

Praktikum Programmierung

Gruppenaufteilung

Das Praktikum findet in zwei Gruppen statt. Beachten Sie bitte, dass Sie nur an dem Ihnen zugewiesenen Termin teilnehmen können. Arbeiten Sie einzeln am Rechner.

Anwesenheitspflicht

Für das Praktikum besteht Anwesenheitspflicht. Tragen Sie sich bitte an jedem Termin in die Anwesenheitsliste ein. Die regelmäßige Anwesenheit und die aktive Teilnahme am Praktikum sind Voraussetzung für das Bestehen des Praktikums. Es wird erwartet, dass alle Studierenden den eigenen Programmcode und die dabei verwendeten Konzepte erklären können.

Abgabe

Zu jedem Praktikumstermin gibt es ein Aufgabenblatt mit mehreren Aufgaben. Zum Bestehen des Praktikums müssen Sie diese selbstständig und erfolgreich bearbeiten. Schicken Sie allen von Ihnen erstellten Programmcode an: prog.eisenbiegler@fh-furtwangen.de. Schicken Sie pro Aufgabe eine E-Mail mit folgendem Aufbau:

- **To (An):** prog.eisenbiegler@fh-furtwangen.de
- **Subject (Betreff):** Geben Sie im Subject hintereinander die Nummer der Aufgabe und Ihren Namen an. Beispiel: 2.4 Michael Maier.
- **Body (Rumpf):** Der Body der Nachricht soll allen Programmcode enthalten. Haben Sie mehrere Klassen geschrieben, so hängen Sie die einzelnen Textstücke bitte hintereinander. Tipp: Übertragen Sie den Programmcode einfach mit Kopieren-Einfügen von der Entwicklungsumgebung in Ihre E-Mail. Verwenden Sie keine Attachements (Anhänge)!

Abgabetermin

Auf jedem Aufgabenblatt finden Sie einen Abgabetermin. Gelingt es Ihnen während des Praktikums nicht, die Aufgaben vollständig zu bearbeiten, so können Sie dies bis zum Abgabetermin nachholen.

Eigene Laptops

Wer einen eigenen Laptop besitzt, kann diesen gerne zum Praktikum mitbringen und darauf programmieren. Die im Praktikum verwendete Entwicklungsumgebung Eclipse ist im Internet für alle gängigen Betriebssysteme frei verfügbar (<http://www.eclipse.org>).

Aufgabenblatt 1

Praktikum am 6.10.2015
keine Abgabe

Programmierung
Wintersemester 2015/2016
Prof. Dr. Dirk Eisenbiegler
Hochschule Furtwangen

Aufgabe 1.1 - Programmieren mit dem SDK

Ziel dieser Aufgabe ist es, das nachfolgende Programm zu übersetzen und auszuführen. In dieser Aufgabe soll dazu das Software Development Kit verwendet werden.

```
public class ErstesProgramm {  
    public static void main(String[] args) {  
        System.out.println("Herzlich willkommen zum Praktikum!");  
        System.out.println("Eine Zufallszahl: " + Math.random());  
        System.out.println("Ende des Programms.");  
    }  
}
```

A) Starten Sie den Texteditor gedit. Das Start-Icon für gedit finden Sie auf dem Desktop links oben. Speichern Sie die Datei unter dem Namen ErstesProgramm.java im Verzeichnis /home/hfu.

B) Starten Sie das Terminal. 

C) Übersetzen Sie den Programmcode mit dem Kommandozeilenbefehl

```
javac ErstesProgramm.java
```

D) Führen Sie das Programm mit dem folgenden Kommandozeilenbefehl aus

```
java ErstesProgramm
```

E) Fügen Sie in dem Programm die nachfolgende Zeile nach der vierten Zeile ein. Übersetzen Sie das Programm erneut und führen Sie es aus.

```
System.out.println("Gleich ist das Programm zu Ende.");
```

F) Der Programmtext enthält ein Pluszeichen +. Ersetzen Sie dieses durch eine Minuszeichen -. Versuchen Sie erneut, das Programm zu übersetzen. Anmerkung: Es müsste jetzt zu einem Übersetzungsfehler kommen. Das Minuszeichen ist an dieser Stelle nicht zulässig.

Aufgabe 1.2 - Programmieren mit Eclipse


In dieser Aufgabe soll das gleiche Programm mit der Entwicklungsumgebung Eclipse ausgeführt werden.

A) Eclipse starten

Starten Sie Eclipse, falls noch nicht geschehen. Sie finden das Start-Icon für Eclipse links oben auf Ihrem Desktop.

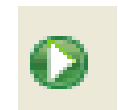
B) Ein Programm starten

Das Programm wurde bereits für Sie eingetippt. Sie sehen es nach dem Start in dem Editor vor sich. Um das Programm zu starten, gehen Sie wie folgt vor: Gehen Sie mit dem Cursor in den Programmtext. Drücken Sie dann mit der

Maus neben dem Start-Knopf  auf das kleine schwarze Dreieck, das nach unten zeigt. Sie sehen dann ein Pulldown-Menü. Wählen Sie dort den Punkt "Run As" und dort wiederum "Java Application". Jetzt wird da Programm übersetzt und gestartet.

C) Ein Programm erneut starten

Wenn Sie ein Programm bereits einmal gestartet haben und es danach noch einmal starten wollen, dann müssen Sie lediglich noch auf den Start-Knopf




drücken (nicht mehr über das kleine schwarze Dreieck rechts daneben).

Die Bedeutung des Start-Knopfes: Das zuletzt gestartete Programm noch einmal starten.

D) Fügen Sie die nachfolgende Programmzeile nach der vierten Zeile in Ihr Programm ein. Starten Sie das Programm erneut

```
System.out.println("Gleich ist das Programm zu Ende.");
```

E) Der Programmtext enthält ein Pluszeichen +. Ersetzen Sie dieses durch eine

Minuszeichen - und speichern Sie die Datei mit . Eclipse weist Sie auf einen Syntaxfehler hin. Ersetzen Sie das Minuszeichen wieder durch ein

Pluszeichen und speichern Sie die Datei erneut mit .

F) Wenn Sie Eclipse verlassen, werden all Ihre Daten gespeichert: Ihr Programmcode und auch alle anderen Einstellungen. Wenn Sie Eclipse wieder starten erscheint der Bildschirm genau so wie Sie ihn zuletzt verlassen haben. Probieren Sie es aus: Eclipse verlassen, Eclipse neu starten.

G) All Ihre Programmdateien und Einstellungen werden im Verzeichnis

/home/hfu/workspace

abgespeichert. Ihren Programmcode finden Sie unter

/home/hfu/workspace/Programmierung/src

Die übersetzten Java-Programme finden Sie unter /

/home/hfuroot/workspace/Programmierung/bin

Starten Sie dieses analog zu Teilaufgabe 1.1 mit dem SDK (Kommandozeilen-Befehl: java).

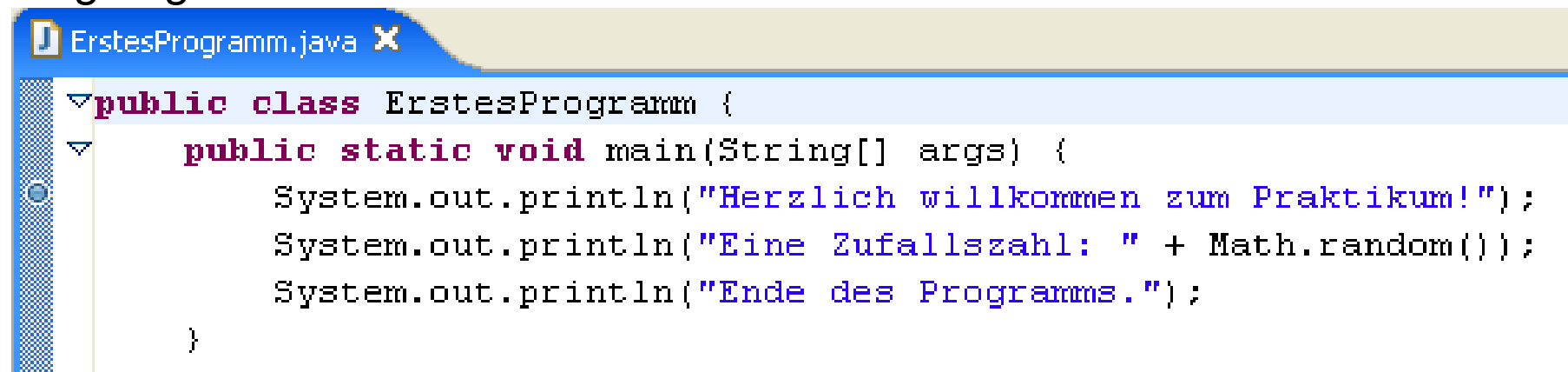
Aufgabe 1.3 - Debugging

In dieser Aufgabe soll das Programm von Aufgabe 1.2 Schritt für Schritt ausgeführt werden („Debugging“).


A) Breakpoint setzen

Setzen Sie einen „Breakpoint“ vor die dritte Zeile. Ein Breakpoint ist in Eclipse durch eine kleine blaue Kreisfläche dargestellt (siehe Abbildung).



Doppelklicken Sie an die entsprechende Stelle und der Breakpoint wird eingefügt.



B) Programm im Debug-Modus starten

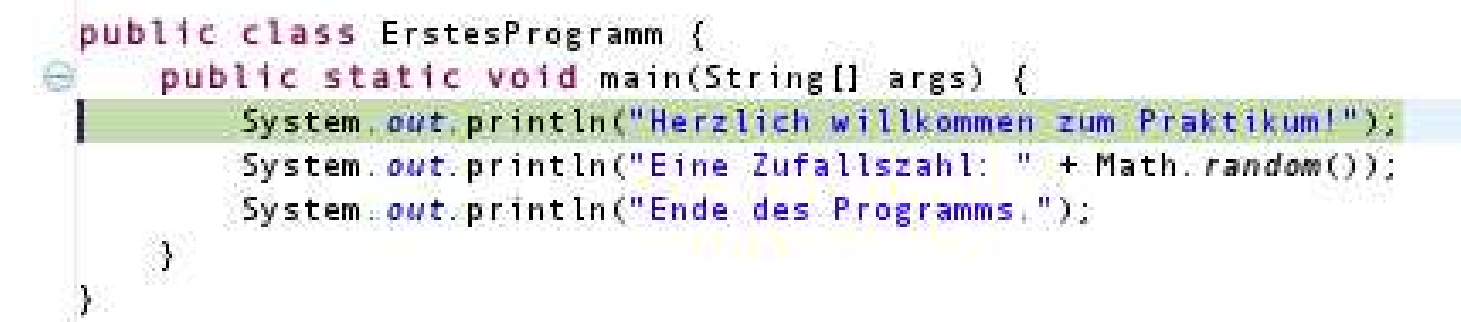
Um das Programm zu im Debug-Modus zu starten, gehen Sie wie folgt vor: Gehen Sie mit dem Cursor in den Programmtext. Drücken Sie dann mit der Maus neben dem Debug-Knopf  auf das kleine schwarze Dreieck, das nach unten zeigt. Sie sehen dann ein Pulldown-Menü. Wählen Sie dort den Punkt „Debug As“ und dort wiederum „Java Application“. Jetzt wird das Programm übersetzt und gestartet. Das Programmablauf ist wie ein gewöhnlicher Programmablauf, nur dass das Programm an Breakpoints anhält und von dort aus Schritt für Schritt ausgeführt werden kann. Bei einem gewöhnlichen Start-Vorgang (ohne Debug-Modus) werden Breakpoints ignoriert.

C) Views und Perspectives




Das Eclipse-Fenster besteht aus mehreren rechteckigen Bereichen (Editoren etc.) , die als Views bezeichnet werden. Eine Zusammenstellung mehrerer Views bildet eine Perspective. Rechts oben im Eclipse-Fenster befinden sich die Knöpfe  und  damit können Sie zwischen der Java- und der Debug-Perspective hin und her-wechseln.

D) Das Programm Schritt für Schritt ausführen


Durch einen Debug-Start wechselt Eclipse automatisch in die Debug-Perspective. Eclipse zeigt Ihnen an, dass das Programm an einer bestimmten Zeile (dem Breakpoint) angehalten wurde. Der Befehl, der als nächstes ausgeführt werden soll, ist grau hinterlegt.



Jetzt können Sie das Programm durch mehrmaliges Drücken des Knopfes


 Befehl für Befehl ausführen. Der Knopf  bewirkt, dass das Programm bis zum nächsten Breakpoint springt. Gibt es im Programm keinen weiteren Breakpoint, so läuft das Programm bis zum Ende. Drücken Sie diesen Knopf, wenn Sie den letzten Befehl erreicht haben. Der Knopf  bewirkt, dass das Programm abgebrochen wird.

E) Ein Programm erneut im Debug-Modus starten

Wenn Sie ein Programm bereits einmal gestartet haben und es danach noch einmal starten wollen, dann müssen Sie lediglich noch auf den Debug-Knopf  drücken (nicht mehr über das kleine schwarze Dreieck rechts daneben). Die Bedeutung des Debug-Knopfes: Das zuletzt gestartete Programm noch einmal starten und zwar im Debug-Modus.

Aufgabe 1.4 - Neue Klasse anlegen

A) Legen Sie eine Klasse mit dem Namen Multiplizierer an:

- ⇒ Drücken Sie 
- ⇒ Geben Sie als *Source Folder* Programmierung/src an.
- ⇒ Geben Sie als *Name* Multiplizierer an.
- ⇒ Setzen Sie ein Häkchen links von *public static void main(String[] args)*.
- ⇒ Drücken Sie auf Finish.
Unter dem Projekt Programmierung erscheint jetzt im Source-Folder (src) und dort im Default-Package die neue Klasse mit dem Namen Multiplizierer. Das Gerüst der Klasse Multiplizierer und auch das Gerüst der Methode main ist bereits von Eclipse generiert worden.

B) Fügen Sie in die main-Methode der Klasse Multiplizierer die nachfolgenden Zeilen ein:

```
int x = Integer.parseInt(args[0]);
int y = Integer.parseInt(args[1]);
int z = x * y;
System.out.println(x + " mal " + y + " ergibt " + z +
".");
```

Aufgabe 1.5 - Programme starten und dabei Kommandozeilenparameter übergeben

A) Nachdem Sie die Klasse Multiplizierer abgespeichert hat, übersetzt Eclipse den Programmcode automatisch und erzeugt die Datei
/root/workspace-prog/Programmierung/bin/Multiplizierer.class
Das Programm Multiplizierer erwartet, dass ihm beim Start die beiden Faktoren (zu multiplizierenden Werte) als Kommandozeilenparameter übergeben werden.


