



JAVA™

Kursbuch Java

Autor: Hubertus Espel (espel@t-online.de)
Version: 02-09-15

„In a world without fences who needs gates?“

Vorwort

Das vorliegende Kursskript wurde als Kursunterlage für den Javakurs des Niedersächsischen Landesinstituts für Fortbildung und Weiterbildung im Schulwesen und Medienpädagogik (NLI) geschrieben, der vom 25. bis 29. September 2000 in den Räumen der BBS am Museumsdorf Cloppenburg (Außenstelle Lönningen) stattfand. Fehler, die während der Kursdurchführung von den Teilnehmern/Teilnehmerinnen entdeckt wurden, sind in der vorliegenden Version des Skripts berichtigt worden.

Das Skript ist nicht für das Selbststudium geschrieben, sondern stellt ein Konzept zur Vermittlung der Programmiersprache Java für größere Lerngruppen vor. Es bedarf nach Ansicht des Autors in allen Übungen der Unterstützung durch eine Lehrperson.

Jede Übungseinheit beginnt mit der Vorstellung der Lernaufgaben. Als Unterstützung enthalten die Übungseinheiten im Folgenden Hilfen (konkrete Anleitungen zur Lösung der einzelnen Aufgaben) und Hinweise (Darstellung von Sachverhalten, die für die Lösung der Aufgaben unabdingbar erscheinen). Jede Übungseinheit wird durch Lösungsvorschläge zu den Aufgaben abgeschlossen.

Bei allen Quelltexten wurde darauf geachtet, dass sie für einen Programmieranfänger leicht zu lesen sind. Auf verschachtelte Anweisungen und Inlinekonstruktionen wurden aus diesem Grund weitgehend verzichtet.

Das Skript kann für Unterrichtszwecke in staatlichen Schulen und Fortbildungseinrichtungen frei genutzt und kopiert werden. Jede darüber hinausgehende Verwendung ohne vorherige Einwilligung des Autors ist untersagt.

Die im Skript erwähnten Software- und Hardwarebezeichnungen sind in den meisten Fällen auch eingetragene Warenzeichen und unterliegen als solche den gesetzlichen Bestimmungen.

Das Skript wurde mit \LaTeX unter Verwendung des Makropackets pdf \LaTeX erstellt.

Der Autor ist für Anregungen und konstruktive Kritik dankbar.

Inhaltsverzeichnis

1	„Hallo Welt“	1
1.1	Aufgabe	1
1.2	Hilfen	2
1.3	Hinweise - Vom Quelltext zur Programmausführung	5
1.3.1	Kompilerspachen	5
1.3.2	Interpretersprachen	5
1.3.3	Javaansatz - „Write Once, Run Anywhere“	6
2	Nekje - Nur ein kleiner Javaeditor	1
2.1	Aufgaben	1
2.2	Hinweise	3
2.2.1	Kurzwahlleiste von Nekje	3
2.3	Dokumentation	4
2.3.1	Die Methoden der Klasse <code>javakurs.io.Ausgabe</code>	4
2.4	Lösungen	6
2.4.1	Lösung zu „HalloWelt2.java“	6
2.4.2	Lösungen zu den ergänzenden Aufgaben	7
3	Von Zahlen und Werten	1
3.1	Aufgaben	1
3.2	Hinweise	3
3.2.1	Einfache Datentypen	3
3.2.2	Arithmetische Operationen	4
3.2.3	Operationen mit Zeichenketten (Strings)	4
3.3	Dokumentation	5
3.3.1	Die Methoden der Klasse <code>javakurs.io.Eingabe</code>	5
3.4	Lösungen	8
4	Machen Sie's mit Methode(n)	1

