

Aufgabensammlung für die Vorlesung Objektorientierte Programmierung

Lehrstuhl Management der IT-Sicherheit

17. November 2008

Diese Aufgabensammlung wird in der Vorlesung Objektorientierte Programmierung verwendet. Verbesserungsvorschläge sind stets willkommen! Eine Übersicht der Aufgaben finden Sie ab Seite 2.

Aufgabenübersicht

1	Das erste Programm (Applikation) - Say Hello.java	5
2	Pi-Näherung	5
3	Rechner.java	6
4	Kontrollstrukturen und Operatoren - SimplerTaschenrechner.java	7
5	Anwendung von Schleifen - ZahlenRaten.java	7
6	Bruch (Rationale Zahlen)	8
7	IsKeywd.java	9
8	Fakultaet.java	10
9	guter und schlechter Programmierstil	10
10	Summenberechnung I	11
11	Summenberechnung II	11
12	Arithmetische Ausdrücke und Bibliotheksfunktionen - Quadratische Gleichungen	11
13	Mehrwertsteuererhöhung	12
14	Berechnung des Bruttokapitalwerts	12
15	Uhrzeit.java	12
16	Sortieren	13
17	Package mathe	13
18	Interface Comparable	14
19	Klasse Singleton	14
20	Ausnahmebehandlung - Bruch.java	14
21	Einfach verkettete Listen (Stack)	15
22	Ampel	16
23	Graphischer Bruch-Taschenrechner	16
24	Die Bottle-Klasse (Bottle.java)	16
25	Handhabung von Variablen und Klassen I (Output.java)	17
26	Arrays (Array.java)	17
27	Kontrolle des Programmflusses mit der if-Struktur I (Keyword.java)	18
28	Modifikation der Bottle-Klasse	18
29	Stein, Schere, Papier	19
30	Fakultät berechnen	20
31	Clock und DigitalClock	20
32	Exception-Handling: Das Interface Pet	21
33	Mehrdimensionale Arrays: Das Pascal'sche Schema	22
34	Tafel (Blackboard)	22
35	StringStorage	24
36	Ein GUI-Beispiel	24
37	Studienbegleitende Leistung WS 2005/ 2006	26
38	Studienbegleitende Leistung WS 2006/ 2007	28

Anmerkung: Die folgende Klasse `OOUtil.java` stellt häufig benötigte Methoden zur Verfügung und wird verwendet, um den Bearbeitern gerade zu Beginn die Konzentration auf das Wesentliche zu ermöglichen:

```
1 import java.io.BufferedReader;
2 import java.io.IOException;
3 import java.io.InputStreamReader;
4
5 public class OOUtil {
6     private final float version = 1.1f;
7     public String toString() { return "OOUtil v" + version; }
8
9     public int readInt() {
10        return readInt("Bitte geben Sie eine ganze Zahl ein: ");
11    }
12
13    public int readInt(String msg) {
14        try {
15            if (msg != null) System.out.print(msg);
16            BufferedReader input = new BufferedReader(new InputStreamReader
17                (System.in));
18            String number = input.readLine();
19            return Integer.parseInt(number);
20        } catch (NumberFormatException e) {
21            return readInt(msg);
22        } catch (IOException e) {
23            e.printStackTrace();
24        }
25        return 0;
26    }
27
28    public float readFloat() {
29        return readFloat("Bitte geben Sie eine Fließkommazahl ein: ");
30    }
31
32    public float readFloat(String msg) {
33        try {
34            if (msg != null) System.out.print(msg);
35            BufferedReader input = new BufferedReader(new InputStreamReader
36                (System.in));
37            String number = input.readLine();
38            return Float.parseFloat(number);
39        } catch (NumberFormatException e) {
40            return readFloat(msg);
41        } catch (IOException e) {
42            e.printStackTrace();
43        }
44        return 0;
45    }
46
47    public double readDouble() {
48        return readDouble("Bitte geben Sie eine Fließkommazahl ein: ");
49    }
50
51    public double readDouble(String msg) {
52        try {
53            if (msg != null) System.out.print(msg);
54            BufferedReader input = new BufferedReader(new InputStreamReader
```

```

53         (System.in));
54         String number = input.readLine();
55         return Double.parseDouble(number);
56     } catch (NumberFormatException e) {
57         return readDouble(msg);
58     } catch (IOException e) {
59         e.printStackTrace();
60     }
61     return 0.0;
62 }
63 public String readString() {
64     return readString("Bitte geben Sie einen String ein: ");
65 }
66
67 public String readString(String msg) {
68     try {
69         if (msg != null) System.out.print(msg);
70         BufferedReader input = new BufferedReader(new InputStreamReader(
71             System.in));
72         return input.readLine();
73     } catch (IOException e) {
74         e.printStackTrace();
75     }
76     return "";
77 }
78 public char readChar() {
79     return readChar("Bitte geben Sie einen Character ein: ");
80 }
81
82 public char readChar(String msg) {
83     if (msg != null) System.out.print(msg);
84     BufferedReader input = new BufferedReader(new InputStreamReader(
85         System.in));
86     try {
87         char c = (char)input.read();
88         if (c != '\n' && c != '\r' && c != '\t') {
89             return c;
90         } else {
91             return readChar(msg);
92         }
93     } catch (IOException e) {
94         e.printStackTrace();
95     }
96     return 'a';
97 }

```

Aufgabe 1 Das erste Programm (Applikation) - Say Hello.java

Lege ein neues Verzeichnis „Uebung1“ und Unterverzeichnis „Applikation“ an und speichere folgenden Code unter SayHello.java ab:

```
1 public class SayHello {
2     public static void main(String[] args) {
3         Mouth mouth = new Mouth();
4         mouth.say("Hello, world");
5     }
6 }
7
8 class Mouth {
9     public Mouth() {
10    }
11
12    public void say(String what) {
13        System.out.println(what);
14    }
15
16    public void kiss() {
17        System.out.println(":-*");
18    }
19 }
```

Wechsle jetzt in der Kommandozeile in das Verzeichnis, in dem die SayHello.java liegt. Die Kommandozeile bzw. die Eingabeaufforderung erhält man z.B. unter WindowsXP indem man mit der „Start“-Taste das Windows-Menü öffnet, dort „Ausführen“ anklickt und dann „cmd“ eingibt.

Weitere wichtige Kommandos unter Windows dafür sind:

- `dir` : gibt den Inhalt des aktuellen Verzeichnisses an
- `cd <Ordner>` : wechselt in den angegebenen Ordner des aktuellen Verzeichnisses
- `cd ..` : geht eine Ebene im Verzeichnisbaum höher

a) Kompiliere mit

```
javac SayHello.java
```

und führe aus mit

```
java SayHello
```

- b) Versuche das Programm `SayHello.java` so umzuschreiben, dass es `Hello, <Name>` ausgibt und speichere es unter `SayHelloName.java` ab. Vergiss dabei nicht, dass der Klassenname dann auch anders lauten muss.
- c) Provoziere selbst ein paar verschiedene Fehlermeldungen in dem Programm `SayHelloName.java`, indem du einfach Strichpunkte weglässt, Klammern löschst oder kleine Schreibfehler einbaust. Versuche dabei die Fehlermeldungen, die der Compiler ausgibt, zu deuten und richtig zu interpretieren.
- d) Füge in das Programm `SayHello.java` einige sinnvolle Kommentare ein. Probiere dabei auch die verschiedenen Arten von Kommentaren aus.
- e) Erweitere das Programm so, dass nach dem Ausgeben von „Hello World“ die Methode `kiss()` aufgerufen wird.

Aufgabe 2 Pi-Näherung

Schreibe eine Klasse `ApproxPi`, die eine Näherung für die Zahl π (nach dem Leibniz-Verfahren) berechnet. Sie besteht nur aus einer `main`-Methode und gibt die Näherung nach jedem Schritt als Gleitkommazahl auf dem Bildschirm aus.

Führe die Approximation doppelt durch. Zuerst soll die Näherung nach 20 Schritten abbrechen, während beim zweiten Mal die Abschätzung bis zu einer Genauigkeit von 0.05 fortgesetzt wird. Wähle dazu jeweils eine passende

Schleife.

Das Leibniz-Verfahren schätzt nach folgender Reihe:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} - \dots$$

1. Um den Einstieg zu erleichtern, ignoriere zu Beginn den Bruch und den Vorzeichenwechsel und berechne $1 + 3 + 5 + 7 + 9 + 11 + 13 - \dots$ und gebe es nach jedem Schritt aus.
2. Bringe nun den Vorzeichenwechsel mit ins Spiel: $1 - 3 + 5 - 7 + 9 - 11 + 13 - \dots$
(Hinweis: Benutze den Modulo-Operator um zu prüfen, ob eine Zahl gerade oder ungerade ist)
3. Um einen Bruch (z.B. $\frac{1}{3}$) zu erzeugen, muss immer 1.0 durch das Ergebnis der einzelnen Schritte (z.B. 3) geteilt werden. Das Ergebnis soll in eine `double`-Zahl sein.
(Hinweis: durch 1.0 und nicht 1 teilen, da sonst eine Integerdivision durchgeführt werden würde)
4. Die Brüche sollen nun nicht mehr nur als `double`-Zahlen gespeichert werden, sondern als Bruch-Objekte (siehe Bruch-Aufgabe).

Aufgabe 3 Rechner.java

Gegeben ist folgender Programmumpf, den man am besten unter `Rechner.java` speichert. ;-)

```
1 public class Rechner {
2     // lege willkürlich ein paar Werte für die Variablen a, b, c und d
   fest
3     int a = 0;
4     int b = 0;
5     int c = 0;
6     int d = 0;
7
8     public Rechner() {
9         // Lies für die Variablen andere Werte ein...
10        // OUtil.readInt() liest einen int-Wert von der Standardeingabe
11    }
12
13    public int addAll() {
14        int sum = 0;
15        // Das Zusammenzählen aller Zahlen in der Variablen sum
16        return sum;
17    }
18
19    public void swap() {
20        // swap soll die Werte der Variablen a und b vertauschen
21        // vor dem Tauschen
22        System.out.println("a hat den Wert: " + a + " und b hat den Wert: "
23            + b + ".");
24        // hier dann das Tauschen
25
26        // Kontrolle:
27        System.out.println("a ist jetzt: " + a + " und b hat den Wert: " + b
28            + ".");
29    }
30
31    public double calcAverage() {
32        double avg = 0.0;
33        // Berechne den Mittelwert von a, b, c und d
34        return avg;
35    }
36
37    public static void main(String[] argv) {
```

```

36     Rechner rechnerInstanz = new Rechner();
37     System.out.println(rechnerInstanz.addAll());
38     rechnerInstanz.swap();
39     // rechnerInstanz.printOutMaximum();
40     System.out.println(rechnerInstanz.calcAverage());
41 }
42 }

```

- Implementiere die Methode `addAll()`, die alle Variablen aufsummiert und das Ergebnis zurückgibt.
- Weise den Variablen `a,b,c` und `d` im Konstruktor neue Werte zu. Verwende hierzu die Methode `readInt()` aus `OOUtil`, die `int`-Werte von der Kommandozeile einliest.
- Implementiere die Methode `swap()`, welche die Werte zweier Variablen (hier `a` und `b`) vertauscht. Dazu brauchst du eine Hilfsvariable.
- Implementiere die Methode `printOutAverage()`, die das arithmetische Mittel der 4 Variablenwerte berechnet. Hinweis: Das arithmetische Mittel muss keine Ganzzahl sein. Um Fehler zu vermeiden solltest du deshalb bei der Division durch `4.0` und nicht durch `4` teilen. Das liegt an der Typkonvertierung in Java. Hintergründe in der Übung.
- Implementiere nun die Methode `printOutMaximum()`, bei der du den größten Wert der Variablen aus gibst. Du kannst dies mit 3 oder allen 4 Variablen machen.