Starten Sie die Konsole. Gehen Sie dann in das Verzeichnis mit der übersetzten Datei

```
cd /root/workspace-prog/Programmierung/bin
```

und starten Sie das Programm wie folgt:

```
java Multiplizierer 4 17
```

B) Auch mit Eclipse können Sie beim Start Kommandozeilenparameter übergeben. Gehen Sie dazu wie folgt vor:

- ⇒ Wählen Sie in Eclipse die Datei Multiplizierer aus.
- ⇒ Drücken Sie auf das schwarze Dreieckchen rechts neben dem Start-Knopf 
- ⇒ Drücken Sie *Open Run Dialog ...*
- ⇒ Wählen Sie *Java Application*.
- ⇒ Gehen Sie in das Tab Parameters und tragen Sie dort unter Program Parameters mit Leerzeichen getrennt
4 17
ein.

Aufgabe 1.6 - Variableninhalte beobachten

Das folgende Programm berechnet alle Zweierpotenzen bis zu einer vorgegebenen Höhe.

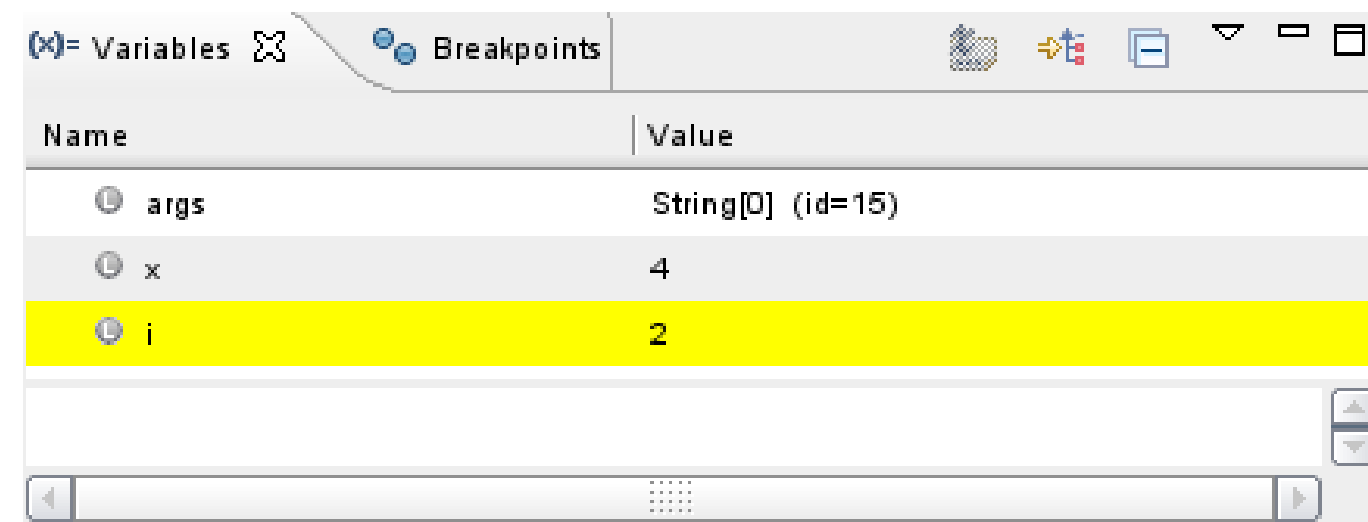
```
public class Zweierpotenzen {  
    public static void main(String[] args) {  
        int x = 1;  
        int i = 0;  
        while (x <= 10) {  
            System.out.println(i + " : " + x);  
            x = x * 2;  
            i = i+1;  
        }  
    }  
}
```

A) Erzeugen Sie eine Klasse mit dem Namen Zweierpotenzen und übernehmen Sie obigen Programmcode. Starten Sie die Klasse.

B) Setzen Sie vor die Zeile

`int x = 1;`

einen Breakpoint und gehen Sie mit dem Debugger das Programm Schritt für Schritt durch. Verfolgen Sie den Ablauf des Programms und beobachten Sie die Variableninhalte. Die Variablen und ihre Werte werden rechts oben dargestellt. Variablen, die ihren Werte im letzten Schritt gerade geändert haben, werden gelb markiert dargestellt.



Aufgabenblatt 2

*Praktikum am 13.10.2015
Abgabe bis spätestens 19.10.2015*

*Programmierung
Wintersemester 2015/2016
Prof. Dr. Dirk Eisenbiegler
Hochschule Furtwangen*

Aufgabe 2.1 - MinMax

- A) Schreiben Sie ein Programm mit dem Klassennamen MinMaxDrei. MinMaxDrei soll von drei Programmparametern das Minimum und das Maximum bestimmen. Zunächst soll das Programm alle drei Programmparameter ausgeben, dann das Minimum und schließlich das Maximum. Testen Sie das Programm mit verschiedenen Eingabewerten. Gehen Sie das Programm mit dem Debugger Schritt für Schritt durch.
- B) Schreiben Sie analog zu Teilaufgabe A) ein Programm mit dem Klassennamen MinMax, das zu einer beliebigen Anzahl von Eingabewerten das Minimum und das Maximum bestimmt.
Hinweis: *args.length* ist die Länge des Arrays *args* - also die Anzahl der Programmparameter.

Aufgabe 2.2 - Palindrom

Schreiben Sie ein Programm, das von einem String prüft, ob er ein Palindrom ist.

Schreiben Sie dazu zunächst eine Methode mit dem Namen *palindrom*. Die Methode *palindrom* hat einen Parameter vom Typ *String* und einen Rückgabewert vom Typ *boolean*, der genau dann den Wert *true* haben soll, wenn der String von links nach rechts gelesen das gleiche Wort ergibt wie von rechts nach links.

Beispiele: "OTTO", "ANNA" und "xYzYx" sind Palindrome, "Katze" ist kein Palindrom.

Die main-Methode soll für den ersten Kommandozeilenparameter entweder den Satz

Der String *xy* ist ein Palindrom.

oder den Satz

Der String *xy* ist kein Palindrom.

ausgeben. Dazu soll in der main-Methode die *palindrom*-Methode verwendet werden.

Hinweise:

- x* Ist *x* eine Variable vom Typ *String*, dann ist *x.length()* die Länge des Strings und *x.charAt(i)* der *i*-te Buchstabe (mit $0 \leq i < x.length()$).
- x* Den Klassennamen können Sie in dieser Aufgabe und auch in den weiteren Aufgaben frei wählen (soweit nicht explizit angegeben). Es empfiehlt sich als Klassennamen den Aufgabennamen zu verwenden. Allerdings müssen Sie beachten, dass Klassennamen immer mit einem Buchstaben beginnen müssen. Empfehlung: Stellen Sie A oder Aufgabe voran. Außerdem darf der Klassennamen keine Leerzeichen und keine Punkte enthalten. Empfehlung: Sie können stattdessen das Underscore-Zeichen `_` verwenden.

Aufgabenblatt 3

*Praktikum am 20.10.2015
Abgabe bis spätestens 26.10.2015*

*Programmierung
Wintersemester 2015/2016
Prof. Dr. Dirk Eisenbiegler
Hochschule Furtwangen*

Aufgabe 3.1 - Summe, Durchschnitt

- A) Schreiben Sie eine Methode, die zu zu einem Array von int-Werten die Summe berechnet.
- B) Schreiben Sie eine Methode, die zu einem Array von int-Werten das arithmetische Mittel bestimmt. Verwenden Sie dabei die Methode aus Teilaufgabe A).

Hinweis:

Als Arithmetisches Mittel der Zahlen x_0, x_1, \dots, x_{n-1} bezeichnet man den Wert $\frac{x_0 + x_1 + \dots + x_{n-1}}{n}$.

Aufgabe 3.2 - Muster

Betrachten Sie die folgenden Muster, die aus den Wörtern "Olive" und "Furtwangen" gebildet wurden.

```
OOOOO
Ol111
Oliii
Olivv
Olive

FFFFFFFFF
Fuuuuuuuuu
Furrrrrrrrr
Furtttttttt
Furtwwwww
Furtwaaaaa
Furtwannnn
Furtwanggg
Furtwangee
Furtwangen
```

Schreiben Sie eine Methode mit dem Namen *muster*, die zu einem beliebigen String ein solches Muster ausgibt.

Aufgabenblatt 4

Praktikum am 27.10.2015
Abgabe bis spätestens 2.11.2015

Programmierung
Wintersemester 2015/2016
Prof. Dr. Dirk Eisenbiegler
Hochschule Furtwangen

Aufgabe 4.1 - Vorkommen von Zeichen zählen

Schreiben Sie eine Methode mit dem Namen *vorkommen*, die einen String zeilenweise Buchstabe für Buchstabe ausgibt und zu jedem Buchstaben angibt, wie oft er im String vorkommt.

Beispiel: Ananasmarmelade

A 1
n 2
a 4
n 2
a 4
s 1
m 2
a 4
r 1
m 2
e 2
l 1
a 4
d 1
e 2

Aufgabe 4.2 - Teilstring suchen

Schreiben Sie eine Methode mit dem Namen *substring*, die in einem String x nach einem Teilstring y durchsucht. Ist y in x als Teil einmal oder mehrfach enthalten, so soll die erste Position zurückgegeben an der y in x beginnt. Die Zählung der Buchstaben beginne beim ersten Buchstaben mit 0. Ist y nicht in x enthalten, so soll -1 zurückgegeben werden.

Beispiel 1:

x ="Hustensaft"

y ="Husten"

Ergebnis: 0

Beispiel 2:

x ="Hustensaft"

y ="saft"

Ergebnis: 6

Beispiel 3:

x ="Ananas"

y ="na"

Ergebnis: 1

Beispiel 4:

x ="Kamel"

y ="Pferd"

Ergebnis: -1

Aufgabe 4.3 - Nullstellensuche durch Intervallschachtelung

Schreiben Sie zwei Methoden mit den Namen f und *nullstelle*.

Die Methode f soll die Funktion $f(x) = e^x + x^2 - 4$ realisieren. Der Parameter und der Rückgabewert der Funktion f sollen beide vom Typ *double* sein.

Die Methode *nullstelle* soll in einem Intervall $[x,y]$ per Intervallschachtelung nach einer Nullstelle suchen. Das Iterationsverfahren soll erst dann abbrechen, wenn ein Wert m gefunden wird, sodass $|f(m)| < z$ gilt. Ist ein solcher Wert m gefunden, so soll die Methode diesen Wert zurückgeben. Die Werte x , y und z sind Parameter der Methode *nullstelle*. Alle Parameter und der Rückgabewert von *nullstelle* seien vom Typ *double*. Implementieren Sie die Methode *nullstelle* mit Hilfe von Rekursion.

Testen Sie die Methode *nullstelle* mit den Werten $x=-1.0$, $y=20.0$, $z=0.0001$ sowie mit den Werten $x=-100.0$, $y=0.0$, $z=0.0001$.

Hinweise:

x Die Java-Funktion $\text{Math.exp}(x)$ berechnet e^x .

x Die Java-Funktion $\text{Math.abs}(x)$ berechnet $|x|$.

Es darf angenommen werden, dass von den beiden Werten $f(x)$ und $f(y)$ einer der beiden Werte größer als 0 ist und der andere kleiner als 0.

Aufgabenblatt 5

*Praktikum am 3.11.2015
Abgabe bis spätestens 9.11.2015*

*Programmierung
Wintersemester 2015/2016
Prof. Dr. Dirk Eisenbiegler
Hochschule Furtwangen*

Aufgabe 5.1 - Rekursion (Fibonacci)

```
public class Fibonacci {  
    public static int fibonacci (int n) {  
        if (n == 0)  
            return 1;  
        if (n == 1)  
            return 1;  
        return fibonacci(n-1) + fibonacci(n-2);  
    }  
}
```

- A) Ohne das Programm zu starten: Bestimmen Sie die Werte von `Fibonacci.fibonacci` für $n = 0, 1, 2, 3, 4, 5, 7$.
- B) Wie verhält sich die Methode `Fibonacci.fibonacci`, wenn sie mit dem Parameter `-1` gestartet wird?
- C) Implementieren Sie `Fibonacci.fibonacci` neu. Die neue Implementierung soll ohne Rekursion auskommen. Für Zahlen kleiner als 0 soll der Rückgabewert 0 sein - für Zahlen größer oder gleich 0 soll der Rückgabewert wie in der bisherigen Implementierung sein.

Aufgabe 5.2 - Rekursion (Fakultät)

In dieser Aufgabe soll die Fakultät $n!$ einer natürlichen Zahl n auf zwei unterschiedliche Arten berechnet werden. Unter der Fakultät einer natürlichen Zahl n versteht man das Produkt aller Zahlen von 1 bis n .

- A) Schreiben Sie eine Methode, die die Fakultät berechnet. Der einzige Parameter ist eine int-Zahl n , der Rückgabewert ist vom Typ int. Berechnen Sie das Ergebnis, indem Sie in einer for-Schleife die Zahlen von 1 bis n in einer Variable aufmultiplizieren.
- B) Schreiben Sie eine Methode mit der gleichen Funktion wie in Teilaufgabe A). Berechnen Sie die Fakultät diesmal jedoch ohne for-Schleife, sondern mit Rekursion. Verwenden Sie die folgende Rekursionsgleichung:

$$n! = \begin{cases} 1 & \text{falls } n=1 \\ (n-1)! \cdot n & \text{falls } n>1 \end{cases}$$

Aufgabenblatt 6

*Praktikum am 10.11.2015
Abgabe bis spätestens 16.11.2015*

*Programmierung
Wintersemester 2015/2016
Prof. Dr. Dirk Eisenbiegler
Hochschule Furtwangen*

Aufgabe 6.1 - Library einbinden

A) Library in das Eclipse-Projekt kopieren

Im Verzeichnis /home/hfu/picture befindet sich die Library mit dem Dateinamen picture.jar. Fügen Sie diese Datei zu dem Eclipse-Projekt hinzu. Gehen Sie dazu in das Dateien-Werkzeug (Icon auf der linken Leiste im Desktop), wählen Sie die Datei aus und kopieren Sie diese: Datei anklicken, rechte Maustaste, kopieren. Gehen Sie dann zu Eclipse, wählen Sie das aktuelle Projekt (Name: Programmierung) und führen Sie dort eine Einfügeoperation durch: rechte Maustaste, „einfügen“.

B) Library zum Build-Path hinzufügen.

In dem Eclipse-Projekt „Programmierung“ ist jetzt die Datei picture.jar zu sehen. Wählen Sie diese Datei mit der Maus aus, drücken Sie die rechte Maustaste und wählen Sie im Menü den Punkt „Build-Path“ und dort „Add To Build Path“. Ab sofort können Sie in dem Projekt die in der Library enthaltenen Klassen verwenden.

