Teilaufgabe C: HTML Listen generieren

Definieren Sie eine Methode `toList` mit folgender Aufrufsignatur

```
String toList(String, FList, String)
```

die eine funktionale Liste in eine HTML Liste umwandelt.

Die Methode soll bspw. für folgenden exemplarischen Aufruf

```
1 System.out.println(toList("<ol>", new FList<Integer>(1, 2, 3, 4), "</ol>"));
```

folgende Konsolenausgabe

```
<ol>
<li>1</li>
<li>2</li><li>3</li><li>4</li>
</ol>
```

erzeugen.

Der exemplarische Aufruf

```
1 System.out.println(
2   toList("<ul>", new FList<String>("Mein", "Name", "ist", "Hase"), "</ul>")
3 );
```

soll hingegen diese Konsolenausgabe

```
<ul>
<li>Mein</li>
<li>Name</li>
<li>ist</li>
<li>Hase</li>
</ul>
```

erzeugen.

Nutzen Sie für Ihre Lösung nur die Möglichkeiten, die Ihnen die `FList` Klasse und deren vorgesehenen Erweiterungen bieten.

24.2.4. Teilaufgabe D: HTML Tabellen generieren

Definieren Sie eine generische Methode `toTable` mit folgender Aufrufsignatur

```
<T> String toTable(FList<FList<T>>)
```

die eine funktionale Liste über funktionale Listen beliebigen Typs `T` in eine HTML Tabelle umwandelt.

Die Methode soll bspw. für folgenden exemplarischen Aufruf

```
1 FList<FList<String>> t1 = new FList<FList<String>>(
2   new FList<String>("A", "B", "C"),
3   new FList<String>("1", "2", "3", "4"),
4   new FList<String>("5", "6", "7"),
5   new FList<String>("8", "9"),
6   new FList<String>()
7 );
8
9 System.out.println(toTable(t1));
```

folgende Konsolenausgabe

```
<table>
<tr><td>A</td><td>B</td><td>C</td></tr>
<tr><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>5</td><td>6</td><td>7</td></tr>
<tr><td>8</td><td>9</td></tr>
<tr></tr>
</table>
```

erzeugen.

Glückwunsch! Geschafft ...

... nur noch vier Semester bis zum *"God of Programming"* ;-)

Dank an ...

Auch dieser Katalog ist nicht perfekt und wird nur mit jedem Semester besser. Studierende oder Mitarbeiter/innen finden Fehler in Aufgabenstellungen oder entwickeln Lösungen, die einfach genialer sind, als die Musterlösungen. Vielen Dank für alle Hinweise und Ideen zu Lösungen oder Aufgaben, die häufig dem Prinzip "Think outside the box" folgen.

- **Eric Massenberg** und **Rebecca Wunderlich**, Informationstechnologie und Design, Tutor Bullet Proofing, WiSe 2012/13 und SoSe 2013
- **Georg Schnabel**, Informatik, Tutor Bullet Proofing, WiSe 2012/13 und SoSe 2013
- **Michael Breuker**, Idee zu Aufgabe 23.2, SoSe 2013
- **Patrick Schuster**, Informatik, Fehler in Aufgabe 11.2 gefunden, WiSe 2013/14
- **Chris Deter** und **René Kremer**, Informatik, Musterlösung zu Aufgabe 11.3 robuster gemacht, WiSe 2013/14
- **Finn Kothe,** Informationstechnologie und Design, Unsauberkeit in Aufgabe 16.3 gefunden und präzisere Formulierung der Aufgabenstellung 16.4 vorgeschlagen, SoSe 2014
- **Florian Löhden**, Informatik, Idee wie die Musterlösung zur Aufgabe 14.3 wesentlich kompakter ausgedrückt werden kann, WiSe 2014/15.
- **Dario Lehmhus** und **Tim Faltin**, Informationstechnologie und Design, Entwicklung der ersten optimalen TicTacToe Strategie WinnerWinnerChickenDinner seit Aufnahme des Testfalls gegen den ZufallsSpieler nicht verlieren zu dürfen (Aufgabe 21.2), SoSe 2015.
- **Torge Tönnies**, Informatik, Entwicklung der optimalen TicTacToe Strategie Hacker (Aufgabe 21.2), SoSe 2015. Sehr rafinierter Ansatz mit einer Reflection-basierten Lösung OO-Schutzmechanismen auszuhebeln, um ungestraft betrügen zu können (das geht aber mittlerweile nicht mehr - mein Bedauern für alle die das zukünftig vorhaben).
- **Marco Torge Gabrecht** und **Jan-Marco Bruhns** (inspiriert von **Robert Vagt**), Informatik, Entwicklung einer wirklich eleganten Lösung der Knobelaufgabe 5.5, WiSe 2015/16. Dieser Ansatz kommt gänzlich ohne Typecasting aus und liegt einfach auf der Hand, wenn man erst einmal darauf gekommen ist. Vor allem ist diese Lösung viel eleganter als die ehemalige Musterlösung. Damit ist mal wieder der Beweis erbracht worden, dass Typecasts meist überflüssig sind.
- **Marco Torge Gabrecht**, Informatik, für die YouTube Playlist mit Lösungserläuterungen zu Aufgaben des WS 2015/16 (`https://www.youtube.com/playlist?list=PL06-G-mnBYTJ2QffB3jgjZH-1TVIwMDr8`).
- **Max Sternitzke**, Informatik, Fehlerhaftes Beispiel in Aufgabe 7.6 gefunden und diverse Typos in Codebeispielen, WiSe 2015/16.
- **Duc Tu Le Anh**, Informatik, Fehler in Aufgabenstellung 17.4 gefunden, SoSe 2016.
- **Yevhenii Vasylenko**, fehlerhaftes JavaDoc in Aufgabe 19.2 gefunden, SoSe 2016.
- **Marco Gabrecht**, fehlerhafter Hinweis in Aufgabe 20.2 gefunden, SoSe 2016.
- **Robert Vagt**, interessante Mastermind Strategie (TicTacToe, Aufgabe 21.2), SoSe 2016.
- **Alenka Rixen**, Informationstechnologie und Design, Cowsay (Aufgabe 3.4) mittels `replaceAll()` vereinfacht, WiSe 2016/17.
- Die geniale Aufgabenidee zu Possible Chessmen (Aufgabe 16.5) hat sich mein 8-jähriger Sohn **Tjorben** ausgedacht (vermutlich basiert sie auf Aufgaben seines Schachlehrers **Michael Weiss**). Ich habe die Aufgabenstellung nur etwas für die Programmierbarkeit präzisiert.
- **David Engelbrecht**, Fehler in Aufgabenstellung whereIsTheMax und Aufgabe 10.6 und PossibleChessmen (Aufgabe 16.5) gefunden, WiSe 2016/17 und SoSe 2017.

Bei allen die ich vergessen habe, entschuldige ich mich hiermit und verspreche sie in diese Liste aufzunehmen.