Aufgabe 4 Kontrollstrukturen und Operatoren - SimplerTaschenrechner.java

Schreibe eine Applikation `SimplerTaschenrechner.java`, die schrittweise 2 Zahlen sowie einen Rechenoperator (+, -, *, /, %) von der Standardeingabe einliest. Anschließend soll das Ergebnis der Verknüpfung der 2 Zahlen mit dem Operator ausgegeben werden. Wähle für die beiden Zahlen und für das Ergebnis jeweils einen geeigneten Datentyp. Die Rechenzeichen können nicht direkt als Operatoren erkannt werden sondern nur als einzelnes Zeichen (verwende den Datentyp `char`). Daher muss eine Abfrage existieren, die ein Mapping in der Art

```

1  if (operator == '+') { result = a + b; }

```

vornimmt.

Statt dieser `if`-Abfrage ist es hier übersichtlicher, eine `switch`-Struktur zu verwenden.

Erweitere anschließend das Programm, so dass bei Eingabe eines falschen Operators die Eingabe solange wiederholt wird, bis ein gültiger Wert eingegeben wurde.

Aufgabe 5 Anwendung von Schleifen - ZahlenRaten.java

Das Spiel „Zahlen raten“ funktioniert so: Ein Spieler A denkt sich eine Zahl zwischen 0 und einer bestimmten Obergrenze aus. Der Spieler B muss nun versuchen die Zahl zu erraten. Nach jedem Versuch bekommt er dazu von A den Hinweis, ob seine geratene Zahl zu groß oder zu klein ist. Errät er die Zahl richtig, endet das Spiel.

Dazu soll eine Klasse `Zahlenraten` entstehen, welche das komplette Spiel in der `main`-Methode realisiert. Zu Beginn wird eine Zufallszahl zwischen 1 und 100 ermittelt. Verwende dazu folgenden Befehl:

```

4     int randValue = (int) (Math.random() * 100) + 1;

```

Anschließend versucht der Benutzer die zufällig generierte Zahl zu erraten und gibt seinen Tipp ein. Nach dem Rateversuch erhält er ein Feedback („Zahl zu hoch“ oder „Zahl zu niedrig“). Dieser Vorgang wird so lange wiederholt, bis der Nutzer die richtige Zahl geraten hat. Am Ende des Spiels wird außerdem angezeigt wie viele Versuche er benötigt hat.

Ist dein Programm fertig, dann spiele solange, bis du das unterboten hast:

```

C:\WINDOWS\system32\cmd.exe
Bitte geben Sie eine Zahl ein: 50
-> Zu hoch...

Bitte geben Sie eine Zahl ein: 25
-> Zu niedrig...

Bitte geben Sie eine Zahl ein: 37

GEWONNEN!!      nach 3 Versuchen

D:\>Pause
Drücken Sie eine beliebige Taste . . .

```

Erweiterungen:

- Erweitere das Programm um eine durch den Benutzer frei wählbare Obergrenze.
- Erweitere das Programm so, dass mehrere Ratespiele (Rate-Sessions) mit der am Anfang festgelegten Obergrenze erfolgen können. Am Ende einer Session wird der Benutzer gefragt, ob er noch mal spielen will.
- Ergänze das Programm um eine Statistikfunktion, die ausgibt, wie viele Versuche der Spieler im Durchschnitt benötigt hat. Vergleiche das Ergebnis mit dem des Nachbarns!
- Falls Dir selbst raten zu langweilig wird, ergänze das Programm um eine Komponente, bei der sich der Spieler die Zahl ausdenkt und der Computer versucht die Zahl zu erraten.

Aufgabe 6 Bruch (Rationale Zahlen)

In der Vorlesung wurde das Beispiel Bruch.java vorgestellt (hier leicht modifiziert):

```

1  /**
2   * Implementiert das Verhalten von einem Bruch mit verschiedenen
3   * Operationen
4   */
5  public class Bruch {
6      /** Es folgen die Attribute */
7      int zaehler;
8      int nenner;
9
10     /** Konstruktor */
11     public Bruch(int z, int n) {
12         zaehler = z;
13         nenner = n;
14         kuerze();
15     }
16
17     /** Hier eine Methodendefinition zum Kürzen */
18     public void kuerze() {
19         // ...
20     }
21
22     /** Methode zum Hinzuaddieren */
23     public void add(Bruch r) {
24         // System.out.println(this + "+" + r);
25         zaehler = zaehler * r.nenner + r.zaehler * nenner;
26         nenner = nenner * r.nenner;
27         kuerze();
28     }
29
30     /** Methode zum Wandeln in einen String */

```

```

30 public String toString() {
31     return "(" + zaehler + "/" + nenner + ")";
32 }
33 }

```

Löse folgende Teilaufgaben:

1. Vervollständige die Methode `kuerze()`. Diese soll die interne Darstellung des Bruchs in eine gekürzte Form bringen. Außerdem soll nach dem Aufruf von `kuerze()` der Nenner immer positiv sein.
Hinweis: Die Klasse `OOUtil` enthält die Methode `ggT` zur Berechnung des größten gemeinsamen Teilers.
2. Füge einen weiteren Konstruktor hinzu, der als Parameter ein Bruch-Objekt erwartet, um dem neuen Bruch-Objekt Zähler und Nenner des übergebenen Objektes zuzuweisen.
Verwende dazu folgenden Code:

```

10 public Bruch(Bruch b) {
11     zaehler = b.zaehler;
12     nenner = b.nenner;
13 }

```

3. Schreibe eine Applikation `BruchTest` um die Implementierung der Klasse `Bruch` zu testen (vgl. Vorlesung). Speichere die Datei `BruchTest.java` im selben Verzeichnis wie `Bruch.java`.
Eine mögliche Testklasse `BruchTest` könnte in etwa so aussehen:

```

1 public class BruchTest {
2     public static void main(String[] args) {
3         Bruch bruch1 = new Bruch(1, 2);
4         Bruch bruch2 = new Bruch(1, 3);
5         System.out.println("bruch1 = " + bruch1);
6         System.out.println("bruch2 = " + bruch2);
7         bruch1.add(bruch2);
8         System.out.println(bruch1);
9     }
10 }

```

4. Implementiere analog zur `add`-Methode folgende Methoden in der `Bruch`-Klasse und verändere auch entsprechend die Klasse `BruchTest`, um die neuen Methoden zu testen.
 - erstelle eine Methode `getDecimal()`, die den Bruch als Gleitkommazahl zurückgibt
 - `public void sub(Bruch b)` soll `Bruch b` von dem `Bruch` subtrahieren, mit dem diese Methode aufgerufen wurde
 - mit `kehrwert()` wird der Bruch auf seinen Kehrwert gesetzt, einen Rückgabewert gibt es hierbei nicht
 - zwei weitere Methoden heißen `mul` und `div` und haben als Parameter jeweils ein `Bruch`-Objekt, aber keinen Rückgabewert. Das aufrufende Objekt wird verändert, indem es im ersten Fall mit dem übergebene Objekt multipliziert und im zweiten Fall dividiert wird.
5. Erzeuge mit Hilfe von `javadoc` eine Dokumentation zur Klasse `Bruch`. Achte dabei auf möglichst sinnvolle Kommentare.
(Hinweis: [JavaSun - How To Write DocComments](#))

Aufgabe 7 `IsKeywd.java`

In der Vorlesung kam das Beispiel `IsKeywd.java` vor, das genau einen Kommandozeilenparameter auf ein gültiges Schlüsselwort auswertet. Hier nun eine ein klein wenig abgeänderte Version:

```

1 class IsKeywd {
2     public static void main(String[] argv) {
3         // list of keywords defined in Java

```

```

4 String[] keywords = { "abstract", "default", "if", "private", "this"
    , "boolean", "do", "implements", "protected", "throw", "break", "
double", "import", "public", "throws", "byte", "else", "
instanceof", "return", "transient", "case", "extends", "int", "
short", "try", "catch", "final", "interface", "static", "void", "
char", "finally", "long", "strictfp", "volatile", "class", "float
", "native", "super", "while", "const", "for", "new", "switch", "
continue", "goto", "package", "synchronized", "assert", "enum"};
5 // check for command line
6 if (argv == null || argv.length != 1) {
7     System.out.println("Usage: iskeyword <keyword>");
8     System.exit(1);
9 }
10
11 String word = argv[0];
12 boolean found = false;
13 int i = 0;
14 do {
15     if (keywords[i].equals(word)) {
16         found = true;
17     }
18     i++;
19 } while (i < keywords.length && found==false);
20 //Verkettung von 2 Bedingungen durch '&&':
21 //1. i kleiner als die Länge des keyword-Arrays
22 //2. found==false ist äquivalent zu !found
23     // display result
24 if (found) {
25     System.out.println(word + " ... yes, found!");
26 } else {
27     System.out.println(word + " ... no, not found!");
28 }
29 }
30 }

```

Schreibe das Programm so um, dass es auch mehrere Parameter (Wörter) entsprechend auswertet.

Um zwei Strings auf Gleichheit zu testen, kannst du leider nicht den == Operator benutzen, da ein String ein Objekt ist. Dafür gibt es eine Methode aus der Klasse String: equals(Object o). Schaue bitte in der API nach, was diese Methode genau macht. Die Verwendung siehst Du im obigen Code.

Aufgabe 8 Fakultät.java

Schreibe ein Programm, das die Fakultät einer vom Benutzer als Kommandozeilenparameter einzugebenden Zahl berechnet und am Bildschirm ausgibt.

Hinweise: Der Algorithmus zur Berechnung der Fakultät wurde in der Vorlesung vorgestellt. Die Umwandlung von einem String in eine Zahl übernimmt die Funktion Integer.parseInt(String s). Nähere Hinweise zur Verwendung gibt es in der Übung und in der API Dokumentation.

Um den Überlauf zu vermeiden kannst Du mit der Klasse BigInteger aus dem package java.math arbeiten (freiwillig).

Aufgabe 9 guter und schlechter Programmierstil

Gegeben ist folgender fast unlesbarer Code:

```

1 public class Test {public static void main(String[] args) {
2 Test t = new Test();
3 System.out.println(t.doSomething(1000));}
4 double doSomething(int schritte) {
5 double pi=0.0;

```

```

6  for(double i=0,n=-1.0,z=1.0;i<schritte; n+=2,pi+=z/n,z*=-1,++i);
7  return pi*4;
8  }
9  }

```

Schreibe das oben gegebene Programm so um, dass auf den ersten Blick ersichtlich ist, was es eigentlich tut. Folgendes kann Dir dabei helfen:

- Einrückungen und richtige Strukturierung
- Vergeben aussagekräftiger Namen
- Umschreiben der for-Schleife, d.h. z.B. Ausdrücke aus dem Schleifenkopf in den Schleifenkörper übertragen
- Kommentare einfügen

Aufgabe 10 Summenberechnung I

Schreibe verschiedene Varianten eines Programms, welches die Summe aller Vielfachen von 3 bis zu einer vom Benutzer vorgegebenen Grenze berechnet. Das Resultat soll auf dem Bildschirm ausgegeben werden. Folgende Varianten sollen realisiert werden.