Aufgabe 6.2 - Bildverarbeitung

In dieser Aufgabe sollen die beiden Methoden `Picture.load` und `Picture.show` verwendet werden. Die Klasse `Picture` ist Bestandteil von `picture.jar`.

Der Methodenaufruf

`Picture.load("/home/hfu/picture/MyPicture.jpg")`
lädt eine Bilddatei und gibt als Rückgabewert ein zweidimensionales Array `p` vom Typ `int[][]` zurück, in dem die Pixel des Bildes abgespeichert sind. Mit dem Befehl `Picture.show(p)` kann das Bild angezeigt werden.

```
int p[][] = Picture.load("/home/hfu/picture/MyPicture.jpg");
Picture.show(p);
```

In der Array-Variablen `p[x][y]` befindet sich die Farbinformation für den Punkt mit den Koordinaten `(x,y)`. Die Punkte sind in dem zweidimensionalen Array wie folgt angeordnet:

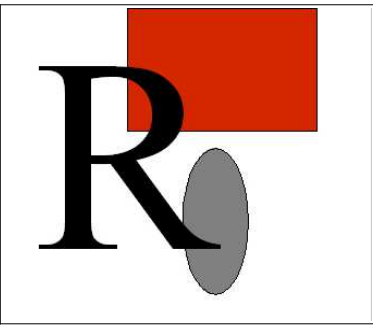
p[0][0]	p[1][0]	p[2][0]	p[3][0]	p[4][0]	p[5][0]
p[0][1]	p[1][1]	p[2][1]	p[3][1]	p[4][1]	p[5][1]
p[0][2]	p[1][2]	p[2][2]	p[3][2]	p[4][2]	p[5][2]
p[0][3]	p[1][3]	p[2][3]	p[3][3]	p[4][3]	p[5][3]

Die Breite des Bildes ist `p.length`, die Höhe ist `p[0].length`. Damit liegt `x` im Bereich von 0 bis `p.length-1` und `y` im Bereich 0 bis `p[0].length-1`.

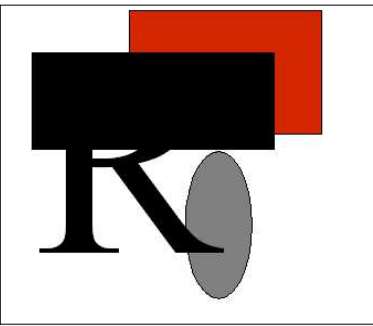
In den nachfolgenden Teilaufgaben sollen die Bilder durch Programme verändert werden, indem die Array-Werte verändert werden. Das Programm soll immer folgenden Aufbau haben: das Bild wird eingelesen, dann verändert, dann angezeigt.

```
int p[][] = Picture.load("/root/picture/MyPicture.jpg");
// hier ist der Code zur Änderung des Bildes einzufügen
Picture.show(p);
```

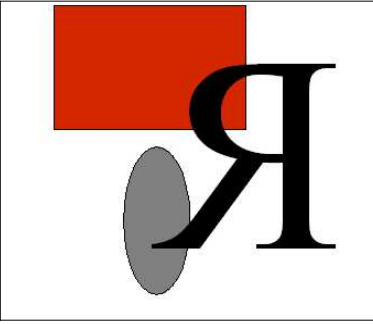
Zur Illustration der Transformationen wird das folgende Beispielbild verwendet:



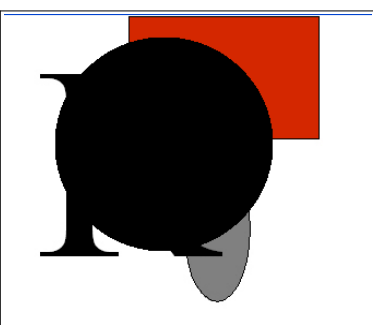
A) Fügen Sie ein schwarzes Rechteck ein. Die Koordinaten der linken oberen Ecke seien `(30,70)`, die der Ecke rechts unten `(250,160)`. Die Farbe Schwarz wird durch den Wert 0 repräsentiert.



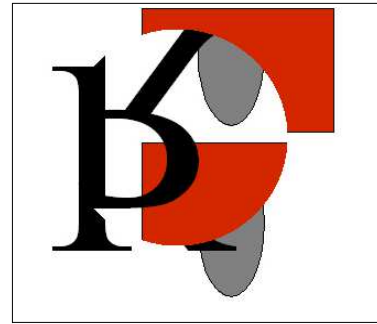
B) Spiegeln Sie das Bild horizontal.



C) Färben Sie alle Punkte in einer Kreisfläche schwarz. Mittelpunkt der Kreisfläche ist `(150,150)`, der Radius ist 100. Hinweis: Bestimmen Sie zu jedem Punkt den Abstand zum Kreismittelpunkt und entscheiden Sie dann, ob Sie ihn schwarz färben.

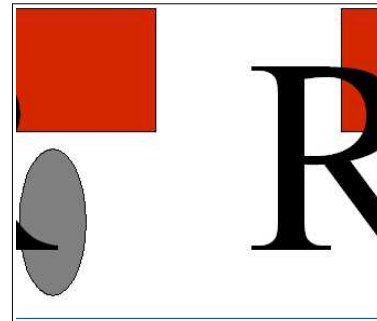


D) Gegeben der Kreis mit Mittelpunkt `(150,150)` und dem Radius 100. Spiegeln Sie innerhalb des Kreise alle Punkte an einer horizontalen Achse, die durch den Mittelpunkt geht.

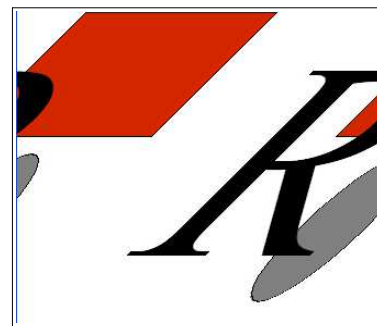


E) Verschieben Sie das Bild um 180 Punkte nach rechts. Lassen Sie das Bild dabei "rotieren", sodass die nach rechts hinausgeschobenen Punkte von links in das Bild geschoben werden.

Hinweis: Führen Sie die Rotation Zeile für Zeile aus. Implementieren Sie zunächst eine Rotation um einen Punkt.



F) Scheren Sie das Bild um 45 Grad.



Aufgabenblatt 7

Praktikum am 17.11.2015
Abgabe bis spätestens 23.11.2015

Programmierung
Wintersemester 2015/2016
Prof. Dr. Dirk Eisenbiegler
Hochschule Furtwangen

Aufgabe 7.1 - Objekte

Tippen Sie die drei Klassen ab. Starten Sie die Klasse *Tester* mit dem Debugger und gehen sie die main-Methode Zeile für Zeile durch. Beobachten Sie dabei die Variablenwerte und insbesondere auch die Inhalte der Objekte.

```
public class Farbe {  
    public int rot;  
    public int grün;  
    public int blau;  
    public Farbe(int r, int g, int b) {  
        rot = r;  
        grün = g;  
        blau = b;  
    }  
}
```

```
public class FarbigerPunkt {  
    public double x;  
    public double y;  
    public Farbe farbe;  
}
```

```
public class Tester {  
    public static void main (String args[]) {  
        Farbe fa = new Farbe(13,14,15);  
        fa.rot = 3;  
        fa.blau = 4;  
        Farbe fb = new Farbe(27,1,1);  
        fa = fb;  
        fa.rot = 17;  
        fa.grün = 18;  
        fb = null;  
        FarbigerPunkt fp = new FarbigerPunkt();  
        fp.x = 5.1;  
        fp.y = 6.6;  
        fp.farbe = fa;  
        fp.farbe.grün = 44;  
        fp.farbe.blau = 47;  
        fb.grün = 8;  
    }  
}
```


Aufgabe 7.2 - Tageszeit

- A) Schreiben Sie eine Klasse mit dem Namen *Tageszeit*. Die Klasse soll drei Objektattribute vom Typ *int* haben: *stunden*, *minuten* und *sekunden*.
- B) Schreiben Sie einen Konstruktor mit den drei Parametern *stunden*, *minuten* und *sekunden*.
- C) Schreiben Sie einen zweiten Konstruktor mit zwei Parametern *stunden* und *minuten*. Die Sekunden sollen von diesem Konstruktor auf 0 gesetzt werden.
- D) Fügen Sie zur Klasse *Tageszeit* eine Objektmethode mit dem Namen *sekundenSeitMitternacht* hinzu, die zu der Tageszeit die Anzahl der Sekunden seit Mitternacht bestimmt. Die Methode hat keinen Parameter und einen Rückgabewert vom Typ *int*. Testen Sie die Methode.
- E) Fügen Sie zur Klasse *Tageszeit* eine Objektmethode mit dem Namen *vormittags* hinzu, die zu der Tageszeit bestimmt, ob es Vormittag ist (vor 12:00). Die Methode hat keinen Parameter und liefert einen Rückgabewert vom Typ *boolean* zurück. Testen Sie die Methode.
- F) Fügen Sie zur Klasse *Tageszeit* eine Objektmethode mit dem Namen *toString()* hinzu, die die Tageszeit in einen String umwandelt.
Format: 14:34.46
Die Methode hat keinen Parameter und liefert einen Rückgabewert vom Typ *String* zurück. Testen Sie die Methode.
- G) Schreiben Sie eine Objektmethode mit dem Namen *vorstellen*. Diese Methode soll drei Parameter vom Typ *int* haben: *stunden*, *minuten* und *sekunden*. Die Methode soll keinen Rückgabewert haben. Die Methode soll die Uhrzeit um die angegebene Anzahl von Stunden, Minuten und Sekunden vorstellen (im Uhrzeigersinn nach vorne verstellen).
- H) Fügen Sie zur Klasse *Tageszeit* eine Objektmethode mit dem Namen *istFrueherAls* hinzu, die bestimmt, ob die Tageszeit früher ist als eine zweite Tageszeit, die der Methode als Parameter übergeben wird. Die Methode hat einen Parameter vom Typ *Tageszeit* und einen Rückgabewert vom Typ *boolean*. Tipp: verwenden Sie dazu die Methode *sekundenSeitMitternacht*. Testen Sie die Methode.

- I) Fügen Sie zur Klasse *Tageszeit* ein statisches Attribut *ampm* vom Typ *boolean* hinzu. Das statische Attribut soll festlegen, ob die *toString*-Methode die Zeit im am/pm-Format ausgeben soll oder in der 24h-Darstellung. Passen Sie die Methode *toString()* entsprechend an. Testen Sie die Methode *toString*.

<i>am/pm-Format</i>	<i>24h-Format</i>
7:13.45 am	7:13.45
2:00.12 pm	14:00.12

Aufgabenblatt 8

Praktikum am 24.11.2015
Abgabe bis spätestens 30.11.2015

Programmierung
Wintersemester 2015/2016
Prof. Dr. Dirk Eisenbiegler
Hochschule Furtwangen

Aufgabe 8.1 - Liste

Die Klasse Liste realisiert eine Liste von int-Werten.

```
public class Liste {  
  
    public int element;  
    public Liste nachfolger;  
  
    public Liste (int w) {  
        element = w;  
        nachfolger = null;  
    }  
  
    public void hinzufuegen (int w) {  
        if (nachfolger == null)  
            nachfolger = new Liste(w);  
        else  
            nachfolger.hinzufuegen(w);  
    }  
}
```

```
public class ListeTester {  
    public static void main(String[] args) {  
        Liste x = new Liste(3);  
        x.hinzufuegen(5);  
        x.hinzufuegen(4);  
        x.hinzufuegen(7);  
    }  
}
```

- A) Tippen Sie die beiden Klassen ab und gehen Sie mit dem Debugger Zeile für Zeile durch die main-Methode von ListeTester.
- B) Fügen Sie zur Klasse *Liste* die Methode *laenge* hinzu, die die Länge der Liste bestimmt. Die Methode *laenge* ist eine Objektmethode. Sie hat keinen Parameter und einen Rückgabewert vom Typ *int*.
- C) Fügen Sie zur Klasse *Liste* die Methode *entfernen* hinzu, die das letzte Element der Liste entfernt. Die Methode *entfernen* ist eine Objektmethode. Sie hat keinen Parameter und keinen Rückgabewert (*void*). Enthält die Liste nur ein einziges Element, so soll die Methode *entfernen* keine Änderung vornehmen.
- D) Fügen Sie zur Klasse *Liste* die Methode *toString* hinzu, die die Liste in Form

eines Strings zurückgibt. In dem String sollen die Elemente durch Kommata getrennt aufgelistet werden. Die Methode *toString* ist eine Objektmethode. Sie hat keinen Parameter und einen Rückgabewert vom Typ String.

- E) Fügen Sie zur Klasse Liste die Methode *summe* hinzu, die von einer Liste die Summe der Werte zurückgibt. Die Methode *summe* hat keinen Parameter und einen Rückgabewert vom Typ int.
- F) Fügen Sie zur Klasse Liste die Objektmethode *addiere* hinzu, die zu jedem Element der Liste einen vorgegebenen Wert hinzuaddiert. Die Methode *addiere* hat einen Parameter vom Typ *int* und keinen Rückgabewert.

Aufgabe 8.2 - Binärer Suchbaum

In dieser Aufgabe soll in Java ein Graph realisiert werden: ein binärer Suchbaum.

Binärer Suchbaum:

- x Jeder Knoten enthält eine andere Zahl.
- x Für jeden Knoten gilt:
Die Zahlen im linken Teilbaum sind kleiner als der Wert im Knoten, und die Zahlen im rechten Teilbaum sind größer als der Wert im Knoten.

Implementierungsidee:

Die Knoten des Baums sind Instanzen einer Klasse mit dem Namen Knoten. Ein Baum ist somit eine Menge von Knoten-Objekten. Den Kanten zwischen den Knoten entsprechen Referenzen: Eine Kante von Knoten a zu Knoten b wird durch ein Attribut in a realisiert, in der die Objektreferenz auf b gespeichert wird. Mit anderen Worten: Ein Attribut von a zeigt nach b.

Eine Klasse mit dem Namen Baum gibt es nicht explizit. Jeder Knoten steht für einen Baum: der Baum, dessen Wurzel er ist.

