Programmiersprache Java – Aufgaben

Unter den Aufgaben befinden sich ab Kapitel 8 auch mehrere größere Programme. Im Normallfall würden wir ein Teil dieser größeren Aufgaben als gemeinsame Übung abhalten. Sie sollen zeigen, dass Sie alle mit dem erlernten Wissen und einem Schritt-für-Schritt Vorgehen auch ohne Vorkenntnisse solche herausfordernden Aufgaben umsetzen können. Unter den aktuellen Umständen sind sie in erster Linie für die Studenten gedacht, die entweder etwas Vorwissen mitbringen, und damit durch die normalen Aufgaben gelangweilt sind – oder für die Studenten, die ein bisschen Ehrgeiz mitbringen. Die entsprechenden Aufgaben sind hier als „optional“ gekennzeichnet.

3 Einführung

3.1 Aufgabe „Hallo Welt“

Installieren Sie auf Ihrem Computer das JDK und Ihre Entwicklungs-Umgebung, und bringen Sie sie zum Laufen.

3.2 Aufgaben „Skript“

Bringen Sie die Beispiele aus dem Kapitel „Mini-Einführung“ bei Ihnen zum Laufen.

4 Praxis

4.1 Aufgabe „Hallo Welt“

Schreiben Sie eine erste Klasse ohne Package mit Main-Funktion, in der „Hallo Welt“ auf der Kommandozeile ausgegeben wird.
Entwickeln und testen Sie das Programm mit einem Editor, dem Java Compiler „javac“, und mit der JVM „java“ in einer Kommandozeile.

4.2 Aufgabe „GUI-Fenster“

Schreiben Sie eine Klasse ohne Package mit Main-Funktion, in der ein GUI Fenster geöffnet wird. Entwickeln und testen Sie das Programm mit einem Editor, dem Java Compiler „javac“, und mit der JVM „java“ in einer Kommandozeile.

4.3 Aufgabe „Package“

Schreiben Sie eine Klasse **in einem Package** mit Main-Funktion, in der etwas Passendes auf der Kommandozeile ausgegeben wird.

1) Entwickeln und testen Sie das Programm mit einem Editor, dem Java Compiler „javac“, und mit der JVM „java“ in einer Kommandozeile.

2) Entwickeln und testen Sie das Programm in Ihrer IDE wie z.B. Eclipse oder NetBeans.

4.4 Aufgabe „Packages“

Schreiben Sie eine Klasse **in einem verschachtelten Package** (d.h. das Package liegt wieder in einem Package) mit Main-Funktion, in der etwas Passendes auf der Kommandozeile ausgegeben wird.

1) Entwickeln und testen Sie das Programm mit einem Editor, dem Java Compiler „javac“, und mit der JVM „java“ in einer Kommandozeile.
2) Entwickeln und testen Sie das Programm in Ihrer IDE wie z.B. Eclipse oder NetBeans.

4.5 Aufgabe „Summe“

Schreiben Sie ein Programm, das zwei Gleitkomma-Zahlen von der Kommandozeile einliest und ihre Summe ausgibt. Fangen sie fehlerhafte Eingaben ab, und geben sie im Falle eines Fehlers eine Meldung aus.

1) Entwickeln und testen Sie das Programm mit einem Editor, dem Java Compiler „javac“, und mit der JVM „java“ in einer Kommandozeile.
2) Entwickeln und testen Sie das Programm in Ihrer IDE wie z.B. Eclipse oder NetBeans.

Dieses Programm hat – bezogen auf unser aktuelles Wissen – einen Knackpunkt: das Einlesen von Kommandozeile ist relativ aufwändig und setzt eine Menge Pragmatismus voraus. Etwas einfacher wäre die Benutzung von Kommandozeilen-Argumenten.

Nun werden viele der noch folgenden Programme so viel Benutzer-Interaktion erfordern, dass Kommandozeilen-Argumente keine Lösung darstellen, aber für diese Aufgabe (und noch einige folgende) wäre dies eine akzeptable Lösung. Und später haben wir ja mehr Erfahrung und mehr Wissen – bis dahin erscheint uns die Eingabe vielleicht gar nicht mehr so mystisch.

Nachteilig an der Kommandozeilen-Argument Lösung ist aber, dass die Änderung der Kommandozeilen-Argumente in Eclipse aufwändiger ist. Für das Entwickeln und Testen in Eclipse wäre das Einlesen von Kommandozeile viel angenehmer und flexibler.

Von daher schlage ich folgenden Kompromiss vor:

- Für den ersten Wurf mit dem Java-Compiler „javac“ und der JVM „java“ implementieren sie die Aufgabe mit Kommandozeilen-Argumenten.
- Und im zweiten Wurf mit Eclipse – und dann ja auch mit viel mehr Erfahrung – lesen Sie die Summanden von der Kommandozeile ein.

4.6 Aufgabe „Ausgabe Verzeichnis“

Compilieren Sie die Programme der fünf vorherigen Aufgaben mit dem Java Compiler „javac“ so, dass der Byte-Code in einem eigenen Verzeichnis liegt. Starten Sie die Programme danach

mit der JVM „java“ in einer Kommandozeile.

5 Typen und Variablen

5.1 Aufgabe „Wert- und Referenz-Semantik“

Schreiben Sie ein kleines Programm, das den Unterschied von Wert-Semantik bei den elementaren Datentypen und Referenz-Semantik bei allen anderen Typen zeigt.

6 Operatoren

6.1 Aufgabe „Inkrement-Operator“

Schreiben Sie ein kleines Programm, das den Unterschied im Verhalten der Prä- und Post-Inkrement Operatoren („++“) verdeutlicht, d.h. wo ein Vertauschen der Operatoren zu einem anderen Ergebnis führt.

7 Kontrollstrukturen

7.1 Aufgabe „Schleifen-Varianten“

Schreiben Sie ein Programm, dass dreimal hintereinander die Zahlen von 1 bis 5 aus gibt. Für die erste „1..5“ Ausgabe soll eine For-Schleife benutzt werden, für die zweite eine While-Schleife, und für die dritte eine Do-Schleife. Jede „1..5“ Ausgabe soll in einer eigenen Zeile stehen, und die Zahlen sollen mit einem Leerzeichen untereinander getrennt sein.

```
Ausgabe
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
```

7.2 Aufgabe „Teilbar?“

Schreiben Sie ein Programm, dass hintereinander die Zahlen von 1 bis 10 und jeweils eine Meldung, ob die Zahl durch 2 bzw. durch 3 teilbar ist, ausgibt. Benutzen Sie, um die Zahlen zu durchlaufen, eine For-Schleife.

```
Mögliche Ausgabe:
1  nicht-durch-2-teilbar  nicht-durch-3-teilbar
2  durch-2-teilbar  nicht-durch-3-teilbar
3  nicht-durch-2-teilbar  durch-3-teilbar
4  durch-2-teilbar  nicht-durch-3-teilbar
5  nicht-durch-2-teilbar  nicht-durch-3-teilbar
...
```