- Summenberechnung
 - unter Verwendung einer for Schleife
 - unter Verwendung einer while Schleife
 - unter Verwendung einer do-while Schleife
- Eingabe der Obergrenze
 - über Standardeingabe (OOUtil verwenden)
 - als Kommandozeilenparameter
- Integration des Berechnungsalgorithmus
 - direkt in die main-Methode
 - in separate Methode, die aus der main-Methode aufgerufen wird (hierzu muss dann auch eine Instanz der umgebenden Klasse erzeugt werden)
 - * mit Ausgabe des Resultats innerhalb der Methode
 - * mit Ausgabe des Resultats in der main-Methode

Aufgabe 11 Summenberechnung II

Entwickeln Sie ein Programm, das die Summe $1 - 1/2 + 1/3 - 1/4 + \dots + 1/9999 - 1/10000$ auf die folgenden vier Arten berechnet:

- Addition der Terme strikt von links nach rechts,
- Addition der Terme strikt von rechts nach links,
- getrennte Addition der positiven und negativen Terme, von links nach rechts, und
- getrennte Addition der positiven und negativen Terme, von rechts nach links.

Verwende für die Summe und die einzelnen Summanden den Datentyp `double` und nicht die `Bruch`-Klasse aus den letzten Aufgabenblättern. Führe das Programm aus und vergleiche die Resultate. Warum die einzelnen Varianten zu unterschiedlichen Resultaten führen, wird in der nächsten Zentralübung erklärt.

Aufgabe 12 Arithmetische Ausdrücke und Bibliotheksfunktionen - Quadratische Gleichungen

Schreibe ein Programm `SolveQuadratic`, welches drei `double`-Koeffizienten `a`, `b` und `c` vom Benutzer einliest und danach das Polynom der Form $f(x) = ax^2 + bx + c$ ausgibt.

Nun soll die Gleichung $f(x) = 0$ gelöst werden.

Unterscheide dazu im Folgenden ob das Polynom quadratisch (d.h. $a \neq 0$) oder linear ist. Der Fall eines konstanten Polynoms muss aber nicht berücksichtigt werden.

Zuerst wird der Benutzer über die Anzahl der erwarteten Lösungen informiert (Auswertung der Diskriminante im quadratischen Fall). Bestimme anschließend die Lösung(en), wobei bei einem Polynom vom Grad 2 die bekannte

Formel $x_{1/2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ verwendet werden sollte.

Methoden zur Berechnung von Potenzen und Quadratwurzeln findest du in der Klasse `Math`. Durchsuche dazu die Dokumentation dieser Klasse nach den Methoden `sqrt` und `pow` und verwende beide an geeigneter Stelle.

Im Übrigen sind alle Methoden der Klasse `Math` statische Methoden, d.h. du kannst sie benutzen ohne vorher eine Instanz der Klasse erzeugt zu haben.

```

C:\WINDOWS\system32\cmd.exe
Bitte einen Wert fuer a eingeben:      1
Bitte einen Wert fuer b eingeben:      3
Bitte einen Wert fuer c eingeben:      2

Das eingegeben Polynom lautet  1.0x^2 + 3.0x + 2.0

Es gibt zwei Lsg:      -1.0 und -2.0

D:\>Pause
Drücken Sie eine beliebige Taste . . .
  
```

Aufgabe 13 Mehrwertsteuererhöhung

Der Mehrwertsteuersatz beträgt normalerweise 19 % bzw. für Grundgüter gilt ein ermäßigter Steuersatz von 7 %. Schreibe nun ein Programm, in dem von einem gegebenen Preis der Anteil an Steuern berechnet werden kann. Hierfür muss der Benutzer eine Eingabe tätigen, wobei ein Zeichen eingelesen wird und geprüft wird ob es sich um ein e für ermäßigt oder um ein r für regulär handelt. Mit der Eingabe einer 0, soll es dem Benutzer möglich sein, das Programm zu verlassen. Realisiere dies einmal mit Hilfe des Konstrukts `switch-case` und einmal mit `if-else`.

Aufgabe 14 Berechnung des Bruttokapitalwerts

Um die Vorteilhaftigkeit von Investitionsprojekten feststellen zu können ist das Konzept des Bruttokapitalwerts (BKW) weit verbreitet. Dabei werden die zukünftig erwarteten jährlichen Einnahmen aus einem Projekt mit Hilfe eines Zinssatzes auf eine Kennzahl verdichtet. Der BKW spiegelt den Wert eines Projektes bezogen auf den jetzigen Zeitpunkt wieder.

Der BKW kann über die Formel $BKW = \sum_{t=1}^n b_t(1+i)^{-t} = \sum_{t=1}^n \frac{b_t}{(1+i)^t}$ berechnet werden, wobei b_t =Einnahme in der Periode t ,

t =Zeitpunkt (Periode) und

i =der (für alle Perioden einheitliche) Zinssatz ist.

Für ein konkretes Projekt mit einem Zinssatz von i und einer Laufzeit von 8 Perioden mit folgender Zahlungsreihe bedeutet dies beispielsweise:

Zeitpunkt t	0	1	2	3	4	5	6	7	8
Einnahmen		10	10	10	20	30	40	40	80

$$BKW = \sum_{t=1}^n \frac{b_t}{(1+i)^t} = \frac{10}{1,1^1} + \frac{10}{1,1^2} + \frac{10}{1,1^3} + \frac{20}{1,1^4} + \frac{30}{1,1^5} + \frac{40}{1,1^6} + \frac{40}{1,1^7} + \frac{80}{1,1^8} = 137,58$$

Folglich hat dieses Projekt im Zeitpunkt 0 den Wert 137,58, d.h. um ein wirtschaftlich erfolgreiches Projekt zu realisieren, sollten hier die Anschaffungskosten nicht mehr als 137,57 betragen.

Schreibe ein Programm, das die Berechnung des Bruttokapitalwertes vornimmt und ausgibt. Hierzu brauchst Du vom Benutzer Daten über den Zinssatz i , die Laufzeit des Projektes und die Einnahmen in jeder Periode, die Du über die Klasse `OOUtil` einlesen kannst. Die Einnahmen sollen in einem Array vom Typ `double` gespeichert werden, wobei das Array so groß ist wie die Anzahl der Perioden (Laufzeit) in diesem Projekt.

Das Runden von `double`-Werten auf beliebig viele Nachkommastellen kann nun die Klasse `OOUtil` übernehmen.

Aufgabe 15 Uhrzeit.java

Es soll die Java-Klasse `Uhrzeit` implementiert werden. Die Klasse soll die Attribute `Stunde`, `Minute`, `Sekunde` enthalten. Im Konstruktor wird die Uhrzeit als String der Form "HH:MM:SS" übergeben. Mit den Methoden `add(String value)` und `sub(String value)` soll die Uhrzeit um den angegebenen Wert erhöht bzw. verringert werden. Wie jede Klasse erbt auch die Klasse `Uhrzeit` die Methode `toString()` von der Klasse `Object`. Überschreiben Sie diese Methode so, dass die Uhrzeit in der Form "HH:MM:SS" ausgegeben wird.

Hinweise:

- Um den im Konstruktor übergebenen String zu zerlegen, gibt es bereits Methoden in der API, z.B.: `String.split()`. Alternativ kann die Klasse `StringTokenizer` verwendet werden.
- Es ist zu beachten, dass es für die Attribute bestimmte Wertebereiche gibt (z.B. Sekunde `[0;59]`). Wenn der Stundenbereich überschritten wird, werden trotzdem keine Tage gezählt.

Ergänzen Sie die Klasse `Uhrzeit` um

- einen Konstruktor, der Stunden, Minuten und Sekunden als `int`-Werte entgegennimmt
- einen Konstruktor ohne Argumente, der die Klasse mit der aktuellen Systemzeit initialisiert (Die Systemzeit bekommen Sie unter Verwendung der Klasse `GregorianCalendar`.)
- die Methode `add(int seconds)`, die die Uhrzeit um die übergebene Anzahl von Sekunden erhöht
- die Methode `sub(int seconds)`, die die Uhrzeit um die übergebene Anzahl an Sekunden verringert.

Schreiben Sie auch eine Applikation `UhrzeitTestApp` um Ihre Implementierung der Klasse `Uhrzeit` zu testen. Dokumentieren Sie alle Ihren selbst erstellten Klassen mit Hilfe von javadoc-Kommentaren.

Aufgabe 16 Sortieren

Implementieren Sie eine Klasse `Sorting` mit den Funktionen `long[] bubbleSort(long[] a)` und `long[] selectionSort(long[] a)`. Testen Sie Ihre Implementierung mit der folgenden Testklasse:

```

1 public class SortingTest {
2     public static void main(String[] args) {
3         long[] a = { 1, 88, 12, 10, 29 };
4         System.out.println("normal:");
5         printOutArray(a);
6         System.out.println("bubbleSort:");
7         printOutArray(Sorting.bubbleSort(a));
8
9         long[] b = { 1, 88, 12, 10, 29 };
10        System.out.println("normal:");
11        printOutArray(b);
12        System.out.println("selectionSort:");
13        printOutArray(Sorting.selectionSort(b));
14    }
15
16    static void printOutArray(long[] array) {
17        for (int i = 0; i < array.length; i++) {
18            System.out.print(array[i] + " ");
19        }
20        System.out.println("");
21    }
22 }

```

Überlegen Sie dabei auch, welche Effekte der Aufruf der Methode auf dem ursprünglichen Array hat. Warum ist das so?

Aufgabe 17 Package mathe

Ein Package ist eine logische Gruppierung von Klassen und wird verwendet, um die Zusammengehörigkeit von größeren Strukturen zu verdeutlichen. Aus der Vorlesung und dem Übungsblatt 4 ist die Klasse `Bruch` bekannt. Legen Sie nun ein Verzeichnis mit dem Namen `mathe` an, in das Sie die Klasse `Bruch` legen und kennzeichnen Sie diese als zum Package `mathe` gehörend.

Auf derselben Ebene, auf der sich das Verzeichnis `mathe` befindet, speichern Sie nun Ihre Applikation `BruchTest.java`. Damit die Applikation weiß, mit welchen Klassen sie arbeiten soll, sollten Sie das Package `mathe` importieren.