- A) Implementieren Sie eine Klasse mit dem Namen *Knoten*. Die Klasse *Knoten* soll ein Attribut *x* vom Typ int haben, in dem die Zahl gespeichert wird. Ferner soll die Klasse Knoten die beiden Attribute *links* und *rechts* vom Typ Knoten haben. *links* und *rechts* sind Referenzen auf die Nachbarknoten im linken bzw. rechten Teilbaum. Existiert kein solcher Teilbaum, so sollen deren Werte *null* sein.
- B) Implementieren Sie einen Konstruktor für die Klasse *Knoten*. Der Konstruktor habe einen Parameter vom Typ int - die Zahl, die in dem Knoten gespeichert werden soll. Der Konstruktor soll die Attribute *links* und *rechts* auf null setzen. Es entsteht ein binärer Suchbaum mit einem Knoten.
- C) Implementieren Sie in der Klasse *Knoten* die Objektmethode *einfüegen*. Die Methode *einfüegen* habe genau einen Parameter vom Typ int. Sie fügt ein neues Element in den Baum ein, dessen Wurzel der Knoten ist. Sollte das Element bereits im Baum enthalten sein, so soll die Methode keine Wirkung haben.
- D) Implementieren Sie in der Klasse *Knoten* die Objektmethode *toString*. Die

Methode *toString* habe keinen Parameter. Sie soll den Baum, dessen Wurzel der Knoten ist, in einen String umwandeln. In dem String sollen jeweils der linke Teilbaum, der Zahlenwert und der rechte Teilbaum durch runde Klammern umschlossen werden.

Beispiele:

- | | |
|------------------------|---|
| (3) | Baum mit einem Knoten dessen Wert 3 ist. |
| ((1) 3) | Baum, dessen Wurzel den Wert 3 hat und dessen linker Teilbaum den Wert 1 hat. |
| ((((1) 2 (5)) 8 (12))) | größerer Baum mit den Werten 1, 2, 5, 8 und 12 |

- E) Implementieren Sie in der Klasse *Knoten* die Objektmethode *suchen*. Die Methode *suchen* habe genau einen Parameter vom Typ *int* und einen Rückgabewert vom Typ *boolean*. Sie soll prüfen, ob der angegebene Wert im Baum enthalten ist, dessen Wurzel der Knoten ist.
Rückgabewert true: das Element ist enthalten.
Rückgabewert false: das Element ist nicht enthalten.

F) Optionale Zusatzaufgabe

Modifizieren Sie die bisherige Implementierung so, dass der binäre Suchbaum zum AVL-Baum wird.

- ⇒ Erinnerung AVL-Baum:
Die Höhe des linken und des rechten Teilbaums dürfen sich um nicht mehr als um eins unterscheiden (AVL-Eigenschaft).
- ⇒ Tipp: Ein Suchbaum mit einem Knoten erfüllt immer die AVL-Eigenschaft. Die AVL-Eigenschaft muss bei Einfügeoperationen erhalten bleiben. Dazu sind ggf. "Rotationen" erforderlich.

Tipp: Speichern Sie in jedem Knoten in einem zusätzlichen Attribut die Höhendifferenz zwischen linken und rechtem Teilbaum. Vorsicht: Dieses Attribut soll nach allen Operationen auf dem Baum immer den richtigen Wert anzeigen.

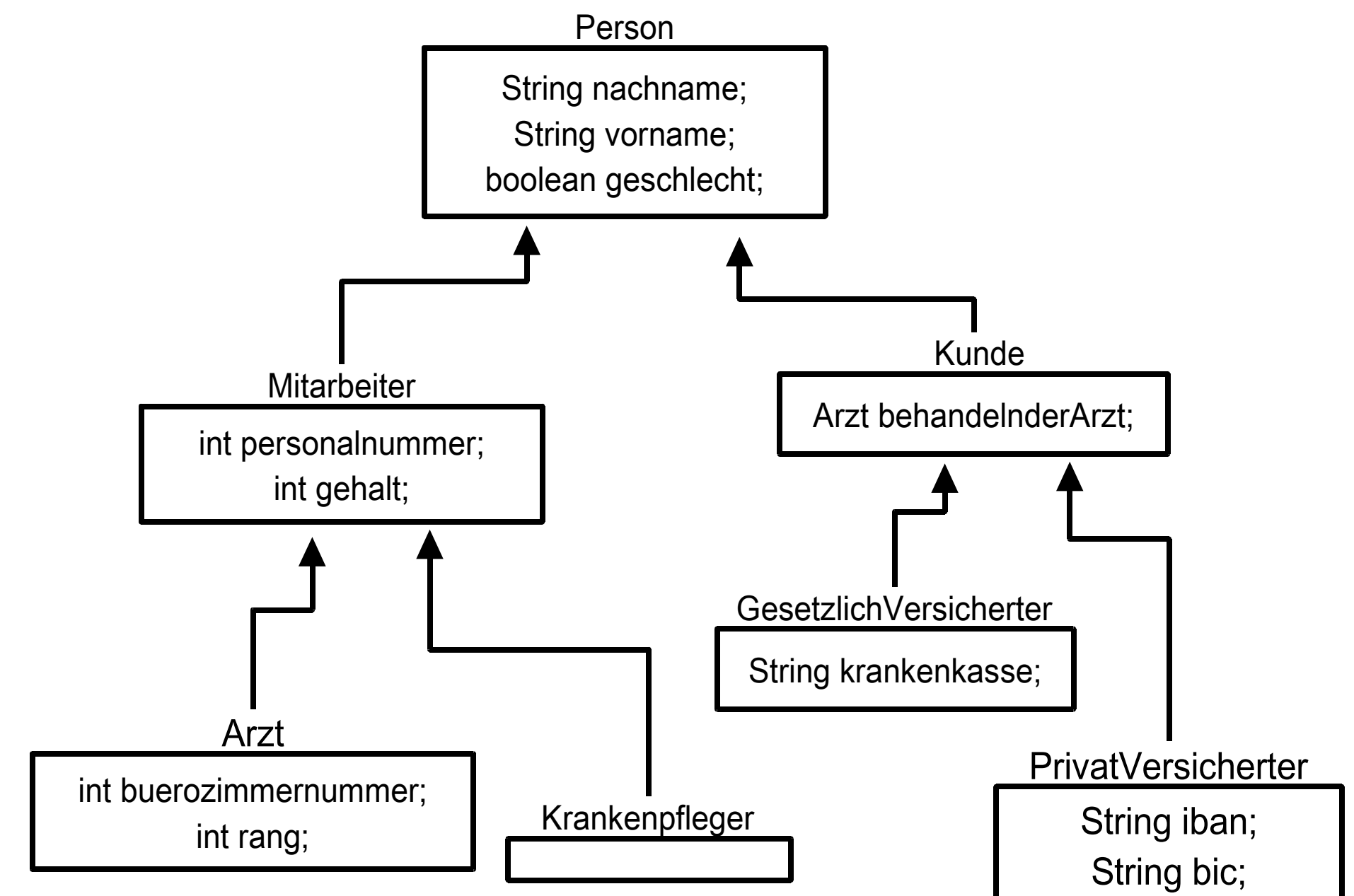
Aufgabenblatt 9

Praktikum am 1.12.2015
Abgabe bis spätestens 7.12.2015

Programmierung
Wintersemester 2015/2016
Prof. Dr. Dirk Eisenbiegler
Hochschule Furtwangen

Aufgabe 9.1 - Personen im Krankenhaus

In dieser Aufgabe sollen Klassen geschrieben werden, deren Objekte die Daten von Kunden und Mitarbeitern in einem Krankenhaus darstellen. Implementieren Sie die Klassen wie unten beschrieben und testen Sie die Funktionalität der Klassen.



A) Schreiben Sie eine Klasse mit dem Namen *Person* und den folgenden nicht-statischen Attributen:

Name des Attributs	Typ des Attributs
nachname	String
vorname	String
geschlecht	boolean

Bei dem Attribut *geschlecht* stehe der Wert *true* für weiblich und *false* für männlich.

B) Fügen Sie zur Klasse *Person* die Objektmethode *anrede* hinzu. Die Methode *anrede* habe keinen Parameter und einen Rückgabewert vom Typ String. Der String soll bei Frauen den Wert "Frau Vorname Nachname" und bei Männern den Wert "Herr Vorname Nachname" haben.

C) Deklarieren Sie eine Klasse mit dem Namen *Mitarbeiter*. Die Klasse *Mitarbeiter* ist Sohn der Klasse *Person*. Zusätzlich habe die Klasse *Mitarbeiter* die folgenden Objektattribute:

Name des Attributs	Typ des Attributs
personalnummer	int
gehalt	int

D) Deklarieren Sie eine Klasse mit dem Namen *Arzt*. Die Klasse *Arzt* ist Sohn der Klasse *Mitarbeiter*. Zusätzlich habe die Klasse *Arzt* die folgenden Objektattribute:

Name des Attributs	Typ des Attributs
buerozimmernummer	int
rang	int

Das Attribut *rang* habe die Werte 0, 1, 2 und 3.
0 stehe für AIP (Arzt im Praktikum), 1 stehe für Facharzt, 2 stehe für Oberarzt und 3 stehe für Chefarzt.

E) Überschreiben Sie in der Klasse *Arzt* die Methode *anrede*, sodass für die einzelnen Ärzte, je nach Rang und je nach Geschlecht, die folgenden Anreden verwendet werden
"Vorname Nachname (AIP)"
"Fachärztin Vorname Nachname"
"Oberärztin Vorname Nachname"
"Chefärztin Vorname Nachname"
"Facharzt Vorname Nachname"
"Oberarzt Vorname Nachname"
"Chefarzt Vorname Nachname"

F) Deklarieren Sie eine Klasse mit dem Namen *Krankenpfleger*. Die Klasse *Krankenpfleger* ist Sohn der Klasse *Mitarbeiter*. Die Klasse *Krankenpfleger* habe keine zusätzlichen Objektattribute.

G) Überschreiben Sie in der Klasse *Krankenpfleger* die Methode *anrede*, sodass für die einzelnen *Krankenpfleger*, je nach Geschlecht, die folgenden Anreden

verwendet werden
"Krankenschwester Vorname Nachname"
"Krankenpfleger Vorname Nachname"

H) Fügen Sie zur Klasse *Krankenpfleger* die Methode *formloseAnrede* hinzu. Die Methode *formloseAnrede* habe keinen Parameter und einen Rückgabewert vom Typ String. In diesem Rückgabewert soll so sein wie bei *anrede* in der Klasse *Person*. Verwenden Sie deshalb in *formloseAnrede* die Methode *anrede* der Klasse *Person*.

I) Deklarieren Sie eine Klasse mit dem Namen *Kunde*. Die Klasse *Kunde* ist Sohn der Klasse *Person*. Zusätzlich habe die Klasse *Kunde* das folgende Objektattribut:

Name des Attributs	Typ des Attributs
behandelnderArzt	Arzt

J) Deklarieren Sie eine Klasse mit dem Namen *GesetzlichVersicherter*. Die Klasse *GesetzlichVersicherter* ist Sohn der Klasse *Kunde*. Zusätzlich habe die Klasse *GesetzlichVersicherter* das folgende Objektattribut:

Name des Attributs	Typ des Attributs
krankenkasse	String

K) Deklarieren Sie eine Klasse mit dem Namen *PrivatVersicherter*. Die Klasse *PrivatVersicherter* ist Sohn der Klasse *Kunde*. Zusätzlich habe die Klasse *PrivatVersicherter* die folgenden Objektattribute:

Name des Attributs	Typ des Attributs
iban	String
bic	String

L) Schreiben Sie zu *Mitarbeiter* eine Methode mit dem Namen *verdientMehrAls*. Diese Methode habe genau einen Parameter vom Typ *Mitarbeiter* und einen Rückgabewert vom Typ boolean, der angibt, ob der Mitarbeiter mehr verdient als der im Parameter angegebene.
Testen Sie diese Methode, indem Sie dieser Methode nacheinander Objekte vom Typ *Mitarbeiter*, *Arzt* und *Krankenpfleger* übergeben.

M) Schreiben Sie zu *Kunde* eine Methode mit dem Namen *arztVerdientMehrAls*, die die Gehälter der behandelnden Ärzte zweier Kunden vergleicht. Die Methode habe genau einen Parameter vom Typ *Kunde* und einen Rückgabewert vom Typ boolean, der angibt, ob der Kunde einen

behandelnden Arzt hat, der mehr verdient als der Arzt des im Parameter angegebenen Kunden. Verwenden Sie dabei die Methode *verdientMehrAls* der Klasse *Mitarbeiter*.

Aufgabe 9.2 - Personen-Liste

Diese Aufgabe baut auf den Klassen von Aufgabe 6.2 auf.

In dieser Aufgabe soll eine Personenliste für alle Personen im Krankenhaus realisiert werden. Personen sollen hinzugefügt und entfernt werden können, und es soll angegeben werden können welche Personen anwesend sind und welche nicht.

A) Schreiben Sie eine Klasse mit dem Namen *Personenliste* mit den folgenden Objektattributen.

Name des Attributs	Typ des Attributs
<i>anzahl</i>	int
<i>personen</i>	Person []
<i>anwesend</i>	boolean []

Erläuterungen:

Das Attribut *anzahl* stehe für die Anzahl der Personen, die in der Liste gespeichert sind. Das Array *personen* dient zur Speicherung der Personen. Die Länge des Array sei 1000. Die Personen-Objekte werden in den ersten Positionen des Arrays gespeichert (0 ... *anzahl*-1). Es soll angenommen werden, dass nie mehr als 1000 Personen gespeichert werden müssen.

Für jede Person wird im Array *anwesend* gespeichert, ob sie anwesend ist oder nicht. Die Länge des Array *anwesend* sei ebenfalls 1000. Die Anwesenheit der Person, die sich im Array *personen* an Position *i* befindet, soll im Array *anwesend* ebenfalls an der Position *i* gespeichert werden.

- B) Schreiben Sie für die Klasse *Personenliste* einen Konstruktor, der eine leere Personenliste erzeugt. Der Konstruktor soll keinen Parameter haben.
- C) Schreiben Sie für die Klasse *Personenliste* eine Methode mit dem Namen *hinzufuegen*. Die Methode habe einen Parameter vom Typ *Person* und keinen Rückgabewert. Die Methode soll eine weitere Person zur Personenliste hinzufügen und deren Anwesenheit auf *false* setzen. Ist die Personenliste bereits voll (*anzahl* == 1000), so soll die Methode keine weitere Person mehr hinzufügen.
- D) Schreiben Sie für die Klasse *Personenliste* eine Methode mit dem Namen *entfernen*. Die Methode habe einen Parameter vom Typ *Person* und keinen

Rückgabewert. Die Methode soll eine Person aus der Personenliste entfernen. Ist die Person nicht in der Personenliste enthalten, so ändert die Methode die Personenliste nicht.

Hinweis: Verschieben Sie beim Entfernen alle nachfolgenden Personen im Array *personen* um eins nach links. Analog für *anwesend*.