7.3 Aufgabe „Lesbare Zahlen“

Schreiben Sie ein Programm, dass hintereinander die Zahlen von 1 bis 5 als lesbaren Text aus gibt. Benutzen Sie, um die Zahlen zu durchlaufen, eine For-Schleife. Jede Zahl-Ausgabe soll in einer eigenen Zeile stehen.

```
Mögliche Ausgabe:
eins
zwei
drei
vier
fuenf
```

7.4 Aufgabe „Zahlen-Liste“

Schreiben Sie ein Programm, dass hintereinander die Zahlen von 1 bis 5 aus gibt. Die Zahlen sollen dabei durch einen Bindestrich ‚-‘ getrennt werden, und die gesamte Zahlenreihe soll von runden Klammern umgeben sein. Benutzen Sie, um die Zahlen zu durchlaufen, eine For-Schleife.

```
Ausgabe:
(1-2-3-4-5)
```

7.5 Aufgabe „Mittelwert und Varianz“

Schreiben Sie ein Programm, das den Mittelwert und die Varianz von Gleitkomma-Zahlen berechnet. Der Benutzer gibt die Werte auf der Kommandozeile an. Der Mittelwert ist die Summe der Werte geteilt durch die Anzahl. Die Varianz ist die Summe der Quadrate der Werte geteilt durch die um eins reduzierte Anzahl. Fangen sie fehlerhafte Eingaben wieder sinnvoll ab. Denken Sie sich ein sinnvolles Abbruchkriterium aus, d.h. was muss der Benutzer eingeben, wenn er fertig mit der Eingabe ist.

Um auch dieses Programm erstmal etwas einfacher zu gestalten, könnten Sie eine Version schreiben, die die Werte als Kommandozeilen-Argumente übergeben bekommt statt Sie einzulesen.

7.6 Aufgabe „Multiplikations-Matrix“

Schreiben Sie ein Programm, das zwei positive Integer-Zahlen einliest, und für diese (von bis inkl.) das Ein-mal-eins als Matrix ausgibt.

Hinweise:

- Fangen Sie fehlerhafte Eingaben ab. Diesen Hinweis gibt es hier jetzt zum letzten Mal, danach wird er als selbstverständlich vorausgesetzt.
- Wie in früheren Aufgaben ist es für den Einstieg möglich, zuerst eine Version mit Kommandozeilen-Argumenten zu schreiben.

Beispiel:

```
Zahl1: 2
Zahl2: 4

   |  2  3  4
---+------------
 2 |  4  6  8
 3 |  6  9  12
 4 |  8  12  16
```

Hinweis – für eine schönere Formatierung fehlt uns leider noch das Wissen.

8 Funktionen

8.1 Aufgabe „Fakultäts Funktion“

Schreiben Sie zwei Versionen der Fakultäts-Funktion. Die Fakultät ist das Produkt der natürlichen Zahlen von 1 bis n. Mathematisch sieht die Definition folgendermaßen aus:

- fak(0) :== 1
- fak(n) :== n * fak(n-1)

Version 1 soll das Ergebnis iterativ berechnen – d.h. mit einer Schleife.
Version 2 soll das Ergebnis rekursiv berechnen – d.h. durch Aufruf von sich selber.
Welche Version würden Sie in der Praxis warum vorziehen?

Schreiben Sie zusätzlich ein kleines Haupt-Programm, dass beide Funktionen für einige Int-Werte benutzt. Achtung – bedenken Sie, dass die Fakultät sehr schnell wächst, und schon „14!“ nicht mehr in einen vorzeichenbehafteten 32-Bit Integer-Wert paßt, und „21!“ auch eine Long-Wert sprengt.

8.2 Aufgabe „Primzahl?“

Schreiben Sie eine Funktion, die eine positive Integer-Zahl erwartet und zurückgibt ob die Zahl eine Primzahl ist. Schreiben Sie ein Programm, dass diese Funktion nutzt. Das Programm soll sowohl mit einem Kommandozeilen-Argument umgehen können, als auch interaktiv mit dem Benutzer die Zahl einlesen können. Hierbei gilt folgende Strategie:

- Sind Kommandozeilen Argumente vorhanden, so werden Sie alle für den Primzahl-Check genommen.
- Ist kein Kommandozeilen Argument vorhanden, so wird eine Zahl interaktiv vom Benutzer erfragt.

8.3 Aufgabe „Schleife rekursiv“

Nachdem Sie nun Funktionen und Rekursion kennen gelernt haben, schreiben Sie die einfachen Zahlen-Schleife aus Aufgabe 7.1 mit einer rekursiven Funktion statt mit einer Schleife.

```
Ausgabe
1 2 3 4 5
```

8.4 Aufgabe „Zahlen-Liste 2“

Und auch die Zahlen-Liste aus Aufgabe 7.4 läßt sich nun mit Funktionen und Rekursion neu schreiben.

```
Ausgabe:
(1-2-3-4-5)
```

Lösung siehe Kapitel todo.

8.5 Aufgabe „Quadratwurzel“ (optional)

Schreiben Sie ein Programm, das die Quadratwurzel einer Fließkomma-Zahl größer-gleich 1.0 berechnet. Benutzen Sie hierbei nur die Grundrechenarten und Vergleiche, indem Sie sich der Lösung durch Intervall-Schachtelung Schritt für Schritt annähern.

Das Konzept der Intervall-Schachtelung arbeitet folgendermaßen:

- Beginnen Sie mit zwei Zahlen, von denen Sie ausgehen können, dass sie kleiner-gleich bzw. größer-gleich der Quadratwurzel sind.
- Berechnen Sie den Mittelwert der beiden Zahlen, und quadrieren Sie diesen.
- Weicht der quadrierte Mittelwert um weniger als einen Toleranzwert *epsilon* (z.B. 1.0e-5) von der Eingabe-Zahl ab, so verwenden Sie den Mittelwert als Ergebnis der Berechnung. D.h. Sie haben die Quadratwurzel gefunden.
- Ist das Quadrat des Mittelwertes größer als die Eingabe-Zahl, so verwenden Sie das untere Intervall für die nächste Iteration.
- Ist das Quadrat des Mittelwertes kleiner als die Eingabe-Zahl, so verwenden Sie das obere Intervall für die nächste Iteration.
- Führen Sie die Iteration so lange durch, bis sich der quadrierte Mittelwert bis auf den Toleranzwert *epsilon* an die Eingabe-Zahl angenähert hat.