Hinweise:

- Die Zugehörigkeit zu einem Package wird zu Beginn der Klassendefinition mit Hilfe der package-Anweisung vorgenommen:
z.B.: `package xyz;`
`class Qwert{...}`
- Im Package selbst befindet sich im Regelfall keine Applikation.
- Die Verwendung von zu importierenden Klassen kennen Sie bereits aus der API-Specification.
- Alle Klassen des Packages importieren Sie mit `import mathe.*;`
- Nur die Klasse `Bruch` importieren Sie mit der Anweisung `import mathe.Bruch;`
- Es gibt auch die Möglichkeit die Applikation ohne import-Anweisung zu schreiben. Hierfür müssen Sie die Klasse über den vollständig qualifizierten Namen ansprechen (hier z.B. `mathe.Bruch bruch1 = new mathe.Bruch(2,3);`)

Aufgabe 18 Interface Comparable

Damit zwei Brüche miteinander verglichen werden können, kann das Interface `Comparable` verwendet werden. Erweitern Sie dazu Ihre Klasse `Bruch` um das o.g. Interface und implementieren Sie die notwendige Methode. Hinweise:

- Ein Interface wird mit `implements` im Klassenrumpf in die Klasse aufgenommen
z.B.: `class ThreadDemo3Frames extends Frame implements Runnable`
- Die Beschreibung des Interfaces mit den notwendigen zu implementierenden Methoden finden Sie in der API-Specification unter `Comparable`. Lesen Sie hier bitte sorgfältig.
- Für die Verwendung von `Bruch`-Objekten in der Methode müssen Sie das als Parameter übergebene Object zuerst auf ein `Bruch`-Objekt casten: `Bruch b = (Bruch) object;`
- Klassen, die auch das Interface `Comparable` verwenden sind z.B. `String` und `GregorianCalendar`.

Aufgabe 19 Klasse Singleton

Eine Klasse, die nur ein einziges Mal instanziiert werden kann, bezeichnet man als `Singleton`. Schreiben Sie eine solche Klasse mit dem Namen `MySingleton` nach den Vorgaben des unten stehenden Rahmenprogramms. Prüfen Sie mit der folgenden Applikation `SingletonRahmen` die korrekte Funktionsweise Ihrer `Singleton`-Klasse:

```

1  class SingletonRahmen {
2      private MySingleton aSingleton, another;
3
4      public static void main(String[] args) {
5          SingletonRahmen c = new SingletonRahmen();
6          c.aSingleton = MySingleton.createMySingleton();
7          if (c.aSingleton != null) {
8              System.out.println(" OK: got first object");
9              c.another = MySingleton.createMySingleton();
10             if (c.another != null){
11                 System.out.println("NOK: there should be only one object");
12             } else {
13                 System.out.println(" OK: fine, it's a singleton");
14             }
15         } else {
16             System.out.println("NOK: no first object");
17         }
18     }
19 }

```

Aufgabe 20 Ausnahmebehandlung - Bruch.java

Ein Problem bei der Berechnung von Brüchen ist die Gefahr einer Division durch 0. Erweitern Sie die Klasse `Bruch` so, dass eine `NullPointerException` auftritt sobald der Nenner 0 ist. Implementieren Sie eine neue

Klasse `BruchTestApplikation`, in der Sie den ersten Bruch in einer eigenen Methode einlesen lassen. Beachten Sie hierbei, dass der Benutzer ein Feedback und eine zweite Chance bekommen sollte, wenn er eine 0 eingegeben hat.

Sorgen Sie zusätzlich dafür, dass die Veränderung der Attribute `nenner` und `zaehler` von außen nicht möglich ist. Verwenden Sie hierzu einen geeigneten Modifier. Implementieren Sie die Methoden `int getZaehler()`, `int getNenner()`, `void setZaehler(int z)` und `void setNenner(int n)`. Auch hier soll die oben genannte Exception auftreten, wenn der Nenner 0 ist.

Hinweise:

- Damit eine `NennerNullException` auftreten kann, sollten Sie zuerst diese implementieren. D.h. Sie schreiben eine eigene Klasse, die sinnvollerweise von `Exception` erbt und überschreiben nach ihrem Wunsch die verschiedenen Methoden. Um feststellen zu können, welche Methoden es in `Exception` gibt, sollten Sie in der API-Specification nachschlagen.
- Sie können gerne in die Klasse `OOUtil` schauen, wie dort etwaige Fehler abgefangen werden.

Mögliche Erweiterung:

- Lassen Sie die Zahlen für den ersten Bruch von der Kommandozeile einlesen und fangen Sie dabei evtl. Falscheingaben, d.h. zu wenige Argumente bzw. Strings anstatt Zahlen ab.

Aufgabe 21 Einfach verkettete Listen (Stack)

Implementieren Sie einen universell einsetzbaren Stack, der auf Objekten basiert. Der Stack soll folgendes Interface implementieren:

```
1 public interface StackInterface{
2     public void push(Object o);
3     public Object pop();
4     public boolean stackIsEmpty();
5 }
```

Schreiben sie hierzu auch eine kleine Testapplikation, in der Sie die Funktionalität des Stacks kurz testen. Eine solche Applikation könnte in etwa folgendermaßen aussehen:

```
1 public class StackTestApplikation {
2     public static void main(String[] args) {
3         new StackTestApplikation();
4     }
5
6     public StackTestApplikation() {
7         Stack s = new Stack();
8         try {
9             s.push(new Bruch(1, 3));
10            s.push(new Bruch(1, 2));
11            s.push(new Bruch(1, 1));
12            while (!s.stackIsEmpty()) {
13                Bruch b = (Bruch) s.pop();
14                System.out.println(b);
15            }
16        } catch (Exception e) {
17            e.printStackTrace();
18        }
19    }
20 }
```

Mögliche Erweiterung:

- Erweitern Sie Ihre Implementierung des Stacks um eine Ausnahmebehandlung. Anregungen hierzu finden Sie in den Vorlesungsunterlagen.

Aufgabe 22 Ampel

Schreiben Sie ein Programm, das eine Ampel in einem Frame darstellt. Die Ampel soll aus drei gleich grossen Kreisen bestehen. Jedes Mal, wenn der Frame neu gezeichnet wird, soll sich die Farbe der Ampel verändern. Dies geschieht, sobald der Benutzer in dem Frame klickt. Das Programm soll sich per Maus-Klick schliessen lassen.

Aufgabe 23 Graphischer Bruch-Taschenrechner

Schreiben Sie eine Applikation `BruchTaschenrechnerApp` mit graphischer Oberfläche, die im Wesentlichen aus folgenden Bausteinen besteht:

- Textfelder zur Eingabe von Brüchen (4 Stück sind hier zweckmässig) mit Beschriftung
- Eine Liste (Checkboxes oder RadioButtons) mit verschiedenen Operationen (mindestens Addition, Subtraktion, Multiplikation und Division)
- Einem Button zum Berechnen des Ergebnisses
- Einem Anzeigefeld für das Ergebnis (nicht editierbares Textfeld oder Label)

Bei einem Klick auf '=' soll das Ergebnis berechnet und anschließend auf dem Bildschirm ausgegeben werden. Hinweise:

- Aus den Textfeldern kann man nur einen String auslesen, der dann geparkt werden muss.
- Vergessen Sie nicht jedem Button einen `ActionListener` zuzuweisen.
- Zum Berechnen der Ergebnisse sollten Sie die Klasse `Bruch` verwenden.
- Hinweise zu den graphischen Elementen mit ihren Methoden gibt es in der API-Specification. Verwenden Sie möglichst nur Elemente aus `javax.swing.*`;

Erweiterungen:

- Bieten Sie weitere Operationen an (Kehrwert-Berchnung, kürzen, Anzeige als Dezimalzahl, ...).
- Integrieren Sie Ihre Ausnahmebehandlung aus der letzten Übung.

Aufgabe 24 Die Bottle-Klasse (Bottle.java)

Implementieren Sie eine Klasse `Bottle`, die wie folgt aussehen soll:

```
1 public class Bottle {
2     // Konstante - kann nicht nachträglich geändert werden
3     public final double MAX_FILL = 1.5;
4
5     // Attribut
6     private double actFill;
7
8     // Der Konstuktur ist eine besondere Methode (Syntax!), die einmalig
9     // bei
10    // Erschaffung des Objektes automatisch aufgerufen wird
11    public Bottle() {
12        // Standardkonstruktor - Setzt in diesem Fall das Attribut auf den
13        // Startwert 0 (dies kann alternativ auch bei der Deklaration des
14        // Attributes geschehen)
15        actFill = 0.0;
16    }
17
18    public Bottle(double fill) {
19        // Zusätzlicher Konstruktor mit Parameter (erspart einen Aufruf
20        // von setFill)
21        actFill = fill;
22    }
23
24    // allgemeine Methoden
25    public double getFill() {
```

```

23     // Gibt den aktuellen Füllstand zurück (der Benutzer kann ja nicht
24         auf actFill zugreifen)
25     return actFill;
26 }
27 public void setFill(double newFill) {
28     // Die Füllmenge wird auf den übergebenen Stand GESETZT
29     actFill = newFill;
30 }
31
32 public void changeFill(double changeFill) {
33     // Die Füllmenge wird um den übergebenen Wert GEÄNDERT
34     actFill = actFill + changeFill;
35 }
36
37 public String toString() {
38     // Es wird eine kurzer String zurückgegeben, der Informationen
39     // über das Bottle-Objekt enthält
40     String str = "Flasche " + actFill + "/" + MAX_FILL;
41     return str;
42 }

```

Aufgabe 25 Handhabung von Variablen und Klassen I (Output.java)

- Schreibe eine Klasse mit dem Namen `Output`, die nur aus der `main`-Methode besteht. Definiere innerhalb dieser Methode eine Variable vom Typ `int` mit dem Namen `a` und weise ihr anschließend den Wert 2 zu. Definiere danach eine weitere solche Variable `b` und weise ihr den Wert 3 zu. Lege nun eine `int`-Variable `c` an, der anschließend die Summe aus `a` und `b` zugewiesen wird. Gib `c` sinnvoll an den Bildschirm aus (z.B. „Summe: ...“). Weise dann der Variable `c` das Produkt aus `a` und `b` zu und gib `c` wieder aus.
- Schreibe eine Klasse `Function` die neben einem leeren Standardkonstruktor über das Attribut `factor` vom Typ `int` und die Methoden `setFactor` sowie `multiply` verfügt. Die erste Methode soll das Attribut auf einen neuen Wert setzen, während die zweite das `factor`-fache eines übergebenen `int`-Parameters zurückgibt. Tipp: Nimm dir die bekannten Klassen `Bruch` und `Bottle` zur Hilfe.
- Speichere die Quellcode-Dateien beider Klassen im selben Verzeichnis. Erweitere die `main`-Methode von `Output` wie folgt: Erschaffe ein Objekt der Klasse `Function` mit dem Namen `f` und benutze sie, um `c` das $(a+b)$ -fache von `a` zuzuweisen. Gib anschließend den Wert von `c` aus.