- E) Schreiben Sie für die Klasse *Personenliste* eine Methode mit dem Namen *kommt*. Die Methode habe einen Parameter vom Typ *Person* und keinen Rückgabewert. Die Methode soll die Anwesenheit der Person auf *true* setzen.
- F) Schreiben Sie für die Klasse *Personenliste* eine Methode mit dem Namen *geht*. Die Methode habe einen Parameter vom Typ *Person* und keinen Rückgabewert. Die Methode soll die Anwesenheit der Person auf *false* setzen.
- G) Schreiben Sie für die Klasse *Personenliste* eine Methode mit dem Namen *findeArzt*. Die Methode habe die Parameter *vorname* und *nachname*, beide vom Typ String, und einen Rückgabewert vom Typ Arzt. Die Methode soll in der Liste nach einem Arzt mit dem angegebenen Namen suchen und diesen zurückgeben.

Aufgabenblatt 10

Praktikum am 8.12.2015
Abgabe bis spätestens 14.12.2015

Programmierung
Wintersemester 2015/2016
Prof. Dr. Dirk Eisenbiegler
Hochschule Furtwangen

Aufgabe 10.1 - Stapel

In dieser Aufgabe soll eine Klasse implementiert werden, die einen Stapel realisiert.

Die Klasse soll einen Konstruktor und zwei Methoden zur Verfügung stellen:

- x Der Konstruktor hat keinen Parameter und soll einen leeren Stapel erzeugen.
- x Die Methode *push* legt ein Objekt auf den Stapel.
- x Die Methode *pull* nimmt ein Objekt vom Stapel und gibt es als Rückgabewert zurück.

Detailbeschreibung:

- x Die Objekte auf dem Stapel sollen vom Typ *Object* sein.
- x Verwenden Sie zur Implementierung der Klasse die Klasse *Vector*.

A) Implementieren Sie die Klasse *Stapel* wie beschrieben.

⇒ Empfehlung: Testen Sie die Klasse mit Objekten vom Typ *String*.

B) Überschreiben Sie in *Stapel* die Methode *toString()* so, dass die Stapelinhalte durch Kommata getrennt aufgezählt werden.

⇒ Beispiel: "auto,katze,hund"

C) Überschreiben Sie in *Stapel* die Methode *equals()* in sinnvoller Weise: Vergleich zweier Stapel (paarweiser Vergleich mit der *equals()*-Methode der Elemente)

D) Überschreiben Sie in *Stapel* die Methode *clone()* so, dass eine Kopie des Stapels erzeugt wird.

E) Schreiben Sie eine Klasse mit dem Namen *IntStapel*, die vom Prinzip genau so funktioniert wie die Klasse *Stapel*, nur dass mit *push* und *pull* anstelle von Objekten vom Typ *Object* Werte vom skalaren Typ *int* abgelegt und abgenommen werden sollen. Verwenden Sie dazu die bereits implementierte Klasse *Stapel* und die Klasse *Integer*.

Aufgabe 10.2 - Funktionen

In dieser Aufgabe soll eine abstrakte Klasse mit dem Namen *Funktion* implementiert werden, die Abbildungen von *double* nach *double* repräsentiert.

Die Klasse *Funktion* soll die folgende abstrakt Methode enthalten.

```
public abstract double f (double x);
```

A) Implementieren Sie die Klasse *Funktion* als eine abstrakte Klasse.

B) Implementieren Sie eine Sohnklasse von *Funktion* mit dem Namen *QuadratischeFunktion*. Die Klasse *QuadratischeFunktion* hat zusätzlich drei Attribute *a*, *b* und *c* - alle vom Typ *double*. Die Klasse soll einen Konstruktor haben, dem die Werte dieser Attribute als Parameter übergeben werden. Die Klasse soll nicht-abstrakt sein, d.h. die Methode *f* soll überschrieben werden. Die Funktion, die *f* realisieren soll, sei:

$$f(x) = a \cdot x^2 + b \cdot x + c$$

C) Implementieren Sie in der Klasse *Funktion* eine Methode mit dem Namen *nullstellensuche*, die innerhalb eines Intervalls $[r, s]$ durch Intervallschachtelung nach einer Nullstelle sucht. Die Intervallschachtelung soll abbrechen, sobald ein *x* gefunden wird mit $|f(x)| \leq t$. Wird keine Nullstelle gefunden, so soll *r-1* zurückgegeben werden. Die Methode *nullstellensuche* hat drei Parameter: *r*, *s* und *t*. Alle drei Parameter sind vom Typ *double*. Der Rückgabewert ist ebenfalls vom Typ *double*. Testen Sie die Methode, indem Sie eine Instanz von *QuadratischeFunktion* bilden.

D) Implementieren Sie eine nicht-abstrakte Sohnklasse von *Funktion* mit dem Namen *TrigonometrischeFunktion*. Die Klasse *TrigonometrischeFunktion* hat zusätzlich zwei Attribute *p* und *q* - beide vom Typ *double*. Die Klasse soll einen Konstruktor haben, dem die Werte dieser Attribute als Parameter übergeben werden. Die Funktion, die *f* realisieren soll, sei:

$$f(x) = \sin(p \cdot x + q)$$

E) Implementieren Sie eine nicht-abstrakte Sohnklasse von *Funktion* mit dem Namen *SummenFunktion*. Die Klasse *SummenFunktion* hat zusätzlich zwei Attribute *g* und *h* - beide vom Typ *Funktion*. Die Klasse soll einen Konstruktor haben, dem die Werte dieser Attribute als Parameter übergeben werden. Die Funktion, die *f* realisieren soll, sei:

$$f(x) = g(x) + h(x)$$

Aufgabenblatt 11

Praktikum am 15.12.2015
Abgabe bis spätestens 21.12.2015

Programmierung
Wintersemester 2015/2016
Prof. Dr. Dirk Eisenbiegler
Hochschule Furtwangen

Aufgabe 11.1 - Methode mit Exception

Schreiben Sie eine statische Methode f , die einen Parameter x vom Typ `int-Array` hat und einen Rückgabewert vom Typ `int`. Der Rückgabewert ist die erste Zahl im Array $x[i]$, die größer ist als ihr Nachfolger $x[i+1]$.

Beispiele:

{ 1, 3, 5, 8, 6, 9, 12 } Rückgabewert: 8

{ 3, 8, 19, 14, 13, 200 } Rückgabewert: 19

{ 7, 4, 2, 1 } Rückgabewert: 7

Für den Fall, dass es kein solches $x[i]$ gibt, soll f eine Exception auslösen (throws Exception).

Beispiele:

{ 1, 2, 3 }

{ }

A) Programmieren Sie die Methode f .

B) Testen Sie die Methode mit den oben aufgeführten Beispielen.

Aufgabe 11.2 - Ausdrücke auswerten

In dieser Aufgabe sollen Strings ausgewertet werden, in denen einfache arithmetische Ausdrücke dargestellt werden. In den Strings seien lediglich die folgenden Zeichen zulässig:

0 1 2 3 4 5 6 7 8 9 + -

Beispiele:

"1+2"
"23"
"14+5-147"

Ziel dieser Aufgabe ist es, eine statische Methode mit dem Namen *auswerten* zu programmieren, die einen solchen String auswertet und den Wert in Form einer int-Zahl zurückgibt.

Bespiele:

Parameterwert	Rückgabewert
"1+2"	3
"23"	23
"14+5-147"	-128

Nicht jede Zeichenkette, die sich aus den oben aufgeführten Zeichen zusammensetzt, enthält einen gültigen arithmetischen Ausdruck. Fälle, in denen der String keinen zulässigen Ausdruck darstellt sind:

	Beispiele
Fall 1: Die Zeichenkette enthält nicht zulässige Zeichen.	"1+x" "23*"
Fall 2: In der Zeichenkette stehen unmittelbar hintereinander mehrere Operatoren (+,-).	"1-4++8" "+-8"
Fall 3: Die Zeichenkette endet mit einem Operator.	"3+42+" "8-"

Für den Fall, dass die Zeichenkette keinen gültigen Ausdruck darstellt, soll die Methode *auswerten* eine Exception mit dem Namen *AuswertungGescheitert* auslösen. In der Exception *AuswertungGescheitert* soll gespeichert werden, wieso

der String kein gültiger Ausdruck ist (Fall 1, Fall 2 oder Fall 3).

- A) Deklarieren Sie eine Klasse mit dem Namen *AuswertungGescheitert* als Sohnklasse der Klasse *Exception*. Die Klasse *AuswertungGescheitert* soll ein int-Attribut mit dem Namen *fall* haben. Das Attribut *fall* soll im folgenden dazu verwendet werden, einen der Werte 1, 2 oder 3 abzuspeichern - je nachdem ob Fall 1, Fall 2 oder Fall 3 eingetreten ist. Schreiben Sie zu *AuswertungGescheitert* einen Konstruktor, dem dieser int-Wert übergeben wird und der diesen im Attribut *fall* speichert. Schreiben Sie zu *AuswertungGescheitert* eine Methode *meldung*, die keine Parameter hat und die in Abhängigkeit des Attributs *fall* eine der drei Fehlermeldungen aus obiger Tabelle als String zurückgibt.
- B) Schreiben Sie die statische Methode *auswerten*. Die Methode hat einen String als Parameter, in dem der Ausdruck dargestellt wird. Der Rückgabewert ist vom Typ int. Gegebenenfalls wird in der Methode *auswerten* eine Exception *AuswertungGescheitert* ausgelöst (throws *AuswertungGescheitert*).

Tipps:

- ⇒ Mit der Klasse *StringTokenizer* können Sie den String in mehrere Teilstücke zerlegen, indem sie an den Operatoren (+,-) "zerschneiden". Verwenden Sie den Konstruktor mit drei Parametern: der erste Parameter ist dabei der String mit dem Ausdruck, der zweite Parameter ist "+-", der dritte Parameter ist *true*.
- ⇒ Die String-Stücke, in denen sich Zahlen befinden, können Sie mit dem Befehl *Integer.parseInt* in eine int-Zahl konvertieren. Hinweis: Kommt es bei der Übersetzung des Strings in eine int-Zahl zu einem Fehler, so wird eine *NumberFormatException* ausgelöst (eine Runtime-Exception). Dies tritt zum Beispiel dann ein, wenn in dem String Buchstaben statt Ziffern enthalten sind.
- C) Testen Sie die Methode *auswerten*. Fangen Sie dabei die Exception *AuswertungGescheitert* ab und geben Sie die die Meldung *meldung()* aus. Testen Sie *auswerten* unter anderem mit:
- "23+14-12-3"
"3+18"
"23414"
"-3-4"
"18-7-"
"14++8"

"13a+9"

"18+y"

Aufgabenblatt 12

Praktikum am 22.12.2015
Abgabe bis spätestens 11.1.2016

Programmierung
Wintersemester 2015/2016
Prof. Dr. Dirk Eisenbiegler
Hochschule Furtwangen

Aufgabe 12.1 - Fensterbestellung

Ein Fenster-Produzent bietet maßgeschneiderte Fenster in verschiedenen Varianten an. In dieser Aufgabe soll eine Klasse *FensterBestellung* geschrieben werden, wobei jede Instanz dieser Klasse eine Fensterbestellung repräsentieren soll.

Jede Fensterbestellung besteht aus einem Kundendatensatz sowie einem oder mehreren Fenstern. Der Kundendatensatz enthält: Vorname, Nachname, Straße, Hausnummer, PLZ, Ort.

Der Hersteller stellt ausschließlich rechteckige Fenster her. Breite und Höhe können millimetergenau vorgegeben werden. Es gibt vier Arten von Fenstern: Holzfenster, Kunststofffenster, Alufenster und Holz-Alu-Verbundfenster. Jedes Fenster besteht aus ein bis drei gleich großen Fensterflügeln, die nebeneinander angeordnet sind. Alle Flügel eines Fensters sollen immer die gleiche Verglasung erhalten. Zur Auswahl stehen eine Doppelverglasung und eine Dreifachverglasung. Jeder Flügel kann mit Festverglasung ausgeliefert werden, mit einem Drehmechanismus oder mit einem Dreh-Kipp-Mechanismus. Bei einem Dreh- oder Dreh-Kipp-Mechanismus kann die Flügelbefestigung links oder rechts sein.

Das Holz der Holzfenster und auch das Holz der Holz-Alu-Fenster kann entweder lasiert sein oder lackiert. Zur Auswahl stehen eine helle und eine dunkle Lasur. Bei den Holzlacken gibt es die Farben weiß, rot, gelb und blau. Die Alu-Teile der Alu-Fenster sowie der Holz-Alu-Fenster werden immer lackiert ausgeliefert. Die möglichen Farben sind auch hier weiß, rot gelb und blau. Kunststofffenster sind immer weiß, sie sind weder lasiert noch lackiert.

- A) Schreiben Sie eine Klasse mit dem Namen *FensterBestellung*, sodass in einer Instanz dieser Klasse alle für eine Bestellung relevanten Daten gespeichert werden können. Verwenden Sie an den dafür geeigneten Stellen Enumerations. Verwenden Sie die Generic-Klasse *Vector<A>* um mehrere Fensterflügel zu einer Liste von Fensterflügeln zusammenzufassen und um mehrere Fenster zu einer Liste von Fenstern zusammenzufassen. Deklarieren Sie zunächst als Zwischenschritt die folgenden Klassen/Enumerations: *Lasur*, *Farbe*, *Verglasung*, *Flügeltyp*, *FlügelBefestigung*, *Fensterflügel*, *Fensterart* und *Fenster*
- B) Schreiben Sie eine Objektmethode *toString()* zur Klasse *FensterBestellung*, die den Inhalt der Bestellung in der üblichen Form als Text ausgibt.