Geben Sie am Anfang der Berechnung den Toleranzwert *epsilon* aus.
Geben Sie vor jeder Iteration die Nummer der Iteration und das aktuelle Intervall aus.
Und geben Sie am Ende natürlich auch die Wurzel aus.

```
Mögliche Ein- und Ausgabe
Wurzel-Berechnung fuer Zahlen groesser-gleich 1.0
Zahl: 4
Berechne Wurzel aus 4 mit einem Epsilon von 0.0001
- Durchlauf  1: 1 < sqrt(4) < 4
- Durchlauf  2: 1 < sqrt(4) < 2.5
- Durchlauf  3: 1.75 < sqrt(4) < 2.5
- Durchlauf  4: 1.75 < sqrt(4) < 2.125
- Durchlauf  5: 1.9375 < sqrt(4) < 2.125
- Durchlauf  6: 1.9375 < sqrt(4) < 2.03125
- Durchlauf  7: 1.98438 < sqrt(4) < 2.03125
- Durchlauf  8: 1.98438 < sqrt(4) < 2.00781
- Durchlauf  9: 1.99609 < sqrt(4) < 2.00781
- Durchlauf 10: 1.99609 < sqrt(4) < 2.00195
- Durchlauf 11: 1.99902 < sqrt(4) < 2.00195
```

```
- Durchlauf 12: 1.99902 < sqrt(4) < 2.00049
- Durchlauf 13: 1.99976 < sqrt(4) < 2.00049
- Durchlauf 14: 1.99976 < sqrt(4) < 2.00012
- Durchlauf 15: 1.99994 < sqrt(4) < 2.00012
- Durchlauf 16: 1.99994 < sqrt(4) < 2.00003
Wurzel: 1.99998
```

Gibt es mehrere Arten, wie sie diese Funktion implementieren können? Wenn ja, dann machen Sie das auch.

9 Bibliotheks-Klassen

9.1 Aufgabe „String-Analyse“

Schreiben Sie ein Programm das eine Zeile einliest, und ausgibt wie oft welches Zeichen in der Zeile vorkommt. Erzeugen Sie eine Ausgabe der Zeichen in der Reihenfolge ihres **ersten** Vorkommens im Eingabetext – weitere Vorkommen der Zeichen werden bei der Ausgabe ignoriert.

```
Mögliche Ein- und Ausgabe:
Eingabe: Hallo Welt: 123332
'H': 1
'a': 1
'l': 3
'o': 1
' ': 2
'W': 1
'e': 1
't': 1
':': 1
'1': 1
'2': 2
'3': 3
```

Implementieren Sie zwei Lösungen:

- Die erste Lösung soll ohne Container auskommen und nur mit den Klassen String, StringBuffer und/oder StringBuilder implementiert werden.
- Im Gegensatz zur ersten Lösung sollen Sie hier Container nutzen – und erreichen damit (hoffentlich) eine viel einfachere Lösung.

9.2 Aufgabe „Hallo <Person>“

Schreiben Sie ein Programm, dass einen kompletten Namen einliest, und diesen Namen mit Sternchen und Leerzeichen umrandet als Anrede ausgibt. Bsp: der eingebene Name ist „Detlef Wilkening“, dann soll folgende Ausgabe erzeugt werden:

```
Mögliche Ein- und Ausgabe:
Name: Detlef Wilkening

**************************
*                        *
* Hallo Detlef Wilkening! *
*                        *
```

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

9.3 Aufgabe „Hallo <Personen>“

Schreiben Sie ein Programm ähnlich Aufgabe 9.2, dass hier aber beliebig viele komplette Namen ein liest. Wird eine leere Eingabe gemacht, so gelten damit alle Namen als eingegeben. Danach sollen alles diese Namen mit Anrede und umrandet mit Sternchen und Leerzeichen ausgegeben werden – die Reihenfolge soll hierbei der Eingabe entsprechen. Die Umrandung soll sich am längsten Namen orientieren. Bsp: die eingegebenen Namen seien „Max“, „Johannes Wolfgang“ und „Werner“, dann soll folgende Ausgabe erzeugt werden:

```
Mögliche Ein- und Ausgabe:
Name: Max
Name: Johannes Wolfgang
Name: Werner
Name:

**************************
*                        *
* Hallo Max!             *
* Hallo Johannes Wolfgang! *
* Hallo Werner!          *
*                        *
**************************
```

9.4 Aufgabe „Lesbare Zahlen 2“

Eine ähnliche Aufgabe wie 7.3: schreiben Sie ein Programm, dass Benutzer-Eingaben von Ziffern von 1 bis 9 als lesbaren Text aus gibt. Die Eingabe einer beliebigen anderen Zahl oder von etwas anderem sollen das Programm beenden.

```
Bitte geben Sie eine Ziffer ein: 2
-> zwei
Bitte geben Sie eine Ziffer ein: 7
-> sieben
Bitte geben Sie eine Ziffer ein: x
Ende
```

Als Unterschied zu Aufgabe 7.3 sollen Sie hier keine Kontrollstruktur (switch oder if) zur Abbildung der Zahlen auf Texte verwenden, sondern einen Container. Machen Sie sich klar, dass diese Lösung kürzer, performanter, und auch noch besser lesbar ist.

9.5 Aufgabe „Lottozahlen“

Schreiben Sie ein Lotto-Programm, d.h. ein Programm dass sechs Zufallszahlen zwischen 1 und 49 inkl. sortiert ausgibt. Aber Achtung – keine Zufallszahl darf mehrfach vorkommen.

9.6 Aufgabe „Telefonbuch“

Schreiben Sie ein erstes ganz ganz einfaches Telefonbuch.

Das Telefonbuch soll Namen und Telefonnummern speichern – am Anfang begnügen wir uns mit einer Nummer pro Name, und es soll kein Name doppelt vorkommen (d.h. jeder Name im Telefonbuch ist ein Unikat). Die Telefon-Nummern sind keine Zahlen, sondern Texte, damit z.B. Leerzeichen oder ein Slash „/“ für eine lesbarere Formatierung möglich sind.

Die Einträge im Telefonbuch sollen im Augenblick noch fest im Programm stehen, d.h. es können im Augenblick keine Einträge hinzugefügt, gelöscht, oder editiert werden.

Der Benutzer kann entweder einen der Befehle „end“ bzw. „list“ oder einen Namen eingeben.

- Bei „end“ soll das Programm beendet werden.
- Bei „list“ sollen alle Einträge im Telefonbuch (zuerst Name, dann Nummer, eine Zeile pro Eintrag – sortiert nach Namen (nicht lexikalisch sortiert, sondern nach Zeichenkodierung) ausgegeben werden.
- Alle anderen Benutzer-Eingaben werden als Namen interpretiert, und im Telefonbuch gesucht. Wird der Name gefunden, so wird die Nummer ausgegeben. Wird der Name nicht gefunden, so wird eine entsprechende Meldung ausgegeben. Achtung – der Name muss exakt übereinstimmen, und es werden auch Groß- und Klein-Schreibung unterschieden.