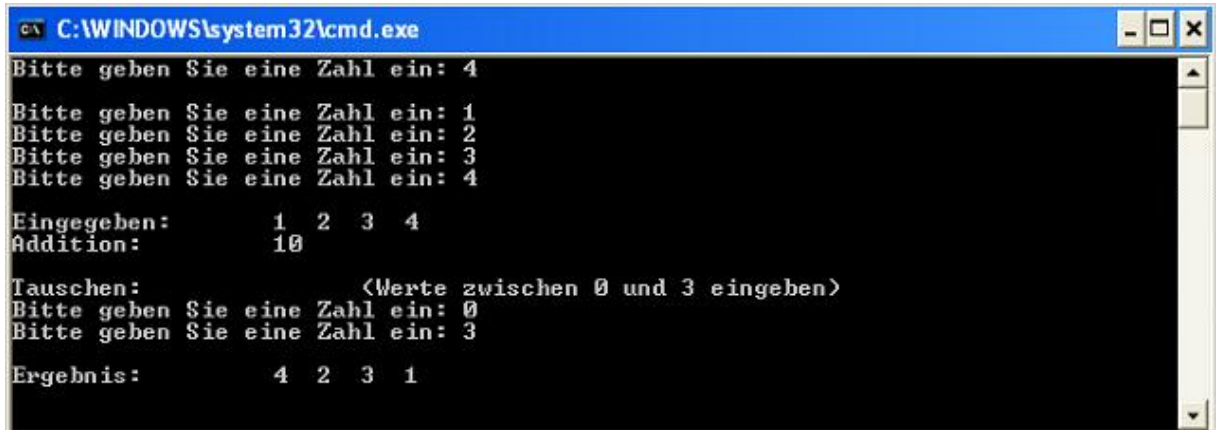
Aufgabe 26 Arrays (Array.java)

- Erstelle eine Klasse `Array`, welche als Attribut ein `int`-Array enthält. Dieses soll allerdings erst im Konstruktor erstellt werden (z.B. `private int[] a;`). Der Konstruktor der Klasse besitzt als Parameter den `int`-Wert `size`, welcher über die Größe des Arrays Auskunft gibt. Erstelle anhand dieser Informationen das Attribut mit dem `new`-Operator. Dieses Array wird auch gleich im Konstruktor mit Werten gefüllt. Verwende eine `for`-Schleife und die Klasse `OOUtil` um vom Benutzer `size` Zahlen zu erfahren. An weiteren Methoden enthält die Klasse `print`, `printAdd` und `change`. `print` gibt die Werte des Arrays einzeln an den Bildschirm aus, während `printAdd` die Summe der Werte ausgibt. `change` schließlich hat `i1` und `i2` als Parameter und vertauscht die Werte an den Positionen `i1` und `i2` des Attributes.
- Erstelle eine Klasse `ArrayApp` die nur aus einer `main`-Methode besteht. Als erstes soll eine Zahl vom Benutzer erfahren werden. Sie dient als Parameter für eine Instanz `a` der in der

ersten Teilaufgabe definierten Array-Klasse.

Anschließend werden die Methoden `print`, `printAdd` und `change` von `a` aufgerufen.

Vor dem Aufruf von `change` müssen zwei weitere Zahlen vom Benutzer eingegeben werden, die dann als Parameter verwendet werden. Gib den Inhalt von `a` nach dem Tausch noch einmal aus. Eine kleine Hilfestellung:



```
C:\WINDOWS\system32\cmd.exe
Bitte geben Sie eine Zahl ein: 4
Bitte geben Sie eine Zahl ein: 1
Bitte geben Sie eine Zahl ein: 2
Bitte geben Sie eine Zahl ein: 3
Bitte geben Sie eine Zahl ein: 4

Eingegeben:      1  2  3  4
Addition:       10

Tauschen:                <Werte zwischen 0 und 3 eingeben>
Bitte geben Sie eine Zahl ein: 0
Bitte geben Sie eine Zahl ein: 3

Ergebnis:         4  2  3  1
```

Aufgabe 27 Kontrolle des Programmflusses mit der if-Struktur I (Keyword.java)

Lege eine Klasse `Keyword` an, welche nur aus der `main`-Methode besteht und bestimme für diese Klasse mittels Kommandozeilenparameter mehrere Wörter (z.B. „uni java mensa“).

Der Benutzer muss anschließend einen String eingeben. Dieser wird mit den verschiedenen Wörtern des Kommandozeilenparameters verglichen. Stimmt die Eingabe mit einem dieser Wörter überein, wird „Gefunden!“ ausgegeben. Wurde keines der Wörter eingegeben, gib „Falsches Wort ...“ aus.

Hinweis: Achte darauf, deinen Code in einer übersichtlichen Form zu schreiben und sinnvolle Kommentare einzufügen!

Aufgabe 28 Modifikation der Bottle-Klasse

- a) Nimm die vorhandene Klasse `Bottle` und ergänze sie, so dass in Zukunft die Füllmenge nur zwischen 0.0 und `MAX_FILL` schwanken kann. Um dies zu erreichen, ändere folgende Methoden:

`setFill`

Ändere diese Methode, so dass sie einen Wert vom Typ `boolean` zurückgeben kann. Falls der übergebene Parameter ungültig ist, setze `actFill` auf 0.0 und gib `false` zurück. Ansonsten wird der Wert wie gehabt zugewiesen und der Rückgabewert ist `true`.

`changeFill`

Kniffliger wird es bei dieser Methode. Sie verfügt nun über den Rückgabewert `double`. Eine Änderung von `actFill` um den Parameter `changeFill` findet auf jeden Fall statt. Prüfe allerdings danach aktuellen Inhalt:

Ist er zu groß, wird der Überschuss zurückgegeben. Im Falle eines negativen Wertes wird `actFill` (also die negative Zahl) zurückgegeben. Ein Rückgabewert von 0.0 signalisiert, dass der Parameter korrekt war. Vergiss dabei nicht, vor Aufruf des `return`-Befehls `actFill` auf einen gültigen Wert zu setzen!

Konstruktoren Lösche den Konstruktor mit dem `double`-Parameter, es bleibt nur der Standardkonstruktor erhalten.

Nenne diese erweiterte Klasse `ModBottle`.

- b) Erstelle die Klasse `ModBottleApp`, die nur aus der `main`-Methode besteht. Lege darin als erstes eine `ModBottle`-Objekt mit dem Namen `bottle` an. Rufe anschließend `setFill` auf. Dabei kann der Benutzer über den Inhalt entscheiden. Liefert `setFill` den Wert `false`, wird `bottle` auf `null` gesetzt.

Jetzt wird `changeFill` aufgerufen - falls dies möglich ist. Auch dafür wird er der Benutzer befragt, um noch einen `double`-Wert zu erfahren. Ist es dir gelungen diese Methode in Aufgabe 2 zu ändern, gib den Rückgabewert an den Bildschirm aus.

Füge nun an das Ende der `main`-Methode eine letzte Abfrage an. Sie ist unabhängig von den bisherigen `if`-Verzweigungen. Prüfe darin, ob `bottle` von `null` verschieden ist. Gib dann entweder das Objekt oder „Keine Flasche!“ aus.

Wenn das Programm fertig ist, sollte es etwa so aussehen:



```
C:\WINDOWS\system32\cmd.exe

Den gewünschten Fuellstand eingeben
Bitte geben Sie eine Flie█kommazahl ein: 1.2

Die gewünschte Änderung eingeben
Bitte geben Sie eine Flie█kommazahl ein: 0.9
-> Ueberschuss: 0.6

Ergebnis: Flasche 1.5/1.5

D:\>Pause
Drücken Sie eine beliebige Taste . . .
```



```
C:\WINDOWS\system32\cmd.exe

Den gewünschten Fuellstand eingeben
Bitte geben Sie eine Flie█kommazahl ein: 2.0

Keine Flasche gefunden!

D:\>Pause
Drücken Sie eine beliebige Taste . . .
```

Aufgabe 29 Stein, Schere, Papier

Nachdem das ewige Zahlraten irgendwann langweilig wird, kommt hier ein weiteres Spiel - das allerdings auch schwieriger zu implementieren ist ...

Sowohl der Spieler als auch der Computer denken sich jeweils zufällig einen Zustand „Schere“, „Stein“ oder „Papier“ aus. Danach werden die beiden Werte verglichen. Dabei schlägt ein Stein eine Schere, Papier schlägt den Stein und die Schere zerschneidet das Papier.

Der Gewinner erhält einen Punkt. Falls sich beide dasselbe ausgesucht haben, werden keine Punkte verteilt. Das Spiel endet, wenn ein Spieler 3 Punkte erreicht hat.

Verwende eine `int`-Variable zur Darstellung der drei Zustände. Der Wert des Computers wird zufällig ermittelt (vgl. Blatt 5 / Aufgabe 1), der des Spielers wird solange abgefragt, bis ein gültiger Wert eingegeben wurde.

Informiere danach den Benutzer über die Wahl des Computers und verteile Siegespunkte. Um die Abfrage zur Ermittlung des Runden-Siegers sinnvoll zu gestalten hilft die `switch`-Struktur und folgende Überlegung:

Bilde die Differenz zwischen dem Wert des Spielers und des Computers. Es gibt neun mögliche Ergebnisse. Fällt dir etwas auf?

Ergänze nun das Spiel „Schere, Stein, Papier“ um eine Hilfsklasse, welche die Speicherung der drei Zustände übernimmt. Dies vereinfacht den Ablauf in der `main`-Methode deutlich - für den Benutzer ergibt sich allerdings kein Unterschied.

Nenne diese Klasse `SSP_Value`. Sie verfügt über ein `int`-Attribut `value`, welches den Zustand wie im ursprünglichen Programm speichert und zwei Konstruktoren. Der eine hat einen `int`-Parameter, der den gewünschten Wert für `value` enthält, der andere keinen Parameter. Letzterer liest solange Werte vom Benutzer ein, bis ein gültiger eingegeben wurde und speichert diesen in `value`.

Um aus dieser Klasse nun wirklich einen Nutzen zu ziehen, implementiere eine `toString`-Methode, die den passenden Inhalt als `String` ausgibt (z.B. „Stein“) und eine Methode `defeats`, die als Parameter eine `SSP_Value`-Klasse hat und einen `int`-Wert zurückgibt. Sie vergleicht den `value`-Wert dieser Klasse und des Parameters, um

den Rundensieger zu ermitteln. Der Rückgabewert ist 1 falls diese Klasse gewinnt, -1 falls die Parameterklasse gewinnt und 0 bei Gleichstand.
Schreibe nun das ursprüngliche Programm um, so dass die Speicherung der Zustände dort von dieser neuen Klasse statt int-Variablen übernommen wird.

```

C:\WINDOWS\system32\cmd.exe
Schere <0>, Stein <1> oder Papier <2>? 1
Stein      vs      Papier      COMPUTER gewinnt...
Zwischenstand: 0 : 1
Schere <0>, Stein <1> oder Papier <2>? 3
Schere <0>, Stein <1> oder Papier <2>? 0
Schere      vs      Papier      SPIELER gewinnt!
Zwischenstand: 1 : 1
Schere <0>, Stein <1> oder Papier <2>? 1
Stein      vs      Papier      COMPUTER gewinnt...
Zwischenstand: 1 : 2
Schere <0>, Stein <1> oder Papier <2>? 2
Papier      vs      Stein      SPIELER gewinnt!
Zwischenstand: 2 : 2
Schere <0>, Stein <1> oder Papier <2>? 1
Stein      vs      Schere      SPIELER gewinnt!
Zwischenstand: 3 : 2
D:\>Pause
Drücken Sie eine beliebige Taste . . . _

```

Aufgabe 30 Fakultät berechnen

Schreibe ein Programm, das die Fakultät einer Zahl berechnet und am Bildschirm ausgibt. Die Definition der Fakultät einer nicht-negativen ganzen Zahl n lautet

$$n! = n * (n - 1) * (n - 2) * \dots * 2 * 1$$

Dabei kann der Benutzer diese Zahl entweder als Kommandozeilenparameter oder über eine direkte Eingabe bestimmen.