- C) Schreiben Sie zur Klasse *FensterBestellung* die Objektmethode *fehler()*, die bestimmt, ob die Bestellung gegen eine der unten aufgeführten technischen Randbedingungen verstößt. Wenn ja, dann soll der Rückgabewert ein String mit einer Beschreibung des Fehlers sein. Enthält die Fensterbestellung keinen Fehler, so soll der Rückgabewert *null* sein.
- ⇒ Da sich bei kleinen Bestellungen die Lieferung nicht lohnt, wird festgelegt, dass jede Bestellung Fenster mit einer Summe der Fensterflächen von mindestens 10m² enthalten muss.
 - ⇒ Jeder Flügel muss mindestens 800mm breit und 800mm hoch sein.
 - ⇒ Alu-Fenster werden nur mit Dreifachverglasung geliefert.
 - ⇒ Die Gesamtgröße der Fenster darf 2000mm Höhe und 5000mm Breite nicht überschreiten. Bei Holzfenstern darf die Breite sogar nicht größer als 4000mm sein.
 - ⇒ Bei einem Fenster mit mehr als einem Flügel darf die Flügelbefestigung des Flügels ganz links nicht auf der rechten Seite sein und die des Flügels ganz rechts nicht links sein.
- D) Schreiben Sie zur Klasse *Bestellung* eine Objektmethode mit dem Namen *preis()*, die als Rückgabewert den Gesamtpreis der Bestellung in Cent bestimmt. Schreiben Sie dazu zunächst in der Klasse *Fenster* eine Methode *preis()*, die den Preis eines einzelnen Fensters bestimmt. Der Preis eines einzelnen Fensters bestimmt sich wie folgt:
- ⇒ Der Grundpreis eines Kunststofffensters ergibt sich als 95€ pro m². Für Holzfenster ist ein Aufschlag von 10%, für Alufenster ein Aufschlag von 20% und für Holz-Alu-Fenster ein Aufschlag von 40% zu zahlen.
 - ⇒ Der Grundpreis versteht sich für ein Fenster mit einem Fensterflügel und dieser Flügel hat eine Festverglasung. Für jeden weiteren Flügel erhöht sich der Preis um 30€. Für jeden Flügel, der keine Festverglasung, sondern einen Drehmechanismus hat, ist ein Aufschlag von 40€ zu zahlen. Für einen Dreh-Kipp-Mechanismus beträgt der Aufschlag 55€.
 - ⇒ Soll ein Holzfenster oder ein Holz-Alufenster eine helle Lasur haben, so ist dafür ein Aufschlag zu zahlen, der sich am Umfang des Fensters orientiert. Der Aufpreis beträgt 4€ pro Meter Umfang.

Aufgabenblatt 13

Praktikum am 12.1.2016
Abgabe bis spätestens 22.1.2016

Programmierung
Wintersemester 2015/2016
Prof. Dr. Dirk Eisenbiegler
Hochschule Furtwangen

Aufgabe 13.1 - Wechselgeld

Mit einem Verkaufsautomat sollen verschiedene Produkte verkauft werden. Nachdem der Kunde mehrere Produkte ausgewählt hat, berechnet der Automat die Summe der Einzelpreise. Dann kommt es zur Bezahlung: Der Benutzer wirft eine bestimmte Anzahl von Münzen ein. Der Automat entscheidet dann, ob genug eingezahlt wurde oder nicht. Im Erfolgsfall händigt er die Produkte aus und gibt das Wechselgeld zurück.

Der Automat akzeptiert die folgenden Münzen: 1 Cent, 2 Cent, 5 Cent, 10 Cent, 20 Cent, 50 Cent, 1 Euro und 2 Euro. Intern verfügt der Automat über einen Speicher für solche Münzen. Münzen dieser Art können bei der Bezahlung der Produkte vom Automaten entgegengenommen und als Wechselgeld an den Kunden zurückgegeben werden.

- A) Schreiben Sie eine Enumeration mit dem Namen Münzart. Jede Konstante dieser Enumeration soll für eine der oben genannten Münzen. Jeder Konstanten soll ein Wert in Cent zugewiesen werden. Realisieren Sie dies mit einem geeigneten Objektattribut.
- B) Schreiben Sie eine Klasse mit dem Namen Münzen. Jede Instanz der Klasse Münzen steht für mehrere Münzen, die unterschiedliche Münzarten haben können. Eine Instanz der Klasse Münzen speichert in einem Objektattribut zu jeder Münzart eine Anzahl. Verwenden Sie dazu die Generic-Klasse *HashTable<A,B>*.

Erläuterung: Eine Instanz der Klasse *Münzen* kann dazu verwendet werden, den Bezahlvorgang des Kunden zu beschreiben. Eine solche Instanz kann jedoch auch dazu verwendet werden, den Stand des im Automaten eingebauten Münzspeichers zu beschreiben. Eine solche Instanz kann schließlich auch dazu verwendet werden, die Wechselgeld-Zahlung zu beschreiben.

- C) Schreiben Sie zur Klasse *Münzen* einen Default-Konstruktor, der eine leer Münzenmenge erzeugt.
- D) Schreiben Sie zur Klasse *Münzen* eine Objektmethode *setMünze*, mit der man die Anzahl einer bestimmten Münzart auf einen bestimmten Wert setzen kann. Münzart und Anzahl sollen Parameter der Methode sein. Die Methode hat keinen Rückgabewert.

- E) Schreiben Sie zur Klasse *Münzen* eine Objektmethode *getMünze*, mit der man zu einer Münzart deren Anzahl abfragen kann. Die Methode soll einen Parameter vom Typ *Münzart* und einen Rückgabewert vom Typ *int* haben, der die Anzahl der Münzen dieses Typs zurückgibt.
- F) Schreiben Sie zur Klasse *Münzen* eine *toString()*-Methode, die den Inhalt eines Münzen-Objekts gemäß dem nachfolgenden Schema in einen *String* verwandelt.
- 2 Euro - 1 Stück
 - 50 Cent - 3 Stück
 - 5 Cent - 2 Stück
 - 1 Cent - 1 Stück
- G) Schreiben Sie zur Klasse *Münzen* eine Objektmethode mit dem Namen *wert*. Diese Methode soll keinen Parameter und einen Rückgabewert vom Typ *int* haben. Der Rückgabewert soll dem Wert der im Objekt enthaltenen Münzen entsprechen.
- H) Schreiben Sie zur Klasse *Münzen* einen Konstruktor mit einem Parameter vom Typ *int*. Der Wert des Parameters steht für einen Cent-Betrag. Der Konstruktor soll ein Münzen-Objekt erzeugen, das Münzen mit diesem Betrag enthält. Dabei sollen möglichst wenige Münzen verwendet werden - Prinzip: „Möglichst große Münzen verwenden“. Ein Betrag von 60 Cent soll beispielsweise nicht mit drei 20 Cent-Münzen repräsentiert werden, sondern mit einer 50 Cent-Münze und einer 10 Cent-Münze.
- I) Schreiben Sie zur Klasse *Münzen* einen Konstruktor mit zwei Parametern: einen vom Typ *int* und einen vom Typ *Münzen*. Der Wert vom Typ *int* repräsentiert einen Cent-Wert. Das zu erzeugende Objekt soll Münzen mit diesem Wert darstellen. Ziel ist es wieder, mit möglichst wenig Münzen auszukommen. Der zweite Parameter enthält alle Münzen, die aktuell im Speicher des Automaten enthalten sind. Nur diese Münzen sollen bei der Erzeugung des Objekts verwendet werden. Der Aufruf des Konstruktors soll scheitern, wenn im Speicher nicht genug Münzen enthalten sind. In diesem Fall soll im Konstruktor eine *RuntimeException* geworfen werden.
- J) Schreiben Sie zur Klasse *Münzen* eine Objektmethode mit dem Namen *normieren*. Diese Methode soll keinen Parameter und keinen Rückgabewert haben. Sie soll den Inhalt des Münzen-Objekts so verändern, dass danach die Anzahl der Münzen minimal ist, ohne dass dabei der Wert verändert wird.

Aufgabenblatt 14

*Zusätzliche Aufgaben
keine Abgabe*

*Programmierung
Wintersemester 2015/2016
Prof. Dr. Dirk Eisenbiegler
Hochschule Furtwangen*

Aufgabe 14.1 - Zeichen vertauscht ausgeben

- A) Schreiben Sie eine Methode, die die Zeichen eines String vertauscht ausgibt:
1. Zeichen, letztes Zeichen, 2. Zeichen vorletztes Zeichen, 3. Zeichen, ...

Beispiel

Parameter: Kokosnusspalme

Ausgabe: Keomkloaspnsus

- B) Schreiben Sie eine Methode, das die Zeichen eines String in folgender Reihenfolge ausgibt:

1. Zeichen, 3. Zeichen, 2. Zeichen, 4. Zeichen, 3. Zeichen, ...

Prinzip: zwei Schritte vor, einen Schritt zurück - bis der letzte Buchstabe erreicht wird

Beispiel

Parameter: Kokosnusspalme

Ausgabe: Kkooksonsunsusspsaplamle

Aufgabe 14.2 - Bitoperationen

Beispiel: 0xFF4A3B87
Rot-Anteil: 4A, Grün-Anteil: 3B, Blau-Anteil: 87

In dieser Aufgabe wird die Klasse Picture verwendet, die bereits in Aufgabe 6.2 vorgestellt wurde.

In jeder der Teilaufgaben soll eine andere Operation an den Farben vorgenommen werden. Ändern Sie jeweils die Farben an den Punkten, die im Kreis mit dem Mittelpunkt (150,150) und dem Radius 100 liegen. Alle anderen Punkte sollen unverändert bleiben.

Jeder Punkt wird durch einen int-Wert dargestellt. Die ersten 8 Bit sind ungenutzt und müssen immer auf 1 gesetzt sein. In den nächsten 8 Bit befindet sich die Farbintensität für die Farbe Rot, in den nächsten 8 Bit die der Farbe Grün und in den letzten 8 Bit die Farbintensität für die Farbe Blau.

1	1	1	1	1	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
255								32								7								128							
ungenutzt								Rot-Anteil								Grün-Anteil								Blau-Anteil							

- A) Setzen Sie in dem Kreis den Rot-Anteil auf konstant 128.
- B) Setzen Sie in dem Kreis für jede Farbe den Farbanteil von x auf 255-x.
- C) Vertauschen Sie die Farbanteile: der Rot-Anteile wird zum Grün-Anteil, der Grün-Anteil zum Blau-Anteil und der Blau-Anteil zum Rot-Anteil.

Hinweise:

- x Durch die Operationen (a & b) werden zwei int-Zahlen a und b durch bitweises UND miteinander verknüpft. Das Ergebnis ist vom Typ int.
- x Durch die Operationen (a | b) werden zwei int-Zahlen a und b durch bitweises ODER miteinander verknüpft. Das Ergebnis ist vom Typ int.
- x Mit (a >> b) wird eine int-Zahl berechnet, die entsteht, wenn man die Bits im int-Wert a um b Stellen nach rechts verschiebt.
- x Mit (a << b) wird eine int-Zahl berechnet, die entsteht, wenn man die Bits im int-Wert a um b Stellen nach links verschiebt.
 - x Um int-Zahlen im Hexadezimalsystem statt im Dezimalsystem anzugeben, muss lediglich 0x vorangestellt werden.

Aufgabe 14.3 - Nullstellensuche durch Newton-Iteration

Schreiben Sie drei Methoden mit den Namen *f*, *g* und *nullstelle*.

Die Methode *f* soll die Funktion $f(x) = e^x + x^2 - 4$ und die Methode *g* die Funktion $g(x) = e^x + 2 \cdot x$ realisieren. Die Funktion *g* ist die erste Ableitung von *f*. Die Parameter und die Rückgabewerte der Funktionen *f* und *g* sollen vom Typ *double* sein.

Die Methode *nullstelle* soll per Newton-Iteration nach einer Nullstelle von *f* suchen. Parameter von *nullstelle* sind *x* und *z*. Der Wert *x* ist der Startwert der Iteration, der Wert *z* ist die Genauigkeit. Das Iterationsverfahren soll abbrechen, sobald ein Wert *m* gefunden wird, sodass $|f(m)| < z$ gilt.

Newton-Iteration

Von einem Näherungswert *x* zum nächsten Näherungswert *x'* kommt man wie folgt:

$$x' = x - \frac{f(x)}{f'(x)}$$

Hinweise:

x Die Java-Funktion `Math.exp(x)` berechnet e^x .

x Die Java-Funktion `Math.abs(x)` berechnet $|x|$.

Aufgabe 14.4 - Zeichensatz ausgeben

In dieser Aufgabe soll der Zeichensatz in Tabellenform ausgegeben werden.

Schreiben Sie eine Methode, die für jede Zahl *x* aus einem Zahlenbereich zwischen *a* und *b* jeweils eine Zeile ausgibt. In jeder Zeile sollen nacheinander die folgenden Daten ausgegeben werden:

- ⇒ die Zahl *x* selbst
- ⇒ der Buchstabe, dessen Code *x* ist
- ⇒ *x* dargestellt als Hexadezimalzahl
- ⇒ *x* dargestellt als Binärzahl

a und *b* sind Parameter der zu implementierenden Methode (beide vom Typ *int*).

Hinweise:

x mit dem Ausdruck

```
(char)x
```

wird der Buchstabe bestimmt, dessen Code *x* ist.

x der Ausdruck

```
Integer.toString(x,b)
```

konvertiert die *int*-Zahl *x* in einen *String*, in dem der Wert der Zahl im Zahlensystem mit der Basis *b* dargestellt wird.

Umwandlung in Binärdarstellung: *b*=2.

Umwandlung in Hexadezimaldarstellung: *b*=16.

Aufgabe 14.5 - Quadrate

Schreiben Sie eine Methode *quadrat*, die Quadrate auf den Bildschirm zeichnet. Die Kanten der Quadrate bestehen aus "*" -Zeichen. Die Quadrate sollen mit "-" -Zeichen ausgefüllt werden. Die Methode *quadrat* soll beliebig große Quadrate zeichnen können. Die Kantenlänge *k* ist Parameter der Methode *quadrat*. Die nachfolgenden Beispiele zeigen Quadrate mit k=4, k=7 und k=1.

```
* * * *
* _ _ *
* _ _ *
* * * *
```

```
* * * * *
* _ _ _ _ *
* _ _ _ _ *
* _ _ _ _ *
* _ _ _ _ *
* _ _ _ _ *
* * * * *
```

```
*
```

Aufgabe 14.6 - Primzahlen

- A) Schreiben Sie eine Methode *teilbar*, die zu zwei int-Zahlen x und y bestimmt, ob x durch y teilbar ist. Der Rückgabewert ist vom Typ boolean.
Hinweis:
Der Java-Ausdruck *a%b* bestimmt den Rest einer Division von a und b.
- B) Schreiben Sie eine Methode *prim*, die bestimmt ob eine int-Zahl eine Primzahl ist. Der Parameter der Zahl *prim* hat den Typ int, der Rückgabewert ist vom Typ boolean.
Primzahlen sind alle Zahlen, die nur durch 1 und durch sich selbst teilbar sind. Die Zahl 1 ist keine Primzahl.
Beispiele für Primzahlen: 2, 3, 5, 7, 11, 13, 17, 19, ...
- C) Jede Zahl lässt sich in eindeutiger Weise in ein Produkt von Primzahlen zerlegen. Beispiel: $168=2\cdot2\cdot2\cdot3\cdot7$
Schreiben Sie eine Methode *primfaktoren*, die zu einer int-Zahl die Primfaktorzerlegung ausgibt.