Nach der Bearbeitung des Befehls „list“ oder eines Namens kann der Benutzer die nächste Eingabe machen.

```
Mögliche Ein- und Ausgabe
Telefonbuch

Eingabe: list

6 Eintraege:
- Bernd => 0555 / 55 55 55
- Detlef => 0123 / 12 21 1
- Dietmar => 0321 / 888 222 99
- Edgar => 0111 / 11 11 1
- Edmund => 0222 / 33 22 11
- Karl => 0111 / 11 22 33

Eingabe: Max

Name "Max" ist nicht im Telefonbuch vorhanden.

Eingabe: Detlef

Detlef => 0123 / 12 21 1

Eingabe: end

Programmende
```

9.6.1 Aufgabe „Platten-Platz Verbrauch“

In der Vorlesung haben wir eine einfache Form der rekursiven Datei-Suche kennengelernt. Nun sollen Sie das Programm etwas erweitern.

- Ihr Programm soll berechnen, wieviel Platten-Platz alle Dateien eines Typs (einer Extension) innerhalb eines Verzeichnisses verbrauchen.
- Zusätzlich soll der Benutzer das Verzeichnis und die Such-Extension (ohne „.“) auf der Kommando-Zeile eingeben können.

- Die Verzeichnis-Eingabe soll sich so oft wiederholen, bis der Benutzer ein gültiges Verzeichnis eingegeben hat.
- Wird keine Extension eingegeben, wird der Platten-Platz Verbrauch aller Dateien im Verzeichnis berechnet.
- Vergleichbar zur rekursiven Datei-Suche in der Vorlesung ignorieren wir Probleme, die sich aus User-Rechten und fehlenden Zugriffs-Rechten resultieren.

Beispiele, unter der Berücksichtigung der Datei-System-Struktur aus der Vorlesung, und entsprechender Datei-Größen:

```
Mögliche Ein- und Ausgabe
Platten-Platz Verbrauch

Geben Sie bitte das Verzeichnis ein: C:\JavaDateiSystemBeispiele
Geben Sie bitte die Extension ein: dat

Gesamtverbrauch: 3472 Bytes
```

```
Mögliche Ein- und Ausgabe
Platten-Platz Verbrauch

Geben Sie bitte das Verzeichnis ein: C:\JavaDateiSystemBeispiele
Geben Sie bitte die Extension ein: txt

Gesamtverbrauch: 5683 Bytes
```

```
Mögliche Ein- und Ausgabe
Platten-Platz Verbrauch

Geben Sie bitte das Verzeichnis ein: C:\JavaDateiSystemBeispiele
Geben Sie bitte die Extension ein:

Gesamtverbrauch: 9155 Bytes
```

```
Mögliche Ein- und Ausgabe
Platten-Platz Verbrauch

Geben Sie bitte das Verzeichnis ein: kein-verzeichnis
"kein-verzeichnis" ist kein Verzeichnis

Geben Sie bitte das Verzeichnis ein: immer noch nicht
"immer noch nicht" ist kein Verzeichnis

Geben Sie bitte das Verzeichnis ein: C:\JavaDateiSystemBeispiele
Geben Sie bitte die Extension ein: dat

Gesamtverbrauch: 3472 Bytes
```

9.7 Aufgabe „Datei-Suche“ (optional)

Die nächste Aufgabe erweitert die einfache Datei-Suche aus der Vorlesung.

- Ihr Programm soll alle Dateien innerhalb eines Verzeichnisses suchen, die einem von mehreren Namen entsprechen.
- Der Benutzer soll das Such-Verzeichnis und die Such-Datei-Namen auf der Kommando-Zeile eingeben können.
- Die Such-Verzeichnis-Eingabe soll sich so oft wiederholen, bis der Benutzer ein gültiges Verzeichnis eingegeben hat.

- Die Such-Datei-Namen-Eingabe soll sich so oft wiederholen, bis der Benutzer einen Leerstring eingibt. Alle Namen bis dahin gelten als zu Suchen.
- Die gefundenen Dateien mit ihren Pfaden sollen sortiert nach den Datei-Namen ausgeben werden. Jeder Ausgabe-Block soll mit dem Datei-Namen beginnen, und danach eine eingerückte Aufzählung der Verzeichnisse (inkl. Datei-Namen).
- Wurde keine Datei zu einem Datei-Such-Namen gefunden, so wird nur der Datei-Such-Name ausgegeben.
- Vergleichbar zur rekursiven Datei-Suche in der Vorlesung ignorieren wir Probleme, die sich aus User-Rechten und fehlenden Zugriffs-Rechten resultieren.

Beispiele, unter der Berücksichtigung der Datei-System-Struktur aus der Vorlesung:

```
Mögliche Ein- und Ausgabe
Datei-Suche
-----------

Geben Sie bitte das Such-Verzeichnis ein: C:\JavaDateiSystemBeispiele
Geben Sie bitte die Such-Namen ein: find.txt
Geben Sie bitte die Such-Namen ein: search.dat
Geben Sie bitte die Such-Namen ein:

find.txt
-> C:\JavaDateiSystemBeispiele\verzeichnis1\verzeichnis3\verzeichnis4\find.txt
-> C:\JavaDateiSystemBeispiele\verzeichnis1\verzeichnis5\find.txt
-> C:\JavaDateiSystemBeispiele\verzeichnis2\find.txt
search.dat
-> C:\JavaDateiSystemBeispiele\search.dat
-> C:\JavaDateiSystemBeispiele\verzeichnis1\verzeichnis3\verzeichnis4\search.dat
```

```
Mögliche Ein- und Ausgabe
Datei-Suche
-----------

Geben Sie bitte das Such-Verzeichnis ein: kein Verzeichnis
"kein Verzeichnis" ist kein Verzeichnis

Geben Sie bitte das Such-Verzeichnis ein: C:\JavaDateiSystemBeispiele
Geben Sie bitte die Such-Namen ein: search.dat
Geben Sie bitte die Such-Namen ein: search.txt
Geben Sie bitte die Such-Namen ein: search.png
Geben Sie bitte die Such-Namen ein:

search.dat
-> C:\JavaDateiSystemBeispiele\search.dat
-> C:\JavaDateiSystemBeispiele\verzeichnis1\verzeichnis3\verzeichnis4\search.dat
search.png
search.txt
```

10 Arrays

10.1 Aufgabe „Lesbare Zahlen 3“

Die gleiche Aufgabe wie vorher: schreiben Sie ein Programm, dass Benutzer-Eingaben von Zahlen von 1 bis 9 als lesbaren Text ausgibt. Die Eingabe einer beliebigen anderen Zahl oder eine Fehleingabe soll das Programm beenden.

```
Bitte geben Sie eine Zahl ein: 2
-> zwei
Bitte geben Sie eine Zahl ein: 7
-> sieben
Bitte geben Sie eine Zahl ein: x
-> Ende
```

Als Unterschied zur letzten Aufgabe sollen Sie hier keinen Container, sondern ein Array verwenden. Machen Sie sich klar, dass diese Lösung vielleicht noch effizienter ist.

11 Klassen

11.1 Aufgabe „Ringzähler“

Schreiben Sie einen Ringzähler. Ein Ringzähler ist ein Zähler, der von einem Startwert an hoch (oder runter) zählt (Delta muss nicht 1 sein) und nach Überschreiten des Endwertes wieder beim Startwert anfängt. Ein solcher Zähler wird in der Praxis häufig gebraucht, z.B. als Index in ein Array, als Verweis auf ein Spielfeld, usw. Wie gehen Sie mit widersprüchlichen Eingaben um?