4.1	Aufgaben	1
4.2	Hinweise	3
4.2.1	Methoden	3
4.2.2	Vergleichsoperatoren	4
4.2.3	Logische Operatoren (Boolsche Operatoren)	4
4.2.4	Bedingte Verzweigung mit if-else	5
4.2.5	Schleifen mit while	5
4.2.6	Weitere Kontrollstrukturen	6
4.3	Lösungen	8
5	Objekte machen Java mächtig	1
5.1	Aufgaben	1
5.2	Hilfen	3
5.3	Hinweise	6
5.3.1	Deklaration einer Objektinstanz	6
5.3.2	Referenz auf ein Objekt	7
5.3.3	Schnittstelle eines Objekts (Interface)	7
5.3.4	Zerstörung eines Objekts	7
5.3.5	Statische Methoden, statische Datenfelder	7
5.4	Dokumentation	9
5.4.1	Die Klasse <code>javakurs.auto.Strasse</code>	9
5.4.2	Die Klasse <code>javakurs.auto.Auto</code>	9
5.4.3	Die Klasse <code>javakurs.io.AusgabeFenster</code>	10
6	Do it yourself	1
6.1	Aufgaben	1
6.2	Hinweise	3
6.2.1	Struktur einer Klasse	3
6.2.2	Definition der Konstruktoren	3
6.2.3	Definition der Eigenschaften	4
6.2.4	Definition der Methoden	5
6.2.5	Kapselung, Schnittstelle	5

6.2.6	Selbstreferenzierung mit <code>this</code>	6
6.2.7	Pakete (Packages)	6
6.2.8	Mehrere Klassen in einer Quelltextdatei	6
6.2.9	Innere Klassen	7
6.2.10	Applikationen objektorientiert schreiben	9
6.3	Lösungen	10
7	Benutzeroberfläche mit AWT	1
7.1	Aufgaben	1
7.2	Hinweise	2
7.2.1	Interfaces	2
7.2.2	Das AWT-Package	3
7.3	Lösungen	9
8	Vererbung am Beispiel farbiger Buttons	1
8.1	Aufgaben	1
8.2	Hinweise	2
8.2.1	Superklasse, Subklasse, Vererbung	2
8.2.2	Zugriffsmodifizierer	3
8.2.3	Überschreiben von Methoden, Polymorphismus	3
8.2.4	Abstrakte Klassen	4
8.2.5	Finale Klassen, finale Methoden	4
8.3	Lösungen	5
9	Der Lebenszyklus eines Applets	1
9.1	Aufgaben	1
9.2	Hinweise	3
9.2.1	Erläuterungen zur Klasse <code>java.applet.Applet</code>	3
10	Septembermorgen	1
10.1	Aufgaben	1
10.2	Hinweise	2
10.2.1	Texformatierungen im Grafik-Kontext	2

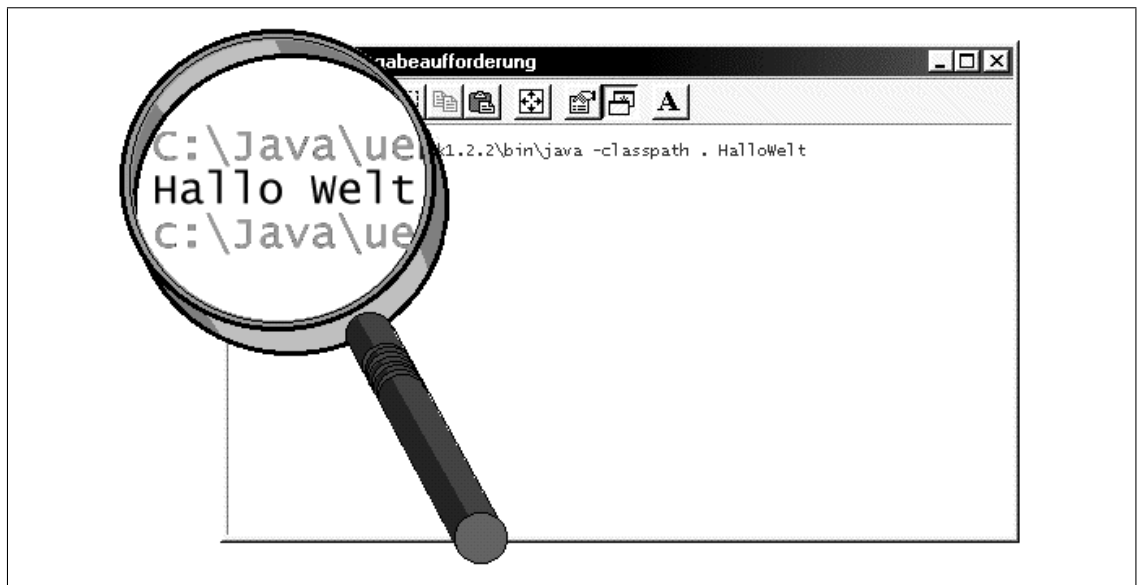
10.3 Lösung	3
11 Wahlergebnisse	1
11.1 Aufgaben	1
11.2 Hinweise	2
11.2.1 Arrays	2
11.3 Hilfen	2
11.4 Lösungen	5
12 Projekte: Farb-Composer und „Tipp die Zahl“	1
12.1 Aufgaben	1
12.2 Hilfen	2
13 Behandlung von Ausnahmen - Java Exceptions	1
13.1 Aufgabe	1
13.2 Hilfen	3
14 Von Dateien und Streams	1
14.1 Aufgaben	1
14.2 Hilfen	2
14.3 Hinweise	2
14.3.1 Das Stream-Konzept	2
14.3.2 Dateien lesen	2
14.3.3 Dateien schreiben	3
14.4 Lösungen	4
15 Multithreading	1
15.1 Aufgaben	1
16 Projekt: Java und Datenbanken	1
16.1 Aufgabe	1
16.2 Hinweise	2
16.2.1 Anlegen einer leeren Access-Datenbank	2
16.2.2 Installation des ODBC-Treibers für eine Access-Datenbank	2