Prüfe zuerst den Kommandozeilenparameter durch Umwandeln des ersten Elementes mittels `Integer.parseInt()`. Nur falls dieses Element nicht existiert (also kein Kommandozeilenparameter übergeben wurde), wird der Benutzer dazu aufgefordert eine Zahl einzugeben. Wiederhole diese Eingabe so lange, bis eine nicht-negative Zahl eingegeben wurde.

Einen Algorithmus zur Berechnung der Fakultät mittels `for`-Schleife findest du in der Vorlesung. Verwende deshalb hier eine `while`-Schleife.

Aufgabe 31 Clock und DigitalClock

Schreibe eine Klasse `Clock`. In ihr soll eine Uhrzeit gespeichert werden.

Dazu verfügt sie über zwei `int`-Attribute (Stunden und Minuten) und über einen Konstruktor, welcher eben diese Variablen bestimmt.

An weiteren Methoden besitzt sie `toString`, welche die Zeit ausgibt (siehe Beispiel) und `formatInt`. Letztere dient dazu, einen `int`-Wert in einen String umzuwandeln. Falls der Zahlenwert einstellig ist hängt sie zudem eine führende „0“ an (Beispielsweise wird 5 in „05“ umgewandelt, während 58 als „58“ zurückgegeben wird).

Dabei sollen die Attribute und die `formatInt` nicht von anderen Klassen benutzt werden können.

Schreibe nun eine Klasse `DigitalClock` die von `Clock` abgeleitet ist. Sie verfügt über einen Konstruktor, der

wie derjenige der Clock-Klasse funktioniert. Überlade diesen ersten Kontruktor dann mit einem zweiten, der als Parameter ein Clock-Objekt erwartet um dessen Stunden- und Minuten-Werte zu übernehmen.

Überschreibe anschließend die toString der Clock, wobei dieses Mal die Zeit in der Form [hh:mm] ausgegeben wird (nutze dazu die formatInt der Clock. Wie muss diese dann deklariert werden?). Um diese beiden Klassen anzuwenden, schreibe eine dritte Klasse. Erschaffe in ihr eine Clock-Variable, belege sie mit einem Clock-Objekt und gib sie an den Bildschirm aus. Lege anschließend ein DigitalClock-Objekt in dieselbe Variable (möglich wegen Polymorphie) und gib nun dieses Ergebnis aus.

Welche toString wurden jeweils verwendet?



```
C:\WINDOWS\system32\cmd.exe
09 Stunden und 17 Minuten
09:17
D:\>Pause
Drücken Sie eine beliebige Taste . . . _
```

Aufgabe 32 Exception-Handling: Das Interface Pet

Schreibe die folgenden Klassen so, dass sie sich mit der beiliegenden SeciApp verwenden lassen. Triff dabei sinnvolle Entscheidungen, falls die Anleitung keine genaue Vorlage liefert.

```
1 public class SeciApp {
2     public static void main(String[] args)    { new SeciApp(); }
3
4     public SeciApp() {
5         // "sloth" = "Faultier" - nach Oxford
6         Object[] obj = { new MaleSeci("Franz"),
7                         new FemaleSeci("Doris"),
8                         new Sloth("Henry")    };
9
10        while (true) {
11            // Der Benutzer erhält den Zugriff auf ein zufälliges Objekt
12            OOUtil input = new OOUtil();
13
14            Pet actPet = (Pet)obj[(int)(Math.random() * obj.length)];
15
16            int inp = 0;
17            System.out.println("\n\n--- Aktuelles Haustier:\t\t" + actPet);
18            while (inp < 1 || inp > 3)
19                inp = input.readInt("\nSchlafen, fressen oder spielen?
20                (1/2/3)\t");
21
22            try {
23                if (inp == 1)    actPet.sleep();
24                else if (inp == 2) actPet.feed();
25                else            actPet.play();
26            }
27            catch (HungerException e)    { System.out.println(e.getMessage()
28                ); }
29            catch (SleepException e)    { System.out.println(e.getMessage()
30                ); }
31            catch (NoHungerException e) { System.out.println(e.getMessage()
32                ); }
33            catch (NoSleepException e) { System.out.println(e.getMessage()
34                ); }
35        }
36    }
37 }
```

Das Interface `Pet` fordert die drei parameterlosen Methoden `sleep`, `feed` und `play` jeweils ohne Rückgabewert.

Um gefüttert zu werden, muss das Haustier hungrig sein, müde um sich schlafen zu legen und darf nicht hungrig bzw. müde sein, um mit ihm zu spielen. Stelle 4 `Exception`-Klassen bereit, die ausgelöst werden, falls eine dieser Voraussetzungen missachtet wird. Überschreibe dabei lediglich die Methode `getMessage`.

Schreibe eine abstrakte Klasse `Secigotchi`, welche das Interface `Pet` einbindet. Ein `Secigotchi` hat einen Namen und eine Variable für den Hunger. Er beträgt bei der Konstruktion 2 und nimmt ab, falls es gefüttert wird.

Leite von `Secigotchi` eine Klasse für ein weibliches und eine für ein männliches `Secigotchi` ab. Sie verfügen zusätzlich über ein Attribut für Kreativität bzw. Logik. Schlafen sie, so steigt der Wert für das jeweilige Attribut, kann aber nicht größer als eine bestimmte Obergrenze werden. Spielt ein solches `Secigotchi`, so steigt der Hunger und im Gegenzug sinkt das spezielle Attribut.

Ein weibliches `Secigotchi` spielt am liebsten „Schere, Stein, Papier“, ein männliches gerne „Zahlenraten“ (verwende den Code aus den vorherigen Übungen).

Schließlich gibt es noch ein anderes Haustier: Das `Faultier`.

Die entsprechende Klasse ist unabhängig vom `Secigotchi`, bindet aber ebenfalls das Interface `Pet` ein. Ein `Faultier` ist ständig hungrig und müde, will aber nie spielen.

Aufgabe 33 Mehrdimensionale Arrays: Das Pascal'sche Schema

Zur Berechnung der Koeffizienten eines Polynoms der Form $(a - b)^n$ kann man das sog. Pascal'sche Dreieck verwenden:

Ordnung $n = 0$					1						
	1				1		1				
	2			1		2		1			
	3		1		3		3		1		
	4		1		4		6		4		1
	5	1		5		10		10		5	
...											

Dabei werden zwei nebeneinander liegende Zahlen addiert, um den Wert der darunter liegenden zu erhalten.

Schreibe ein Programm, welches dieses Schema einer beliebigen Ordnung mit möglichst geringem Speicheraufwand berechnet und ausgibt.

Aufgabe 34 Tafel (Blackboard)

1. Erstelle eine Klasse `Blackboard`, welche über drei Attribut verfügt:

- `lettering` ist vom Typ `String` und enthält später das, was auf der Tafel steht.
- `bbCounter` und `bbId` sind vom Typ `int`
- `bbCounter` ist statisch (Schlüsselwort `static` steht vor `int`) und wird mit 0 initialisiert, die anderen Attribute bekommen noch keine Werte und werden nur deklariert. `bbCounter` zählt, wie viele Objekte tatsächlich schon erzeugt wurden und jedes `Tafel`-Objekt hat ihre persönliche Nummer, die in `bbId` gespeichert ist.

2. Die Klasse hat zwei Konstruktoren:

Im Standardkonstruktor (ohne Parameter) wird `bbId` der Wert von `bbCounter + 1` zugewiesen und anschließend `bbCounter` um 1 erhöht. `lettering` wurde noch nicht näher bestimmt und soll jetzt auf einen leeren `String` verweisen.

Der zweite Konstruktor hat einen Parameter vom Typ `String` und funktioniert ansonsten genauso wie der Standardkonstruktor, nur dass `lettering` nun auf den `String` verweist, der an den Konstruktor übergeben wurde.

3. Erstelle nun eine zweite Klasse `Board_Tester`, die nur die `main`-Methode besitzt und in dieser zwei `Blackboard`-Objekte erzeugt. Einmal ohne Parameter und einmal mit einem dem `String`, den der Benutzer eingeben kann.
Die nachfolgenden Methoden können (nachdem sie geschrieben wurden) mir Hilfe der erzeugten Objekte in der Klasse `Board_Tester` ausprobiert werden.
(Es sollte auch hier mit Benutzereingaben gearbeitet werden, anfangs können aber auch zu Testzwecken direkt in den Code Strings geschrieben werden, um `lettering` zu füllen.)
4. Schreibe nun eine Methode `write`, die nichts zurück gibt, aber einen `String` als Parameter erwartet. Dieser übergebene `String` wird nun an `lettering` angehängt und damit auf die Tafel geschrieben.
5. Eine weitere Methode `delAll`, ohne Parameter und Rückgabewert soll den Inhalt von `lettering` löschen. (Tipp: `lettering` auf einen Leerstring setzen)
6. Füge nun eine Methode `toString` hinzu, die einen `String` zurückgibt, über das, was aktuell auf der Tafel steht. Die `bbId` ist für jedes Tafel-Objekt eindeutig, daher ist es übersichtlicher, wenn auch sie mit ausgegeben wird.
z.B. `System.out.println("Auf Tafel " +bbId+ " steht: " +lettering);`
7. Erweitere die Methode `toString` so, dass es einen anderen Text ausgibt, wenn die Tafel leer ist (also `lettering` auf einen Leerstring verweist). Hinweis: Die Länge eines Strings wird bei Aufruf der Methode `length()` der Klasse `String` zurückgegeben.
8. Um den Inhalt einer Tafel auf die andere zu schreiben, können zwei Methoden nützlich sein: `overrideAll` und `addAll`. Diese sollen jeweils ein `Blackboard`-Objekt als Parameter bekommen und nichts zurückgeben. In der ersten Methode wird der Inhalt von `lettering` komplett durch das `lettering` des übergebenen `Blackboard`-Objektes ersetzt, bei der zweiten Methode wird es lediglich angehängt.
9. Eine weitere Methode soll `delBackTo` heißen und `i` Zeichen (diese Anzahl wird als Parameter an die Methode übergeben) vom Ende von `lettering` her löschen.
Dafür ist eine Prüfung notwendig, nämlich ob `lettering` länger ist, als `i`. Ist das nicht der Fall, soll `false` zurückgeben werden, bei erfolgreichem Löschvorgang `true`.
Der Vorgang des Löschens findet statt, wenn `lettering` länger ist als `i`. Hier kann mit Hilfe der Methode `substring(int beginIndex, int endIndex)` der Klasse `String` der Teil von `lettering` extrahiert werden, der übrig bleiben soll.
(Der Rückgabewert sollte dann in `Board_Tester` verwendet werden, um dem Benutzer zu zeigen ob sein Vorhaben erfolgreich war oder nicht)

```
--> Objekte werden erzeugt
Bitte geben Sie ein, was auf der 2. Tafel stehen soll: 1 + 1 = 2
Auf der Tafel 1 steht nichts
Auf der Tafel 2 steht:
    1 + 1 = 2
Bitte geben Sie ein, was auf der 1. Tafel stehen soll: Das ist falsch!
Auf der Tafel 1 steht:
    Das ist falsch!
--> Tafel 1 wird gelöscht
Auf der Tafel 1 steht nichts
Bitte geben Sie ein, was auf der 1. Tafel stehen soll: Das ist richtig!
Auf der Tafel 1 steht:
    Das ist richtig!

Tafel 1 wird (inhaltlich) auf Tafel 2 kopiert
Auf der Tafel 1 steht:
    Das ist richtig!
Auf der Tafel 2 steht:
    Das ist richtig!
```

```

Die Tafeln werden geloescht und neu beschrieben
Bitte geben Sie ein, was auf der 1. Tafel stehen soll: Maria geht
Bitte geben Sie ein, was auf der 2. Tafel stehen soll: im Wald spazieren und...
--> Tafel 2 wird (inhaltlich) an Tafel 1 angehängt
Auf der Tafel 1 steht:
    Maria geht im Wald spazieren und...
Auf der Tafel 2 steht:
    im Wald spazieren und...
Bitte geben Sie an, wie viele Zeichen von Tafel 1 gelöscht werden sollen : 7
--> Es wird versucht die letzten 7 Zeichen von Tafel 1 zu löschen
...Erledigt
Auf der Tafel 1 steht:
    Maria geht im Wald spazieren