Beim Erzeugen eines Ringzählers sollte der Nutzer folgende Möglichkeiten haben:

- Angabe nur vom End-Wert => Start-Wert ist 0, Delta ist 1
- Angabe von Start- und End-Wert => Delta ist 1
- Angabe von Start- und End-Wert und Delta

Der Ringzähler sollte z.B. mit folgenden Dingen umgehen können – denken Sie sich eine gute Fehlerbehandlung im Rahmen ihrer Möglichkeiten für problematische Fälle aus:

- End-Wert vor Start-Wert, Delta positiv
- End-Wert nach Start-Wert, Delta negativ
- Delta ist 0

11.2 Aufgabe „Kontaktdaten 1“

Schreiben Sie eine kleine Kontaktdaten-Verwaltung. Folgende Fähigkeiten soll das Programm in der ersten Version haben:

- Menü für folgende Aufgaben:
  - Eingabe neuer Personen
  - Ausgabe aller Personen – unsortiert
  - Ausgabe aller Personen, deren Vor- oder Nach-Name mit einem bestimmten String-Muster anfängt (Groß- und Kleinschreibung wird nicht unterschieden).
- Das Menü soll der Benutzer durch die Eingabe einzelner Buchstaben (mit Return) anwählen können.

Personen bestehen aus:

- Vorname

- Nachname
- Telefon (als String)

Mögliche Ein- und Ausgabe:

```
Bitte waehlen Sie eine Aktion aus
- l : Liste
- n : Neu
- s : Suchen
- e : Ende
> l

5 Kontakte:
- Detlef Wilkening - Tel: 1234
- Dieter Schmidt
- Dietmar Mueller - Tel: 456
- Edgar Wilkens
- Ralf Eberleh - Tel: 555 444

Bitte waehlen Sie eine Aktion aus
- l : Liste
- n : Neu
- s : Suchen
- e : Ende
> n

Geben Sie eine neue Person ein
Vorname: Martina
Nachname: Winkens
Telefon: 98 76 54

Bitte waehlen Sie eine Aktion aus
- l : Liste
- n : Neu
- s : Suchen
- e : Ende
> l

6 Kontakte:
- Detlef Wilkening - Tel: 1234
- Dieter Schmidt
- Dietmar Mueller - Tel: 456
- Edgar Wilkens
- Ralf Eberleh - Tel: 555 444
- Martina Winkens - Tel: 98 76 54

Bitte waehlen Sie eine Aktion aus
- l : Liste
- n : Neu
- s : Suchen
- e : Ende
> s

Geben Sie einen Suchstring ein: e
- Edgar Wilkens
- Ralf Eberleh - Tel: 555 444

Bitte waehlen Sie eine Aktion aus
- l : Liste
- n : Neu
- s : Suchen
- e : Ende
> s

Geben Sie einen Suchstring ein: de
- Detlef Wilkening - Tel: 1234

Bitte waehlen Sie eine Aktion aus
- l : Liste
```

```
- n : Neu
- s : Suchen
- e : Ende
> s

Geben Sie einen Suchstring ein: x
- keinen passenden Eintrag gefunden

Bitte waehlen Sie eine Aktion aus
- l : Liste
- n : Neu
- s : Suchen
- e : Ende
> e

Ende
```

Schauen Sie sich bei der Lösung ruhig noch mal die Aufgabe „Telefonbuch“ an. Aber Achtung – viele Dinge sind hier anders:

- Es gibt ein Menü
- Die Liste aller Personen ist hier unsortiert
- Personen können eingegeben werden
- Das Suchen funktioniert auch mit Anfangs-Mustern ohne Berücksichtigung von Groß- und Klein-Schreibung.
- Und vor allem: jetzt kennen sie Klassen. Und da Sie ja wissen, dass wir die Kontaktdaten Verwaltung später noch aufbohren wollen – sollten Sie sie sauber strukturieren.

11.3 Aufgabe „Kontaktdaten 2“ (optional)

Und damit wir gleich sehen, ob Sie Ihr Programm sauber strukturiert haben, bohren wir es direkt auf: Eine Person soll neben Vor-, Nachname und Telefon auch noch eine Beschreibung (String) enthalten.

Mögliche Ein- und Ausgabe:

```
Bitte waehlen Sie eine Aktion aus
- l : Liste
- n : Neu
- s : Suchen
- e : Ende
> n

Geben Sie eine neue Person ein
Vorname: Detlef
Nachname: Wilkening
Telefon: 1234
Beschreibung: Java Dozent

Bitte waehlen Sie eine Aktion aus
- l : Liste
- n : Neu
- s : Suchen
- e : Ende
> n

Geben Sie eine neue Person ein
Vorname: Bernd
Nachname: Mustermann
Telefon:
```

```
Beschreibung:

Bitte waehlen Sie eine Aktion aus
- l : Liste
- n : Neu
- s : Suchen
- e : Ende
> l

2 Kontakte:
- Detlef Wilkening - Tel: 1234 => Java Dozent
- Bernd Mustermann

Bitte waehlen Sie eine Aktion aus
- l : Liste
- n : Neu
- s : Suchen
- e : Ende
> e

Ende
```

Wenn Sie ihr Programm gut designt haben, dann sollte dies mit sehr wenigen Änderungen am Code machbar sein. Und die Änderungen sollten auch nur sehr lokal sein.

12 Klassen-Details

12.1 Aufgabe „Türme von Hanoi“

Simulieren Sie die „Türme von Hanoi“.

Die „Türme von Hanoi“ ist ein Spiel, bei dem am Anfang alle Scheiben auf einem Stab liegen, und dann zu einem anderen Stab transportiert werden sollen. Für den Transport gelten zwei Regeln:

- Es darf immer nur eine Scheibe bewegt werden.
- Es darf immer nur eine kleinere Scheibe auf einer größeren liegen, nie umgekehrt.

Schreiben Sie ein Programm, dass am Anfang die Anzahl an Scheiben einliest, mit denen die Türme von Hanoi gespielt werden sollen. Danach soll das Spiel mit entsprechender Ausgabe simuliert werden.

```
Mögliche Ein- und Ausgabe
Anzahl Scheiben: 3

1.
   -|-      |        |
  --|--     |        |
 ---|---    |        |
************************

2.
    |       |        |
  --|--     |        |
 ---|---    |      -|-
************************

3.
```

```
    |       |       |
    |       |       |
 ---|---  --|--    -|-
************************

4.
    |       |       |
    |      -|-      |
 ---|---  --|--     |
************************

5.
    |       |       |
    |      -|-      |
    |     --|--  ---|---
************************

6.
    |       |       |
    |       |       |
   -|-    --|--  ---|---
************************

7.
    |       |       |
    |       |     --|--
   -|-      |    ---|---
************************

8.
    |       |      -|-
    |       |     --|--
    |       |    ---|---
************************
```

12.2 Aufgabe „Gerüchteküche“

Kleine Dörfer sind Gerüchteküchen erster Klasse. Jeder kennt jeden, und jeder redet über jeden, bzw. schweigt wissend. All das wollen wir in einem kleinen Programm nachbilden.