16.2.3	Eine Verbindung zur Datenbank aufbauen	4
16.2.4	Eine Tabelle einrichten	5
16.2.5	Datensätze speichern	6
16.2.6	Datensätze auslesen	7
16.2.7	Die Klasse JdbcMySQLDemo	8
A	Ausgewählte Java-Ressourcen im Internet	1
A.1	Allgemeines	1
A.2	Bücher/Tutorials	1
A.3	Online-Kurse	1
A.4	Entwicklungsumgebungen (IDEs)	1
A.5	Ressourcen-Sites	2
A.6	Newsgroups/Diskussionsforen	2
A.7	Didaktik und Methodik	2

Übung 1

„Hallo Welt“

1.1 Aufgabe



„Hallo Welt“ - die Ausgabe dieser zwei Worte auf dem Bildschirm ist seit Generationen der Programmklassiker für den Programmierlehrling und steht in der Regel ganz am Anfang beim Erlernen einer neuen Sprache.

Auch Ihnen wollen wir den bewegenden Moment nicht verwehren, wenn ihr erstes Java-Programm die Meldung „Hallo Welt“ auf dem Bildschirm schreibt.

Auf geht's - schreiben Sie also ein Java-Programm, das nach dem Vorbild der Abbildung „Hallo Welt“ auf dem Bildschirm ausgibt.

Unterstützende Materialien:

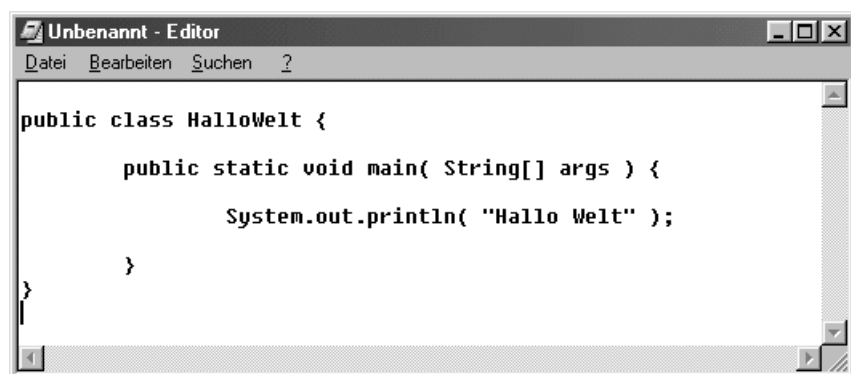
- 1.2 Hilfen
- 1.3 Hinweise - Vom Quelltext zur Programmausführung

1.2 Hilfen

„Halt - Moment“, werden Sie sagen. „Ich kann überhaupt (noch) kein Java.“

Sie haben Recht und darum wird der Weg zur Lösung des Problems nachfolgend Schritt für Schritt beschrieben.