```

Aufgabe 35 StringStorage

Schreibe eine Klasse namens `StringStorage`. Diese Klasse soll eine gewisse Anzahl Wörter speichern, die der Benutzer eingegeben hat und diese auch wieder ausgeben. Um dies umzusetzen, verfügt die Klasse als Attribut über ein `String`-Array, welches drei Elemente aufnehmen kann und von außerhalb der Klasse nicht sichtbar ist.

Die Klasse verfügt darüber hinaus auch über die Methode `addString`, welche einen übergebenen `String` zum Array hinzufügt. Dies kann aber nur geschehen, falls in dem Array noch noch Platz ist. Je nachdem ob dies möglich war oder nicht, wird `true` oder `false` zurückgegeben.

Des Weiteren verfügt sie über die Methode `toString` (vgl `Bottle`-Klasse). Der Rückgabewert ist einfach die Aneinanderreihung aller `String`s im Array und dazwischen jeweils ein Leerzeichen (Tipp: `+ " "`).

Vorsicht: Ist der Platz noch leer, sollte natürlich nichts zu dem `String` hinzugefügt werden - also insbesondere nicht `null`.

Schreibe nun eine Testanwendung für die oben erstellte Klasse. Diese soll `StorageApp` heißen und nur aus der `main`-Methode bestehen. Darin soll folgendes geschehen:

Zuerst wird ein `StringStorage`-Objekt erstellt. Danach wird der Benutzer wiederholt dazu aufgefordert Wörter einzugeben, die als `String` eingelesen werden und sofort nach der Eingabe im `StringStorage`-Objekt gespeichert werden. Nach jeder Benutzereingabe wird der Inhalt des Objektes an den Bildschirm ausgegeben. Die Eingabe endet erst, wenn das `Storage`-Objekt voll ist.

```

Bitte geben Sie einen String ein: Hallo
-> Hallo
Bitte geben Sie einen String ein: Welt
-> Hallo Welt
Bitte geben Sie einen String ein: !
-> Hallo Welt !
Bitte geben Sie einen String ein: Das
-> Kein Platz mehr!

```

Aufgabe 36 Ein GUI-Beispiel

Es liegt die Klasse `PaintView` bei, welche von `JFrame` abgeleitet wurde.

```

1  /*
2  *  Dieses Programm dient als Beispiel für eine einfache Oberfläche.
3  *  Der Code sollte grundlegend verstanden werden. Dazu empfiehlt es sich
   einige
4  *  Klassen und Interfaces in der API-Hilfe nachzuschlagen
5  */

```

```

6
7 import java.awt.*;
8 import java.awt.event.*;
9 import javax.swing.*;
10
11 public class PaintView extends JFrame
12                        implements ActionListener,
13                        ItemListener,
14                        WindowListener {
15     public static void main(String[] arg) { new PaintView(); }
16
17     private static final Color[] COLOR = { Color.RED, Color.BLUE,
18     Color.GREEN };
19     private static final String[] COLOR_NAMES = { "rot", "blau", "grün"
20     };
21     private static final Shape[] SHAPE = { new Triangle(), new
22     Square(), new Hourglass() };
23     private Checkbox[] chbColor = new Checkbox[COLOR.length];
24     private Checkbox[] chbShape = new Checkbox[SHAPE.length];
25     private JButton butPaint = new JButton("Zeichnen");
26     private Shape actShape = new Shape();
27
28     public PaintView() {
29         super("Zeichenprogramm");
30         getContentPane().setLayout(new BorderLayout());
31
32         // Die Checkboxes erstellen und jeweils in ein Panel packen
33         // Dabei sind die Checkbox-Gruppen nötig, um die
34         // Zusammengehörigkeit zu regeln
35         JPanel colorPanel = new JPanel(new GridLayout(COLOR_NAMES.length,
36         1));
37         CheckboxGroup colorGroup = new CheckboxGroup();
38         for (int c = 0; c < chbColor.length; c++) {
39             chbColor[c] = new Checkbox(COLOR_NAMES[c], colorGroup, false);
40             chbColor[c].addItemListener(this);
41             colorPanel.add(chbColor[c]);
42         }
43         JPanel shapePanel = new JPanel(new GridLayout(SHAPE.length, 1));
44         CheckboxGroup shapeGroup = new CheckboxGroup();
45         for (int c = 0; c < chbShape.length; c++) {
46             chbShape[c] = new Checkbox(
47                 "" + (SHAPE[c] == null ? "Nichts" : SHAPE[c].toString()),
48                 shapeGroup, false);
49             chbShape[c].addItemListener(this);
50             shapePanel.add(chbShape[c]);
51         }
52
53         // Den Button in ein eigenes Panel einschließen, um gewöhnliche Größe
54         // zu erreichen
55         JPanel butPanel = new JPanel();
56         butPanel.add(butPaint);
57         // Die Checkboxes und den Button links (west) platzieren
58         JPanel westPanel = new JPanel(new GridLayout(2, 1));
59         westPanel.add(colorPanel);
60         westPanel.add(shapePanel);
61         westPanel.add(butPanel);
62         getContentPane().add(westPanel, BorderLayout.WEST);

```

```

56     // Abschließende Befehle
57     butPaint.addActionListener(this);
58     chbColor[0].setState(true);
59     chbShape[0].setState(true);
60     itemStateChanged(null);
61     setSize(350, 200);
62     setVisible(true);
63 }
64
65 // Diese Methode wird immer dann aufgerufen, wenn das Fenster neu
66 // gezeichnet werden muss
67 public void paint(Graphics g) {
68     g.setColor(getBackground());
69     actShape.paint(g);
70 }
71 // Wird vom Interface "ActionListener" gefordert
72 public void actionPerformed(ActionEvent event) {
73     // Prüfen, welches Event diesen Aufruf ausgelöst hat (in diesem
74     // Beispiel eigentlich überflüssig)
75     if (event.getSource() == butPaint) {
76         // Form zeichnen. Dazu wird ein Aufruf der paint() erzwungen
77         paint(getGraphics());
78     }
79 }
80 // Wird von Interface "ItemListener" gefordert
81 public void itemStateChanged(ItemEvent event) {
82     // Diese Methode wird aufgerufen, falls eine Checkbox verändert
83     // wurde
84     for (int c = 0; c < chbShape.length; c++)
85         if (chbShape[c].getState() && actShape.getClass() != SHAPE[c].
86             getClass()) {
87             actShape = SHAPE[c];
88             actShape.createShape( (int) chbShape[0].getLocation().getX() +
89                 210,
90                 (int) chbShape[0].getLocation().getY() +
91                 60 );
92         }
93     for (int c = 0; c < chbColor.length; c++)
94         if (chbColor[c].getState()) actShape.setColor(COLOR[c]);
95 }
96 // Wird vom Interface "WindowListener" gefordert
97 public void windowActivated(WindowEvent event)    {}
98 public void windowClosed(WindowEvent event)     {}
99 public void windowClosing(WindowEvent event)    { System.exit(0);
100 }
101 public void windowDeactivated(WindowEvent event) {}
102 public void windowDeiconified(WindowEvent event) {}
103 public void windowIconified(WindowEvent event)  {}
104 public void windowOpened(WindowEvent event)     {}
105 }

```

Versuche dabei das Grundkonzept dieses Beispiels zu verstehen, es wird in der Besprechung genauer erläutert.

Aufgabe 37 Studienbegleitende Leistung WS 2005/ 2006

Aufgabenstellung

Ein Secigotchi ist ein dem Tamagotchi ähnliches Wesen aus der Welt der Vorlesung Objektorientierte Programmierung am Lehrstuhl Management der Informationssicherheit.

Schreiben Sie eine nicht-instancierbare Klasse Secigotchi. Von dieser werden 2 Klassen abgeleitet: eine für ein männliches Secigotchi und eine für ein weibliches Secigotchi.

Allen Secigotchis gemeinsam sind folgende Attribute: Name, Alter, Gewicht, Hunger, Gesundheit und eine Glückszahl. Darüber hinaus besitzen alle eine Liste von Freunden (realisiert als ein Array von Secigotchis). Weibliche Secigotchis haben zusätzlich das Attribut Kreativität, die männlichen besitzen das Attribut logisches Denken. Definieren Sie für die jeweiligen Attribute geeignete Datentypen. Beachten Sie dabei, dass es auch Attribute gibt, die ihre Werte nicht verändern können (z.B. Glückszahl und Name) und wählen Sie hierfür geeignete Modifikatoren.

Alle Secigotchis können:

- Essen
 - Ab einem bestimmten Wert, hat es keinen Hunger mehr und gibt dies dem Benutzer zu verstehen.
 - Andernfalls isst das Secigotchi und freut sich über das Essen. Der Hunger nimmt dabei ab und das Gewicht zu.
- Spielen
 - Männliche Secigotchis spielen immer nur das Spiel Zahlenraten (siehe 3. Übungsblatt), wobei sich der Benutzer eine Zahl ausdenkt und sie raten. Je weniger Versuche ein Secigotchi gebraucht hat, desto stärker wird sein logisches Denken ausgeprägt.
 - Weibliche Secigotchis knobeln gerne. Es gibt Stein, Schere und Papier. Dabei schlägt ein Stein eine Schere, Papier schlägt den Stein und die Schere zerschneidet das Papier. Was das Secigotchi wählt, ist vom Zufall bestimmt. Gewonnen hat man mit insgesamt 3 Siegpunkten, d.h. es sind maximal 5 Spieldurchgänge nötig. Ein Sieg bzw. eine Niederlage des Secigotchis wirkt sich auf die Kreativität aus.
- Sport treiben
 - Wenn das Secigotchi krank ist, d.h. ab einem bestimmten Schwellenwert, kann es keinen Sport treiben.
 - Beim Sport kann sich das Secigotchi mit einer bestimmten Wahrscheinlichkeit verletzen (Gesundheit sinkt erheblich). Ob es sich verletzt, wird dem Zufall überlassen.
- Arzt aufsuchen
 - Der Arztbesuch lässt die Gesundheit steigen.
- Schlafen
 - Der Schlaf erhöht das Alter und den Hunger und erniedrigt das Gewicht.
 - Falls eine bestimmte Zufallszahl der Glückszahl des Secigotchis entspricht, steigt die Gesundheit erheblich, weil es so schön geträumt hat.
- Status anzeigen
 - Alle aktuellen Eigenschaften des Secigotchis werden ausgegeben. Beachten Sie dabei, dass manche Eigenschaften alle Secigotchis besitzen und manche geschlechtsspezifisch sind.