Simulieren Sie ein Dorf mit n Einwohnern. Am Anfang kennt nur ein beliebiger Einwohner (nehmen Sie einfach den ersten) das Gerücht.

Danach simulieren Sie das Geschehen, indem Sie maximal m Durchläufe starten. In jedem Durchlauf treffen sich zwei zufällige Einwohner des Dorfes und reden miteinander. Was hierbei passiert, ist davon abhängig, welchen Status bzgl. des Gerüchts die beiden Einwohner haben.

Einwohner 1	Einwohner 2	=>	Einwohner 1	Einwohner 2	Änderung
unwissend	unwissend		unwissend	unwissend	-
unwissend	erzählend		erzählend	erzählend	+
unwissend	schweigsam		unwissend	schweigsam	-
erzählend	unwissend		erzählend	erzählend	+
erzählend	erzählend		schweigsam	schweigsam	+
erzählend	schweigsam		schweigsam	schweigsam	+
schweigsam	unwissend		schweigsam	unwissend	-
schweigsam	erzählend		schweigsam	schweigsam	+
schweigsam	schweigsam		schweigsam	schweigsam	-

Das Programm ist damit im Prinzip beschrieben. Hier noch einige Hinweise:

- Der Benutzer soll am Anfang eingeben, wie viele Bewohner das Dorf hat, und wie viele Durchläufe maximal gewünscht sind.
- Gibt es stabile Zustände? D.h. Zustände, bei denen es keine Änderung der Status der Einwohner mehr gibt? Wenn ja, welche? Erreicht das Programm einen solchen stabilen Zustand, soll die Simulation mit einer entsprechenden Meldung beendet werden.
- Wird das Programm durch die max. Anzahl an Durchläufen beendet, soll die Simulation auch hier mit einer entsprechenden Meldung beendet werden.
- Das Programm soll nach einer Status-Änderung folgende Informationen in einer Zeile ausgeben:
  - Nummer des Durchlaufs (n-stellig in Abhängigkeit der max. Durchläufe, rechtsbündig, führende Leerzeichen)
  - Anzahl an unwissenden Einwohnern in der Form „U(_x)“ (n-stellig in Abhängigkeit der max. Durchläufe, rechtsbündig, führende Underscores)
  - Anzahl an erzählenden Einwohnern – Form anlog zu den Unwissenden
  - Anzahl an schweigsamen Einwohnern – Form anlog zu den Unwissenden
  - Ein Zeichen für jeden Einwohner:
    - Unwissend => „_“
    - Erzählend => „|“
    - Schweigsam => „*“

Eine Simulation könnte also folgendermaßen aussehen:

```
Mögliche Ein- und Ausgabe:
Geruechtekueche
- Anzahl Personen: 30
- Max Durchlaeufe: 400

  0: U(29) E( 1) S( 0) |______________________________
  3: U(28) E( 2) S( 0) |_____________________|________
  5: U(27) E( 3) S( 0) |_____________________|__|_____
  6: U(26) E( 4) S( 0) |__________|__________|__|_____
  8: U(26) E( 2) S( 2) |__________*__________|__*_____
  9: U(25) E( 3) S( 2) |_____|____*__________|__*_____
 19: U(25) E( 2) S( 3) |_____*____*__________|__*_____
 26: U(24) E( 3) S( 3) |_____*____*__|_______|__*_____
 36: U(23) E( 4) S( 3) |_____*__|*__|_______|__*_____
 41: U(22) E( 5) S( 3) |__|__*__|*__|_______|__*_____
 45: U(22) E( 4) S( 4) *__|__*__|*__|_______|__*_____
 47: U(22) E( 2) S( 6) *__|__*__|*__*_______*__*_____
 62: U(21) E( 3) S( 6) *__|__*__|*__*_______*__*___|__
109: U(21) E( 2) S( 7) *__|__*__**__*_______*__*___|__
116: U(21) E( 1) S( 8) *__|__*__**__*_______*__*____*__
136: U(20) E( 2) S( 8) *__|__*__**__*_____|_*__*____*__
139: U(19) E( 3) S( 8) *__|__*__**__*____||_*__*____*__
159: U(19) E( 1) S(10) *__|__*__**__*____**_*__*____*__
182: U(19) E( 0) S(11) *__*__*__**__*____**_*_**_*____*__

Ende wegen stabilem Zustand
```

12.3 Aufgabe „Tic-Tac-Toe 1“ (optional)

Schreiben Sie ein einfaches Tic-Tac-Toe.

Tic-Tac-Toe ist ein einfaches Spiel, das auf einem 3x3 Brett gespielt wird. Es wird abwechselnd

gezogen – jedes Mal legt ein Spieler einen seiner Spielsteine auf ein Brett. Gewonnen hat, wer zuerst 3 eigene Steine in einer Reihe hat (vertikal, horizontal bzw. diagonal). Nach spätestens neun Zügen ist das Spiel vorbei – hat dann keiner der Spieler eine vollständige Reihe geschafft, ist das Spiel unentschieden.

Hinweis – wird das Spiel von einem Spieler optimal gespielt, kann er nicht verlieren. D.h. spielen beide Spieler optimal, so wird es immer unentschieden ausgehen.

Fangen Sie das Programm einfach und klein an, und gehen Sie im ersten Schritt sehr pragmatisch vor. D.h. versuchen Sie das Programm erstmal vollständig (im Sinne von „*man kann spielen*“) zum Laufen zu bringen:

- Machen Sie eine einfache pragmatische Eingabe, z.B. indem die Felder von 1..9 durchnummeriert sind, und Sie einfach nur die Feld-Nummer von der Tastatur abholen.
- Machen Sie nach jedem Zug eine einfache Ausgabe in die Konsole, z.B. indem ein Feld mit „_“ (unbelegt), „X“ (Spieler 1) und „O“ (Spieler 2) ausgegeben wird.
- Legen Sie einfach fest, dass z.B. immer der Mensch anfängt und der Computer der zweite Spieler ist.
- Während des Spiels ist kein Abbruch möglich.
- Und vor allen Dingen verwenden Sie erstmal nicht zu viel Arbeit auf einen guten Computerspieler (zum einen würden Sie dann nicht mehr gewinnen können, zum anderen ist dies eher ein algorithmisches und nicht ein Java Problem). Im einfachsten Fall programmieren Sie einfach einen Computerspieler, der das erste leere Feld besetzt. Wenn dann irgendwann alles gut, stabil, schön usw. läuft, dann sollten Sie sich aber ruhig noch an dieses Problem heranwagen und einen optimalen Computerspieler implementieren.