Aufgabe 14.7 - Geheimschrift

Schreiben Sie zwei Methoden *encrypt* und *decrypt*, mit der ein String *s* verschlüsselt bzw. entschlüsselt werden soll.

Der String *s* soll Zeichen für Zeichen durch Vertauschung verschlüsselt werden. Wie vertauscht wird, wird durch einen Verschlüsselungsstring *x* beschrieben:

- ⇒ Ist das zu verschlüsselnde Zeichen in *x* enthalten, so soll es durch das nachfolgende Zeichen in *x* ersetzt werden.
- ⇒ Ist das zu verschlüsselnde Zeichen das letzte Zeichen in *x*, so wird es durch das erste Zeichen in *x* ersetzt.
- ⇒ Ist das Zeichen nicht in *x* enthalten, so bleibt es unverändert.

Beispiel:

s="Hallo Herr Müller!"

x="aKurHül!"

Verschlüsselter String: "üK!!o üeHH M!!eHa"

Die Entschlüsselungsfunktion vertauscht genau entgegengesetzt.

Die Methode *encrypt* hat die beiden String-Parameter *s* und *x* und einen Rückgabewert vom Typ String. Die Methode *decrypt* hat ebenfalls zwei String-Parameter *s* und *x* und einen Rückgabewert vom Typ String.

Aufgabe 14.8 - Skalarprodukt

Schreiben Sie eine Methode, die das Skalarprodukt zweier Vektoren berechnet.

Die Methode hat zwei Parameter *a* und *b*. Beide Parameter sind int-Arrays. Die int-Arrays stehen für die zu multiplizierenden Vektoren. Es soll angenommen werden, dass beide Arrays die gleiche Länge haben. Der Rückgabewert der Methode sei vom Typ int und sein Wert sei das Skalarprodukt der Vektoren *a* und *b*.

Hinweis:

Unter dem (kanonischen) Skalarprodukt zweier Vektoren $\vec{a}=(a_0, a_1, \dots, a_{n-1})$ und $\vec{b}=(b_0, b_1, \dots, b_{n-1})$ versteht man:

$$\vec{a} \cdot \vec{b} = a_0 \cdot b_0 + a_1 \cdot b_1 + \dots + a_{n-1} \cdot b_{n-1}$$

Aufgabe 14.9 - Sortieren

Schreiben Sie eine Methode, die einen Array von double-Werten sortiert.

Hinweis:

- x Die Methode hat einen Parameter vom Typ double-Array (eine Objekt-Referenz!). Der Rückgabewert der Methode ist void. Innerhalb der Methode sollen die Variablenwerte an den einzelnen Positionen so vertauscht werden, dass schließlich alle Werte in aufsteigender Reihenfolge sortiert sind.
- x Einfacher Algorithmus (als Vorschlag):
 - ⇒ Das Minimum aller Werte suchen, diesen Wert mit dem Wert an der ersten Position vertauschen.
 - ⇒ Ab der zweiten Position nach dem minimalen Wert suchen, diesen Wert mit dem Wert an der zweiten Position vertauschen.
 - ⇒ ...

Aufgabe 14.10 - Morsen

In dieser Aufgabe soll der Roboter Morsezeichen erzeugen.

Beim Morsen gibt es zwei verschiedene Signalwerte: Strich und Punkt. Ein Strich dauert 0,3s, ein Punkt dauert 0,1s. Der Roboter soll Striche und Punkte durch das gleichzeitige Einschalten der beiden Lämpchen und durch einen Ton signalisieren – jeweils mit der entsprechenden Länge. Der Ton soll bei einem Strich die Frequenz 440Hz und bei einem Punkt die Frequenz 659Hz haben. Zwischen den Zeichen soll jeweils eine Pause von 0,1s sein, zwischen zwei Buchstaben eine Pause von 0,3s.

Morse-Alphabet:

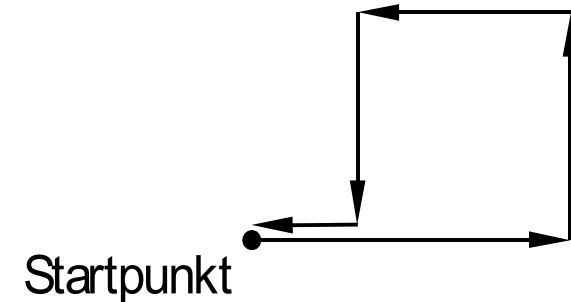
A	.-	B	-...	C	-.-.
D	-..	E	.	F	..-.
G	--.	H	I	..
J	.-.-	K	-.-	L	.-..
M	--	N	-.	O	---
P	.-.-	Q	--.-	R	.-.
S	...	T	-	U	..-
V	...-	W	.-.-	X	-.-.-
Y	-.-.-	Z	--..		

Der Roboter soll die folgende Buchstabenfolge morsen:

ROBOTERGESCHWAETZ

Aufgabe 14.11 - Roboter-Fahrt

In dieser Aufgabe soll ein Programm geschrieben werden, durch das der Roboter die folgende Figur abfährt.



Technische Details:

- x Nach jeder Bewegung – Vorwärtsbewegung oder Drehung – soll der Roboter eine Pause von 0,5s einlegen.
- x Verwenden Sie zur Fortbewegung die Geschwindigkeit $200 \frac{mm}{s}$.
- x An den Eckpunkten soll sich der Roboter im Stand um 90° drehen: ein Rad dreht sich vorwärts, das andere rückwärts.

Aufgabe 14.12 - Roboter-Tanz

In dieser Aufgabe soll ein Programm entwickelt werden, das den Roboter „tanzen“ lässt.

Der Grundrythmus ist ein 4/4-Takt. Jeder Takt besteht also aus vier Schlägen mit gleichem zeitlichen Abstand. Der Roboter kennt folgende Schritte:

- x Der Grundschrift besteht aus einer Vorwärts- und einer Rückwärtsbewegung, die jeweils einen Takt lang dauert. Dabei bewegt sich der Roboter jeweils nur in der ersten Hälfte des Taktes und steht in der zweiten Hälfte des Taktes.
- x Die halbe Drehung dauert einen Takt lang. In der ersten Hälfte des Taktes dreht sich der Roboter auf der Stelle, in der zweiten Hälfte steht der Roboter still. Die halbe Drehung kann sowohl nach links als auch nach rechts ausgeführt werden.
- x Die ganze Drehung dauert zwei Takte lang. In den ersten 1,5 Takten dreht sich der Roboter nach rechts, dann bleibt er einen halben Takt lang stehen.

Der Tanz beginnt mit einem Takt Einzählen, bei dem sich der Roboter nicht bewegt. Anschließend folgen drei Grundschriffe zu je 2 Takten, dann eine halbe Drehung nach links, dann eine halbe Drehung nach rechts. Dies wird wiederholt: drei Grundschriffe, halbe Drehung links, halbe Drehung rechts. Es folgen wieder drei Grundschriffe und zum Abschluss eine ganze Drehung.

Während der Drehbewegungen – halben und ganzen Drehungen – schaltet der Roboter das entsprechende Lämpchen an: rechtes Lämpchen bei Rechtsdrehung, linkes Lämpchen bei Linksdrehung. Außerdem schaltet er beim Grundschrift das Lämpchen im dritten und vierten Schlag für jeweils 0,1s an. Beim Einzählen wird das Lämpchen bei jedem Schlag für jeweils 0,1s eingeschaltet.

Während des Tanzes erzeugt der Roboter zwei verschiedene Töne: das A (440 Hz) und das E (659 Hz). Die Note A spielt er immer dann, wenn eine Bewegung erfolgt. Der Ton dauert genau so lange wie die Bewegung. Die Note B spielt der Roboter immer dann, wenn er ein 0,1s langes Lichtsignal gibt. Der Ton dauert genau so lange wie das Lichtsignal.

Nehmen Sie als Geschwindigkeit für alle Bewegungen $200 \frac{mm}{s}$. Nehmen Sie


zunächst an, dass ein Takt 1,2s dauert. Modifizieren Sie anschließend Ihr Programm so, dass ein Takt 1s dauert.

Roboter-Kurzanleitung

Vorbereitung

- x Stellen Sie sicher, dass der Roboter mit zwei Leuchtdioden (rote Lämpchen) und einem Piezo-Summer (kleiner runder Lautsprecher) ausgestattet ist.
 - ⇒ Die beiden Leuchtdioden müssen mit den Steuerausgängen P9 (rechte Leuchtdiode) und P10 (linke Leuchtdiode) verbunden sein.
 - ⇒ Der Piezo-Summer muss mit dem Steuerausgang P2 verbunden sein.
- x Kopieren Sie, so noch nicht geschehen, das Verzeichnis \\server2000\studenten\Eisenbiegler_public\prog nach X:\prog
- x Starten Sie die Entwicklungsumgebung für den Roboter: Javalin Stamp IDE
- x Gehen Sie im Menü zum Eintrag *Project* und dort zu *Global Options*. Das Feld CLASSPATH ist bereits mit einer Zeichenkette gefüllt. Fügen Sie an das Ende der Zeichenkette die folgende Zeichenkette an:
;X:\prog\robot

Funktionstest

- x Laden Sie die Datei X:\prog\robot\FunctionTest.java (Menüpunkt: File – Open)
- x Verbinden Sie über das serielle Kabel den Roboter mit dem Computer.
- x Drücken Sie den Button  **Program**, den Sie unterhalb der Menüleiste finden.
 - ⇒ Das Programm wird übersetzt, dann auf den Roboter übertragen und schließlich auf dem Roboter gestartet. Ist der Roboter richtig installiert, so sollte er zunächst vier Töne spielen, dann mit dem linken Lämpchen blinken, dann mit dem rechten, dann ein wenig nach vorne fahren, warten und schließlich ein wenig rückwärts fahren.

Ein neues Roboter-Programm erstellen

Erstellen Sie eine Neue Datei im Verzeichnis X:\prog\robot. In der Entwicklungsumgebung: File – New, dann File – Save as. Die Endung .java wird beim speichern automatisch an den Dateinamen.

Der Inhalt der Datei soll dann folgenden Aufbau haben:

```
public class X extends Robot {  
    public static void main() {  
        // Fügen Sie die Anweisungen hier ein!  
    }  
}
```

Achten Sie darauf, dass in der ersten Zeile an der Stelle des X Ihr Dateiname (ohne .java-Endung) steht.

Roboter-Befehle

setSpeed(x,y)	<p>Steuerung der Räder</p> <p>x und y sind ganzzahlige Zahlenwerte, durch die die Geschwindigkeit des linken (x) und des rechten Rades (y) in $\frac{mm}{s}$ angegeben wird.</p> <p>Mit einem Wert von 0 wird das betreffende Rad angehalten.</p> <p>Positive Werte stehen für eine Vorwärtsbewegung, negative Werte für eine Rückwärtsbewegung. Der Betrag der Werte muss im Bereich zwischen 75 und 400 liegen.</p>
setLights(x,y)	<p>Steuerung der Lämpchen.</p> <p>x steht für das linke Lämpchen, y für das rechte Lämpchen. x und y dürfen die Werte 0 und 1 annehmen: 0=aus, 1=ein.</p>
setSound(x)	<p>Steuerung des Lautsprechers</p> <p>Der ganzzahlige Wert x gibt die Frequenz in Hertz an. Mit x=0 wird der Lautsprecher ausgeschaltet.</p>
wait(x)	<p>Warten</p> <p>Der Roboter macht x Millisekunden lang nichts.</p>

Beispielprogramm

```
public class FunctionTest extends Robot {

    public static void blink(int x, int y, int n) {
        for (int i=0; i<n; i++) {
            setLights(x,y);
            wait(200);
            setLights(0,0);
            wait(200);
        }
    }

    public static void main() {
        setSound(440);
        wait(500);
        setSound(554);
        wait(500);
        setSound(659);
        wait(500);
        setSound(880);
        wait(500);
        setSound(0);
        blink(1,0,5);
        blink(1,0,5);
        setSpeed(100,100);
        wait(5000);
        setSpeed(0,0);
        wait(2000);
        setSpeed(-100,-100);
        wait(5000);
        setSpeed(0,0);
    }
}
```

Aufgabe 14.13 - Bank

```
public class Konto {

    public int kontonummer;
    public String vorname;
    public String nachname;
    public int guthaben;
    public Vector buchungen;

    public Konto(String vorname, String nachname, int kontonummer) {
        this.vorname = vorname;
        this.nachname = nachname;
        this.kontonummer = kontonummer;
        this.guthaben = 0;
        buchungen = new Vector();
    }

    public void buchungHinzufügen(Buchung buchung) throws BankException{
        if (buchung.empfänger!=null && buchung.empfänger.equals(this))
            guthaben = guthaben + buchung.betrag;
        if (buchung.auftraggeber!=null &&
            buchung.auftraggeber.equals(this)) {
            if (guthaben<buchung.betrag)
                throw new BankException(
                    "Buchungsbetrag zu hoch für Kontostand",this);
            guthaben = guthaben - buchung.betrag;
        }
        buchungen.add(buchung);
    }

    public boolean equals(Object x) {
        try {
            Konto konto = (Konto) x;
            return
                vorname.equals(konto.vorname) &&
                nachname.equals(konto.nachname) &&
                (kontonummer == konto.kontonummer);
        } catch (ClassCastException e) {
            return false;
        }
    }

    public String toString() {
        StringBuffer sb = new StringBuffer();
        sb.append("Eigentümer:  " + vorname + " " + nachname + "\n" );
        sb.append("Kontonummer: " + kontonummer + "\n" );
        sb.append("Guthaben:    " + guthaben + "\n");
        sb.append("Buchungen:\n");
        for (int i=0; i<buchungen.size(); i++)
            sb.append(((Buchung)buchungen.elementAt(i)).toString() + "\n");
        return sb.toString();
    }

    public String eigentümer() {
        return vorname + " " + nachname + " (" + kontonummer + ")";
    }
}
```



```

public class Buchung {

    public Konto auftraggeber;
    public Konto empfänger;
    public int betrag;

    public Buchung(Konto auftraggeber, Konto empfänger, int betrag) {
        this.auftraggeber = auftraggeber;
        this.empfänger = empfänger;
        this.betrag = betrag;
    }

    public String toString() {
        String b = betrag + "€";
        if ((auftraggeber!=null) && (empfänger==null))
            return "Auszahlung an "+auftraggeber.eigentümer()+" "+b;
        if ((auftraggeber==null) && (empfänger!=null))
            return "Einzahlung von " + empfänger.eigentümer() + " " + b;
        return "Überweisung von " + auftraggeber.eigentümer() + " an " +
            empfänger.eigentümer() + " " + b;
    }
}