Bei der Geburt eines Secigotchis soll es 2 Möglichkeiten geben. Zum einen sollen bis auf den Namen alle Attribute zufällig festgelegt werden und zum anderen kann der Benutzer konkrete Werte eingeben. Nach der Geburt soll jeweils ausgegeben werden, ob es männlich oder weiblich ist. Das Secigotchi fällt ins Koma, wenn gewisse Werte unter- bzw. überschritten werden.

Schreiben Sie je eine Methode, die es dem Secigotchi ermöglicht ein anderes Secigotchi in seine Liste von Freunden aufzunehmen und auch wieder zu verlieren. Eine Verwendung der Methoden brauchen Sie nicht vorzunehmen. Ein Secigotchi kann maximal eine fest vorgegebene Anzahl (z.B. 3) an Freunden haben.

Schreiben Sie eine konkrete Applikation, die als Kommandozeilenparameter die gesamte Anzahl lebensfähiger Secigotchis erwartet. Die Applikation besteht insgesamt aus mehreren Runden. Wie viele es sind, kann der Benutzer entscheiden. In einer Runde wird mit allen Secigotchis nacheinander gespielt. Eine Spielesession besteht aus einer Statusanzeige und einer individuell festgelegten Aktion, die der Benutzer mit Hilfe der Tastatur

eingeben kann. Sie endet mit dem Schlafen des Secigotchis bis zur nächsten Session.

Achten Sie immer auf eine verständliche Benutzerführung und dokumentieren Sie Ihre selbst erstellten Klassen mit Hilfe von javadoc-Kommentaren. Rechnen Sie auch mit Fehlern bzw. Falscheingaben des Benutzers.

Hinweise zur Bearbeitung

Zur Erzeugung der Zufallszahlen sollte die Klasse `java.util.Random` verwendet werden und muss vor der Klassendefinition, d.h. ganz am Anfang mit `import java.util.*;` eingebunden werden. Die Klasse `OOUtil` dürfen Sie in der neuesten Version im vollen Umfang einsetzen.

Bitte versuchen Sie die Aufgabenstellung so exakt wie möglich umzusetzen und auch eine entsprechende Version abzugeben. Erweiterungen, wie z.B. grafische Repräsentation des Secigotchis, sind zu begrüßen. Allerdings müssen solche erweiterten Lösungen getrennt abgegeben werden. Lustige und einfallsreiche Benutzerausgaben der Secigotchis erhöhen zwar nicht die Anzahl der Punkte, sorgen aber immer für gute Stimmung bei den Korrektoren.

Aufgabe 38 Studienbegleitende Leistung WS 2006/ 2007

Aufgabenstellung

Dir ist über Nacht eine einzigartige Idee eingefallen: Wie wäre es mit einem Plattform, in der sich Studenten eintragen und vorstellen können. Um das ganze zu testen, willst du zuerst einen „Prototypen“ in Java programmieren ...

Erstelle ein Programm, in welchem man mehrere Studenten speichern und ihre Einträge verwalten bzw. ändern kann. Verwende dazu 3 Klassen:

Student

Die Klasse `Student` beinhaltet alle Informationen eines einzelnen Studenten. Außerdem stellt sie Operationen wie z.B. die Änderung des Wohnortes bereit. Dabei hat sie Attribute für Name, Wohnort, Alter und Gruppenzugehörigkeit. Letztere wird mittels eines boolean-Arrays umgesetzt, wobei ein `true` anzeigt, dass der Student der jeweiligen Gruppe angehört. Der Konstruktor übergibt die Parameter für Name und Alter an die passenden Attribute und setzt die beiden übrigen Attribute auf Standardwerte.

Weiter sind `get`-Methoden für alle Attribute und `change`-Methoden für Adresse und die Gruppen vorhanden. Es wird für die Adresse der neue String gespeichert, bei den Gruppen das jeweilige Element auf den umgekehrten Wert gesetzt.

```

C:\WINDOWS\system32\cmd.exe
----- Hauptmenue -----
Pos 0 Markus           Alter 21
Pos 1 Daniel          Alter 20
Pos 2 Tobias           Alter 22
Pos 3 [Leer]
Pos 4 [Leer]

      1 Position auswahlen      2 Student suchen
      3 Liste sortieren        0 Programme beenden

Auswahl      1
Position     0

----- Untermenue -----

      Markus <21> ist in keiner Gruppe Mitglied

      1 Wohnort aendern      2 Gruppen aendern
      3 Student loeschen    0 Zurueck

Auswahl      1
Adresse     Regensburg

----- Untermenue -----

      Markus <21>, wohnt in Regensburg und ist in keiner Gruppe Mitglied

      1 Wohnort aendern      2 Gruppen aendern
      3 Student loeschen    0 Zurueck

Auswahl      2

      1 "Java-Fans"          2 "Gluehwein trotz Regen"
      3 "Weihnachts-Deko-Fanatiker"

Auswahl      1

----- Untermenue -----

      Markus <21>, wohnt in Regensburg und ist Mitglied in
      "Java-Fans"

      1 Wohnort aendern      2 Gruppen aendern

```

Um Informationen über das Objekt bereitzustellen, verwende die toString-Methode für eine kurze Info (Hauptmenü) und eine ähnliche Methode für genauere Informationen (Submenü). Orientiere dich dabei stark an den Abbildungen!

Zum Vergleich mit anderen Student-Objekten existiert die Methode equals. Sie gibt true zurück, falls die Objekte im Bezug auf Name und Alter übereinstimmen. Weiter gibt es eine Methode zum Vergleich nach Alter (true, falls der übergebene Student älter ist als dieser) und eine zum Vergleich nach dem Namen (true, falls dieser Student früher im Alphabet auftaucht als der übergebene - verwende dazu `String.compareTo()`).

Data

Diese Klasse hat als Attribut das Student[]-Objekt und verwaltet den Zugriff auf dieses Array. Dabei werden leere Plätze mit null gekennzeichnet.

- Konstruktor Erschafft das Student[]-Attribut und setzt alle Einträge auf null. Die Größe des Arrays wird mittels eines Parameters bestimmt.
- get Liefert Element an einer bestimmten Stelle im Array. Ist diese Position ungültig, wird null zurückgegeben.
- add Nimmt den Studenten in das Array auf. Falls der Student schon in dem Array aufgenommen ist (verwende equals der Studenten-Klasse) oder das Array voll ist, wird

	keine Operation durchgeführt und false zurückgegeben.
isIndex	Prüft, ob ein übergebener int-Wert ein gültiger Index auf das Array ist.
delete	Löscht die Position im Array und gibt das Objekt, welches auf diesem Platz war, zurück
	sortAge Sortiert das Array nach Alter. Verwende dazu die Bubble-Sort aus der Vorlesung und eine passende Vergleichs-Methode der Studenten-Klasse.
sortName	Sortiert das Array nach Name (wie oben)
toString	Gibt eine Übersicht der Datenbank zurück

VzApp

Die Klasse VzApp ist die übergeordnete Klasse und kümmert sich um die Kommunikation mit dem Benutzer. Dazu fängt sie Eingaben ab und gewährleistet eine gewisse Menüstruktur. Sie besitzt ein Attribut der Data-Klasse und gibt die Befehle des Anwenders an diese weiter.

Die Eingabe des Benutzers wird dabei als int-Wert realisiert, der entsprechend ausgewertet wird. Bei der Eingabe eines ungültigen Wertes wird die Eingabe solange wiederholt, bis ein korrekter Wert eingegeben wurde. Dabei soll folgende Menü-Struktur umgesetzt werden:

Hauptmenü	-> Ausgabe einer groben Übersicht der Datenbank	
1	Position auswählen	
	[Position leer]	Neuen Student anlegen
	[Position belegt]	Submenü
2	Student suchen	
	[Gefunden]	diese Position auswählen (wie 1)
	[Nicht gefunden]	keine Aktion
3	Liste sortieren	
	1 Nach Alter	
	2 Nach Name	
0	Programm beenden	

Wählt der Benutzer „1“, so kann er eine Position aus der Datenbank direkt auswählen. Ist diese leer, so wird er aufgefordert einen neuen Studenten zu erstellen. War sie besetzt, so wird ein Submenü aufgerufen, welches Operationen für diesen Studenten bereitstellt.

Mittels „Student suchen“ kann der Benutzer den Namen und das Alter eines Studenten eingeben. Anhand dieser Angaben wird ein entsprechender Eintrag in der Datenbank gesucht. Es wird nur der erste Treffer berücksichtigt und in diesem Fall wird das Submenü aufgerufen.

Submenü	-> Ausführliche Info über den Studenten	
1	Wohnort eintragen	
2	Gruppenzugehörigkeit ändern	
	1 „“-Gruppe	2 „“-Gruppe
	3 „“-Gruppe	4 „“-Gruppe
3	Student löschen	
0	Zurück	

Dieses Submenü wird in eine entsprechende Methode ausgelagert. Im Gegensatz zum Hauptmenü wird hier nicht die komplette Datenbank ausgegeben, sondern nur detaillierte Informationen zu einem Studenten. Dabei soll die VzApp-Klasse gewährleisten, dass dieses Menü nur erreicht werden kann, falls im Data-Objekt ein gültiger Student ausgewählt wurde.

Das Programm bleibt solange in diesem Untermenü bis der Benutzer den „Zurück“-Befehl nutzt.

```
C:\WINDOWS\system32\cmd.exe

----- Hauptmenue -----
Pos 0 Markus      Alter 21
Pos 1 Daniel      Alter 20
Pos 2 Tobias       Alter 22
Pos 3 [Leer]
Pos 4 [Leer]

      1 Position auswahlen      2 Student suchen
      3 Liste sortieren        0 Programme beenden

Auswahl      1
Position      3
Name          Daniela
Alter        23
-> Neuer Student erfolgreich erstellt

----- Hauptmenue -----
Pos 0 Markus      Alter 21
Pos 1 Daniel      Alter 20
Pos 2 Tobias       Alter 22
Pos 3 Daniela     Alter 23
Pos 4 [Leer]

      1 Position auswahlen      2 Student suchen
      3 Liste sortieren        0 Programme beenden

Auswahl      3
Auswahl      1 Nach Name      2 Nach Alter
Auswahl      2

----- Hauptmenue -----
Pos 0 Daniel      Alter 20
Pos 1 Markus      Alter 21
Pos 2 Tobias       Alter 22
Pos 3 Daniela     Alter 23
Pos 4 [Leer]
```