Mögliche Ein- und Ausgabe:

```
___
___
___
Bitte machen sie ihren Zug (1..9): 1

Mensch zieht 1 - Feld 1/1

O__
___
___
Computer (naechstes freies Feld) zieht 2 - Feld 1/2

OX_
___
___
Bitte machen sie ihren Zug (1..9): 4

Mensch zieht 4 - Feld 2/1

OX_
O__
___
Computer (naechstes freies Feld) zieht 3 - Feld 1/3

OXX
O__
```

```
___
Bitte machen sie ihren Zug (1..9): 7

Mensch zieht 7 - Feld 3/1

OXX
O__
O__

Mensch hat gewonnen
```

Hinweise:

- Lassen Sie sich nicht von dem scheinbar großen Problem einschüchtern. Programmieren heißt immer auch – egal ob prozedural, funktional, objekt-orientiert, oder anders – ein Problem in kleinere Teilprobleme zu zerlegen, bis man diese lösen kann.
- Fangen Sie auch nicht einfach blind an zu Implementieren – dazu ist diese Aufgabe wahrscheinlich schon zu groß. Setzen Sie sich hin, und überlegen Sie, welche Teile Sie zuerst warum und wie implementieren. Welche hängen dann davon ab?
- Identifizieren Sie die wichtigsten Konzepte des Spiels (welche sind das?) und packen Sie diese in Klassen.
- Überlegen Sie, welche Schnittstellen die Klassen benötigen. Welche Klasse hat welche Aufgaben bzw. Verantwortlichkeiten?
- Es ist eine interessante und wichtige Aufgabe sich zu überlegen, an welcher Stelle welche Arbeit erledigt wird – die Kriterien dafür sind Klarheit, Erweiterbarkeit, Allgemeinheit, Wiederverwendbarkeit, usw.

13 Packages

Keine expliziten Aufgaben

14 Vererbung

14.1 Aufgabe „Obstkorb“

Implementieren sie das Obstkorb Programm aus der Vorlesung. Erweitern sie es um eine Klasse für Zitronen.

14.2 Aufgabe „ProgressBar“ (optional)

Ein typisches Problem der Art aus dem Kapitel Modul-Entkopplung ist eine Fortschritts-Anzeige („ProgressBar“). In der BL Schicht findet eine Verarbeitung statt, die längere Zeit braucht. Damit der Benutzer eine Rückkopplung bekommt, soll eine Fortschritts-Anzeige aufgeblendet werden. Nur, die BL-Schicht kennt keine UI-Schicht und weiß auch nicht, was für eine Fortschritts-Anzeige im jeweiligen UI-Kontext sinnvoll ist. Daher bietet sich eine Entkopplung über ein Interface an.

- Denken sie sich eine einfache Verarbeitung in der BL-Schicht aus, die etwas Zeit kostet.

Z.B. eine einfache Schleife mit einem „Thread.Sleep“

- Schreiben sie ein erstes kleines Programm ohne Fortschritts-Anzeige mit BL- und UI-Schicht.
- Definieren sie ein Interface, mit dem eine Fortschritts-Anzeige betrieben werden kann.
- Implementieren sie eine Fortschritts-Anzeige.
- Integrieren sie Interface und Fortschritts-Anzeige in ihr Programm.
- Implementieren sie eine weitere Fortschritts-Anzeige. Zeigen sie durch einen einfachen Austausch der Fortschritts-Anzeigen dass die BL-Schicht mit beiden betrieben werden kann, ohne dass die BL-Schicht betroffen ist (d.h. geändert werden muss).
- Mögliche Fortschritts-Anzeigen auf Konsolen-Ebene wären z.B.:
  - Ausgabe, die von „0 %“ bis „100 %“ hochzählt.
  - Liniengrafik mit z.B. dem Zeichen „|“.

14.3 Aufgabe „Kontaktdaten 3“

Erweitern sie die kleine Kontaktdaten-Verwaltung 2. Gegenüber der damaligen Aufgabe soll dieses Programm aber mehrere Arten von Kontakten verwalten können:

- Single – entspricht der Person aus der alten Aufgabe.
  Besteht aus:
  - Vorname
  - Nachname
  - Telefon (als String)
  - Beschreibung

  Suchen über Vor- und Nachname
- Paar
  Besteht aus:
  - Vorname 1
  - Vorname 2
  - Nachname (gemeinsamer)
  - Telefon (als String)

  Suchen über beide Vornamen und den gemeinsamen Nachname
- Firmenkontakt
  - Firmenname
  - Ansprechpartner
    - Vorname
    - Nachname
    - Position
    - Telefon (als String)

  Suchen über Firmenname, Vor- und Nachname des Ansprech-Partners

14.4 Aufgabe „Kontaktdaten 4“

Trennen sie die Kontaktdaten-Verwaltung in ihre groben Strukturen auf, und packen sie diese

in eigene Packages. Besonders wichtig ist hier die **vollständige Trennung** von Benutzer-Ein- und -Ausgabe und der eigentlichen Programm-Logik. Damit können wir später problemlos eine grafische Oberfläche auf die Programm-Logik aufsetzen.

14.5 Aufgabe „Tic-Tac-Toe 2“ (optional)

Bauen sie das Tic-Tac-Toe 1 Programm so um, dass Vererbung und Polymorphie benutzt wird. Welche Klasse bieten sich im Tic-Tac-Toe dafür an? Wo können sie das Programm dadurch vereinfachen?

14.6 Aufgabe „Tic-Tac-Toe 3“ (optional)

Nach dem Umbau sollte es möglich sein, dass Tic-Tac-Toe so zu erweitern, dass der Benutzer am Anfang die Spieler dynamisch festlegen kann. D.h. implementieren sie eine Abfrage, mit der der Benutzer definiert, ob Spieler 1 bzw. Spieler 2 jeweils ein Mensch oder der Computer ist.

14.7 Aufgabe „Tic-Tac-Toe 4“ (optional)

Implementieren sie einen weiteren einfachen Computer-Spieler, der einfach das Feld per Zufall festlegt. Integrieren sie den Spieler in das Programm und die Spieler-Auswahl von Kapitel 14.6. Sehen sie, wie einfach die Erweiterung des Spiels um einen weiteren Spieler geworden ist?

14.8 Aufgabe „Tic-Tac-Toe 5“ (optional)

Trennen sie das Tic-Tac-Toe in seine groben Strukturen auf, und packen sie diese in eigene Packages. Besonders wichtig ist auch hier die **vollständige Trennung** von Benutzer-Ein- und Ausgabe und der eigentlichen Programm-Logik. Immerhin wollen wir ja auch das Tic-Tac-Toe später noch mit einer grafischen Benutzeroberfläche versehen, aber das Spiel nicht neuschreiben müssen sondern die gesamte Logik wiederverwenden können.