```

```

public class BankException extends Exception {

    private String meldung;
    private Konto konto;

    public BankException(String meldung, Konto konto) {
        this.meldung = meldung;
        this.konto = konto;
    }

    public String toString() {
        return meldung + "\nKonto:" + konto.eigentümer();
    }
}

```

```

public class Bank {

    private Vector konten_nach_nummer = new Vector();
    private Hashtable konten_nach_name = new Hashtable();

    public int eröffneKonto(String vorname, String nachname)
        throws BankException {
        int kontonummer = konten_nach_nummer.size();
        Konto konto = new Konto(vorname, nachname, kontonummer);
        konten_nach_nummer.insertElementAt(konto,kontonummer);
        String name = vorname + " " + nachname;
        if (konten_nach_name.containsKey(name))
            throw new BankException("Kunde existiert bereits", konto);
        konten_nach_name.put(name, konto);
        return kontonummer;
    }

    public Konto konto(int kontonummer) throws BankException {
        if (kontonummer<0 || kontonummer >= konten_nach_nummer.size())
            throw new BankException("Kontonummer existiert nicht", null);
        return (Konto)konten_nach_nummer.elementAt(kontonummer);
    }

    public Konto konto(String vorname, String nachname)
        throws BankException {
        String name = vorname + " " + nachname;
        if (!konten_nach_name.containsKey(name))
            throw new BankException("Kunde existiert nicht", null);
        return (Konto)konten_nach_name.get(vorname + " " + nachname);
    }

    public void auszahlen(int auftraggeber_nummer, int betrag)
        throws BankException {
        Konto auftraggeber = konto(auftraggeber_nummer);
        Buchung buchung = new Buchung(auftraggeber,null,betrag);
        auftraggeber.buchungHinzufügen(buchung);
    }

    public void einzahlen(int empfänger_nummer, int betrag)
        throws BankException {
        Konto empfänger = konto(empfänger_nummer);
        Buchung buchung = new Buchung(null,empfänger,betrag);
        empfänger.buchungHinzufügen(buchung);
    }

    public void überweisen(int auftraggeber_nummer,
        int empfänger_nummer, int betrag) throws BankException {
        Konto empfänger = konto(empfänger_nummer);
        Konto auftraggeber = konto(auftraggeber_nummer);
        Buchung buchung = new Buchung(auftraggeber,empfänger,betrag);
        auftraggeber.buchungHinzufügen(buchung);
        empfänger.buchungHinzufügen(buchung);
    }

    public String toString() {
        StringBuffer sb = new StringBuffer();
        Enumeration enum = konten_nach_nummer.elements();
        while (enum.hasMoreElements())
            sb.append(((Konto)enum.nextElement()).toString() + "\n");
        return sb.toString();
    }
}

```



```

public class Tester {
    public static void main(String[] args) {
        Bank bank = new Bank();
        try {
            bank.eröffneKonto("Helmut", "Müller");
            bank.eröffneKonto("Katrin", "Neumann");
            bank.eröffneKonto("Olaf", "Maier");
            bank.einzahlen(1,1500);
            bank.auszahlen(1,300);
            bank.überweisen(1,0,500);
            bank.überweisen(0,2,300);
            bank.auszahlen(1,1000);
            bank.auszahlen(1,1000);
        } catch (BankException e) {
            System.out.println(e+"\n\n");
        }
        System.out.println(bank);
    }
}

```

- x Die Klasse Bank enthält zwei Methoden mit dem gleichen Namen nämlich konto. Ist das zulässig?
- x In der Klasse Bank werden die Konten in zwei verschiedenen Datenstrukturen aufbewahrt (Vector, HashTable). Existiert jedes Kontoobjekt doppelt?
- x In der Methode eröffneKonto werden die Methoden konten_nach_nummer.add und konten_nach_name.put aufgerufen. Beide Methoden erwarten als Parameter Objekte der Klasse Object - übergeben werden jedoch Konto-Objekte und Strings. Ist das zulässig?
- x In den beiden konto-Methoden der Klasse Bank werden jeweils Konten nachgeschlagen. Welche Bedeutung hat jeweils der Type-Cast? Kann es zu einer ClassCastException kommen?
- x In der Methode equals von Konto werden Vorname, Nachname und Kontonummer verglichen. Warum geschieht dies einmal mit .equals und einmal mit dem Operator == ?
- x Welche Bedeutung hat in der Methode equals von Konto die ClassCastException? Wann wird der Rückgabewert true und wann false? Was wäre anders, wenn man die ClassCastException nicht abfangen würde?
- x Wo kommt es in Tester.main zu einer BankException? Welche der Anweisungen im try-Block werden ausgeführt?

Aufgabenblatt 15

Musterlösungen

Musterlösung zu 2.2

Vorbemerkung zu den Musterlösungen: Es gibt im allgemeinen sehr viele Möglichkeiten, ein Programm richtig zu implementieren. Andere, ebenfalls richtige Lösungen können von den Musterlösungen stark abweichen.

Idee

Man durchläuft alle Buchstaben von links bis zur Mitte und vergleicht die Buchstaben mit dem jeweils gegenüberliegenden Buchstaben. In Java hat der erste Buchstabe eines Strings die Position 0, der letzte die Position $n-1$, wobei n die Länge des Strings sei.

Es müssen alle Buchstaben von der Position 0 bis zur Position $(n/2)-1$ durchlaufen werden.

Erläuterung: Ist n gerade, so liegt die Mitte zwischen den Buchstaben an den Positionen $(n/2)-1$ und $(n/2)$. Betrachten Sie dazu das Beispiel ANNA. Ist n ungerade, so liegt die Mitte bei der Position $n/2$ (Anmerkung: bei der Division mit Ganzzahlen wird abgerundet). Betrachten Sie dazu das Beispiel maoam. Die Mittelposition liegt hier bei der Position $(n/2)=5/2=2$ (Buchstabe o). Es müssen also auch hier alle Positionen von 0 bis $(n/2)-1$ durchlaufen werden.

Gegenüberliegend heißt hier: der Buchstabe an der Position i ist zu vergleichen mit dem Buchstaben an der Position $n-1-i$.

Beispiel: ANNA

Länge des Strings $n=4$. Die Variable i durchläuft die Werte 0 bis $(n/2)-1=1$. Der Buchstabe an der Position $i=0$ wird mit dem Buchstaben an der Position $n-1-i=3$ verglichen. Der Buchstabe an der Position $i=1$ wird mit dem Buchstaben an der Position $n-1-i=2$ verglichen.

A	N	N	A
0	1	2	3

Beispiel: maoam

Länge des Strings $n=5$. Die Variable i durchläuft die Werte 0 bis $(n/2)-1=1$. Der Buchstabe an der Position $i=0$ wird mit dem Buchstaben an der Position $n-1-i=4$ verglichen. Der Buchstabe an der Position $i=1$ wird mit dem Buchstaben an der Position $n-1-i=3$ verglichen.

m	a	o	a	m
0	1	2	3	4

Implementierung

```
public class Palindrom {

    public static void main(String[] args) {
        String eingabe = args[0];
        if (palindrom(eingabe))
            System.out.println(
                "Der String " + eingabe + " ist ein Palindrom.");
        else
            System.out.println(
                "Der String " + eingabe + " ist kein Palindrom.");
    }

    public static boolean palindrom (String wort) {
        boolean abweichungGefunden = false;
        int n = wort.length();
        // Die Variable i durchläuft alle Positionen des Strings von
        // links (Position 0) bis zur Mitte (Position n/2-1).
        for (int i=0; i<=(n/2)-1; i++) {
            // Für jede Position i wird der Buchstabe an der Position i
            // mit dem Buchstaben der gegenüberliegenden Position n-1-i
            // verglichen.
            if (wort.charAt(i) != wort.charAt(n-1-i)) {
                abweichungGefunden = true;
            }
        }
        if (abweichungGefunden) {
            // wurde eine Abweichung gefunden, so ist der String kein
            // Palindrom
            return false;
        } else {
            // wurde keine Abweichung gefunden, so ist der String ein
            // Palindrom
            return true;
        }
    }
}
```

Musterlösung zu 3.1

Implementierung

```
public class Aufgabe_4_1 {

    public static int summe (int [] a) {
        int s = 0;
        for (int i=0; i<a.length; i++)
            s = s + a[i];
        return s;
    }

    public static int mittel (int [] a) {
        return summe(a) / a.length;
    }

}
```

Test:

```
public class Tester {
    public static void main(String[] args) {
        int werte [] = {1,3,4,3,8,2};
        System.out.println(Aufgabe_4_1.summe(werte));
        System.out.println(Aufgabe_4_1.mittel(werte));
    }
}
```

Musterlösung zu 4.1

Idee

Man schreibt zunächst eine Methode *vork*, die die Anzahl der Vorkommen eines Buchstabens in einem String berechnet. Darauf aufbauend implementiert man dann die Methode *vorkommen*. Die Methode *vorkommen* durchläuft alle Buchstaben des Strings und bestimmt dabei mit Hilfe der Methode *vork* die Anzahl der Vorkommen dieses Buchstabens im Gesamt-String.

Implementierung

```
public class Vorkommen {  
  
    public static int vork (String wort, char buchstabe) {  
        int anzahl=0;  
        for (int i=0; i<wort.length(); i++)  
            if (wort.charAt(i) == buchstabe)  
                anzahl++;  
        return anzahl;  
    }  
  
    public static void vorkommen (String wort) {  
        for (int i=0; i<wort.length(); i++)  
            System.out.println(wort.charAt(i) + " " + vork(wort,  
wort.charAt(i)));  
    }  
}
```

Musterlösung zu 4.2

Idee

Man durchläuft alle Positionen des Strings x und prüft, ob von dieser Position an der String y steht.

Seien nx und ny die Längen der Strings x und y. Man muss nicht alle Positionen 0..(nx-1) durchlaufen. Nach der Position nx-ny hat der String y keinen Platz mehr. Es reicht also, die Positionen 0..(nx-ny) zu durchlaufen.

Es wird im String x nach einer Position i gesucht, sodass für alle j, mit j=0,1,...ny-1 gilt, dass der i+j-te Buchstabe von x gleich dem j-ten Buchstaben von y ist.

Implementierung

```
public class Substring {  
  
    public static int substring (String x, String y) {  
        int nx = x.length();  
        int ny = y.length();  
        for (int i=0; i<=nx-ny; i++) {  
            boolean abweichungGefunden = false;  
            for (int j=0; j<ny; j++)  
                if (x.charAt(i+j) != y.charAt(j))  
                    abweichungGefunden = true;  
            if (abweichungGefunden == false)  
                return i;  
        }  
        return -1;  
    }  
}
```

Musterlösung zu 4.2

Idee

Die Realisierung der Methode *f* ist einfach.

Die Methode *nullstelle* sucht nach einer Nullstelle im Bereich $[x,y]$. Voraussetzung ist, dass einer der beiden Funktionswerte $f(x)$ und $f(y)$ größer als 0 ist und der andere kleiner als 0. Es gibt zwei Möglichkeiten:

- ⇒ 1. Fall: $f(x) < 0$ und somit $f(y) > 0$
- ⇒ 2. Fall: $f(x) > 0$ und somit $f(y) < 0$

Hat man es mit dem 2. Fall zu tun, so ruft man die Funktion *nullstelle* mit vertauschten Parametern auf.

Jetzt kann man davon ausgehen, dass $f(x) < 0$ gilt und $f(y) > 0$. Man berechnet nun das arithmetische Mittel $m = (x+y)/2$.

Ist $f(m)$ eine Nullstelle, ist also der Absolutbetrag von $f(m)$ kleiner als die vorgegebene Genauigkeit z , so ist die Berechnung beendet und der Wert m wird als Ergebnis zurückgegeben.

Ist $f(m)$ keine Nullstelle und ist $f(m)$ kleiner als 0, so muss die Nullstelle zwischen m und y gesucht werden: rekursiver Aufruf der Methode *nullstelle* mit dem Intervall $[m,y]$.

Ist $f(m)$ keine Nullstelle und ist $f(m)$ größer als 0, so muss die Nullstelle zwischen x und m gesucht werden: rekursiver Aufruf der Methode *nullstelle* mit dem Intervall $[x,m]$.

Implementierung

```
public class Nullstellensuche {

    public static double nullstelle(double x, double y, double z) {
        if (f(x) > 0)
            return nullstelle(y, x, z);
        double m = (x + y) / 2;
        if (Math.abs(f(m)) < z)
            return m;
        else {
            if (f(m) < 0)
                return nullstelle(m, y, z);
            else
                return nullstelle(x, m, z);
        }
    }

    public static double f(double x) {
        return Math.exp(x) + x * x - 4.0;
    }
}
```

Musterlösung zu Fehler: Referenz nicht gefunden

Implementierung

```
public class Quadrat {  
    public static void quadrat(int breite) {  
        for (int i = 0; i < breite; i++)  
            for (int j = 0; j < breite; j++) {  
                if ((i == 0) || (i == breite - 1) ||  
                    (j == 0) || (j == breite - 1))  
                    System.out.print("*");  
                else  
                    System.out.print("-");  
                if (j == breite - 1)  
                    System.out.println();  
            }  
    }  
}
```

Testlauf:

```
public class Tester {  
    public static void main(String[] args) {  
        Quadrat.quadrat(4);  
        Quadrat.quadrat(7);  
        Quadrat.quadrat(1);  
    }  
}
```

Musterlösung zu 14.7

Implementierung

```
public class Geheimschrift {  
    public static char encrypt_char(char c, String x) {  
        for (int j = 0; j < x.length(); j++)  
            if (x.charAt(j) == c)  
                if (j < x.length() - 1)  
                    return x.charAt(j + 1);  
                else  
                    return x.charAt(0);  
        return c;  
    }  
    public static String encrypt(String s, String x) {  
        String w = "";  
        for (int i = 0; i < s.length(); i++)  
            w = w + encrypt_char(s.charAt(i), x);  
        return w;  
    }  
    public static char decrypt_char(char c, String x) {  
        for (int j = 0; j < x.length(); j++)  
            if (x.charAt(j) == c)  
                if (j > 0)  
                    return x.charAt(j-1);  
                else  
                    return x.charAt(x.length()-1);  
        return c;  
    }  
    public static String decrypt(String s, String x) {  
        String w = "";  
        for (int i = 0; i < s.length(); i++)  
            w = w + decrypt_char(s.charAt(i), x);  
        return w;  
    }  
}
```

Testlauf:

```
public class Tester {  
    public static void main(String[] args) {  
        String s = "Hallo Herr Müller!";
```



```
String x = "aKurHül!";  
System.out.println(s);  
String se = Geheimschrift.encrypt(s,x);  
System.out.println(se);  
String sed = Geheimschrift.decrypt(se,x);  
System.out.println(sed);  
}  
}
```