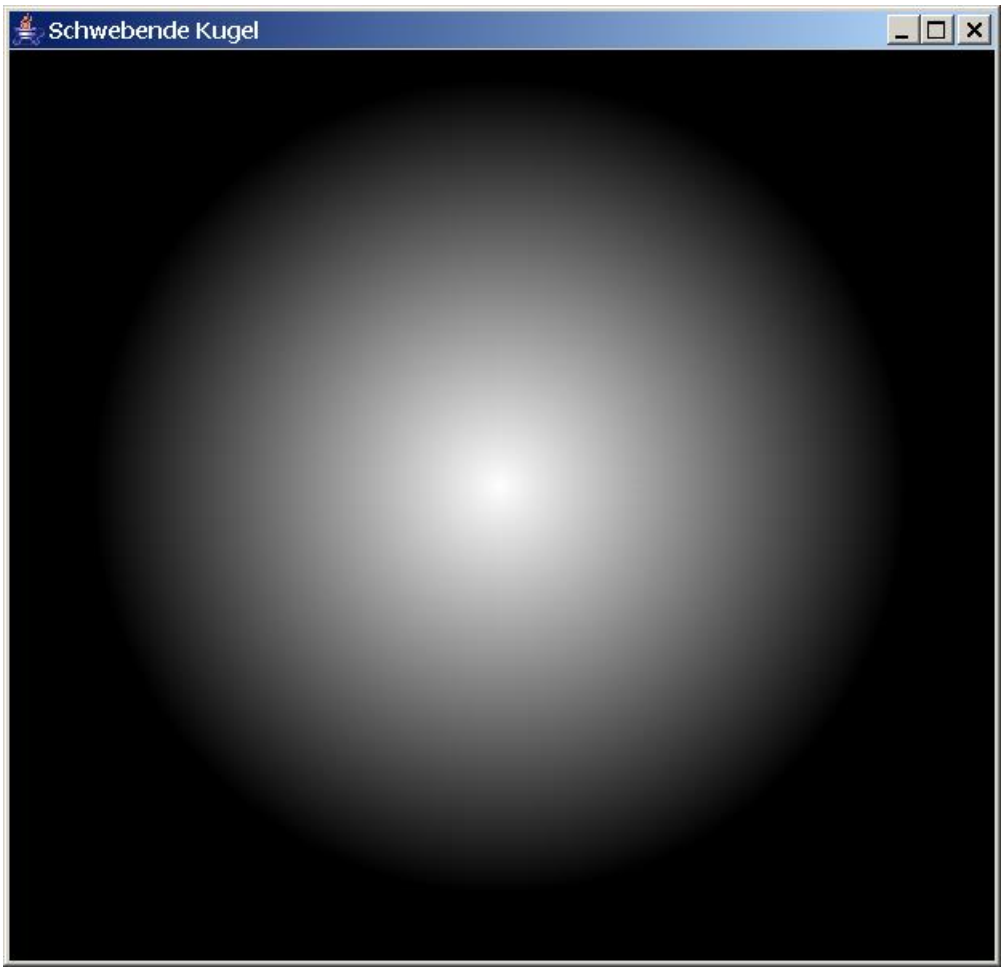
15 Innere Klassen

Keine expliziten Aufgaben

16 GUI Programmierung

16.1 Aufgabe „Schwebende Kugel“

Implementieren Sie ein Fenster, das eine schwebende Kugel wie in der Abbildung enthält. Hinweis – es sind wirklich nur die problemlosen Zeichen-Funktionen benutzt worden. Es ist einfach ein bisschen geschickt mit der Farbe gespielt worden.



17 Philosophie der GUI Programmierung

Keine expliziten Aufgaben

18 Event-Modelle von Swing

18.1 Aufgabe „MainFrame“

Schreiben Sie eine eigene Frame-Klasse „MainFrame“, die von „JFrame“ abgeleitet ist, aber beim Schließen des Fensters immer automatisch das Programm beendet.

- Welche Möglichkeiten haben Sie diese Aufgabe zu lösen?
- Welche Vor- und Nachteile haben die einzelnen Lösungen?
- Welche würden Sie bevorzugen?

18.2 Aufgabe „Scribble 5 zeichnet Rechtecke“

Implementieren Sie mit dem externen Event-Modell ein Rechteck-Scribble, das statt Linien-Züge Rechtecke zeichnet.

- Entwickeln Sie zuerst eine Version ohne Modell, und konzentrieren sie sich auf die Event-Behandlung und die eigentliche Zeichen-Aufgabe.
- Danach entwickeln Sie ein passendes Modell, und integrieren es in Ihr Rechteck-Scribble.

19 Swing Layouts

19.1 Aufgabe „Layouts 1“

Entwickeln Sie ein Fenster, das oben zwei Reihen mit je 6 Buttons enthält, und den Rest des Bildschirms „frei“ lässt.

19.2 Aufgabe „Layouts 2“

Entwickeln Sie ein Fenster, das an den Seiten je 5 Buttons unter einander enthält, und im restlichen Mittelbereich einen einzelnen Button.

20 Swing GUI-Elemente

20.1 Aufgabe „Liste“

Programmieren Sie ein Fenster mit einer Liste – Swing Klasse „JList“ – mit Scrollbars und Listen-Model. Die Benutzung der Listen-Klasse ist quasi analog zur Tabellen-Klasse, nur sind Liste und Listen-Modell einfacher, da eine Liste nur eine Spalte hat.

20.2 Aufgabe „Scribble 6“

Implementieren Sie zuerst ein Scribble, das sowohl Linienzüge als auch Rechtecke zeichnen kann. Setzen sie für die Auswahl Buttons oder ein Menü ein. Implementieren Sie das Scribble in mehreren Schritten:

- Erste Version ohne Modell, mit dem Haupt-Augenmerk auf die Auswahl und die Integration von „Scribble 3 und „Scribble 5.
- In der zweiten Version müssen sie ein Modell entwickeln, das sowohl Linien-Züge als auch Rechtecke aufnehmen kann. Nehmen sie das Model vom „Scribble 4“, erweitern es um Rechtecke, und integrieren es. Denken sie schon nach vorne – auf Dauer werden noch andere Grund-Elemente (Ellipsen, Texte,...) gezeichnet werden müssen. Am besten entwickeln sie also ein Modell, das offen ist für neue Grund-Elemente und mit verschiedenen Arten von Grund-Elementen umgehen kann.

- Wenn sie eine gute offene Lösung für das Modell gefunden haben, ist die Integration einer weiteren Zeichen-Figur kein Problem. Also lassen sie ihr Scribble noch Ellipsen zeichnen können – natürlich zusätzlich zu den Polygonen und Rechtecken, und natürlich auch mit Model. An dem Aufwand, den sie treiben müssen, um Ellipsen zu integrieren können sie erkennen, wie gut ihr Programm-Design ist.

20.3 Aufgabe „TextField“ (optional)

Entwickeln Sie ein eigenes Text-Feld Element, das gegenüber dem Original-Element „JTextField“ noch zusätzlich ein externes Event für die Änderung des Inhalts hat, d.h. einen Change-Listener.

20.4 Aufgabe „Kontaktdaten 5“ (optional)

Implementieren Sie ein GUI für die „Kontaktdaten-Verwaltung 4“