1. Sie brauchen eine ASCII-Editor, mit dem Sie den Java-Quelltext schreiben und abspeichern können. Wenn Sie nichts Besseres zur Hand haben, nehmen Sie einfach den Windows-Editor Notepad.exe (meist zu finden unter Start | Programme | Zubehör | Editor)¹.
2. Nachdem Sie den Editor gestartet haben, können Sie den Quelltext für das „Hallo Welt“-Programm eingeben. Kopieren Sie dazu einfach das Muster der Abbildung!



```
public class HalloWelt {  
  
    public static void main( String[] args ) {  
  
        System.out.println( "Hallo Welt" );  
  
    }  
  
}
```

Mit „class HalloWelt {...}“ schaffen (definieren) Sie als äußere Hülle des Programms ein neue Java-Klasse mit dem Klassennamen „HalloWelt“, deren Funktionalität anschließend innerhalb der geschweiften Klammern festgelegt wird.

Mit „public static void main(String args[]) {...}“ definieren Sie eine Methode mit Namen „main“, die innerhalb der geschweiften Klammern Programmanweisungen enthält. Jede Java-Klasse, die als eigenständiges Computerprogramm (Applikation) ausführbar sein soll, benötigt eine solche Methode „main()“. Die Methode „main()“ stellt standardmäßig den Einstiegspunkt für die Ausführung von Java-Applikationen dar.

Die Anweisung „System.out.println("Hallo Welt");“ beauftragt das Betriebssystem auf dem Standardausgabegerät (i. d. R. auf dem Bildschirm) die Meldung „Hallo Welt“ auszugeben. Benutzt wird die vordefinierte Methode `println()`, die in der Standardbibliothek von Java bereitgestellt wird.

Wichtig: Jede Programmanweisung muss in Java mit einem Semikolon abschließen.

3. Nun müssen Sie den Java-Quelltext abspeichern. Über den Menüpunkt „Datei | Speichern unter ...“ des Editors öffnen Sie ein Eingabefenster. Hier können Sie das Verzeichnis wählen, in dem Sie speichern wollen, und einen Namen für die Quelltextdatei vergeben.

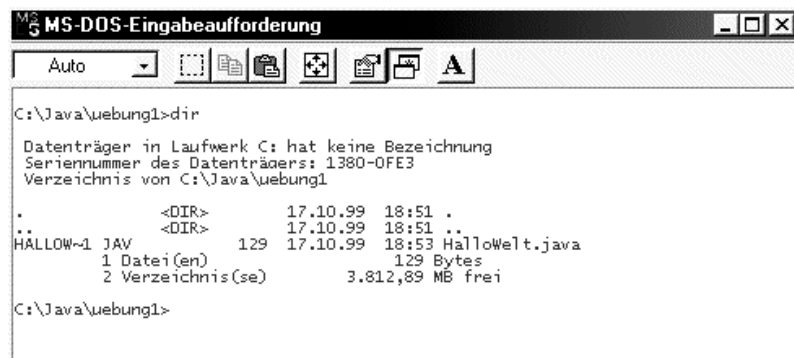
Vorschlag: Speichern Sie die Datei auf einem Laufwerk Ihrer Wahl in einem Unterverzeichnis „\Java\uebung1“.

In der Wahl des Dateinamens sind Sie nicht frei. Java verlangt, dass Sie die Datei nach der Hauptklasse benennen, die Sie im Quelltext definiert haben. Ihre Datei mit der Klasse „HalloWelt“ müssen Sie daher zwingend in der Datei „HalloWelt.java“ abspeichern.

Wichtig: Achten Sie beim Benennen der Datei peinlich genau auf Groß- und Kleinschreibung! Im Windows-Programm Notepad.exe sollten Sie den Dateinamen in Anführungszeichen einschließen, andernfalls hängt der Editor u. U. der Datei den Suffix „.txt“ an - das dürfen Sie aber auf keinen Fall zulassen.

¹Eine gute Alternative zu Notepad.exe ist Kaiedit (Quelle: <http://www.kaiedit.de>)

Nach dem Speichern der Datei „HalloWelt.java“ kontrollieren Sie am Besten im DOS-Fenster, ob die Datei ordnungsgemäß abgelegt wurde. Es sollte sich nach der Eingabe des Dir-Befehls der abgebildete Verzeichnisinhalt zeigen.



```

MS-DOS-Eingabeaufforderung
Auto
C:\Java\uebung1>dir

Datenträger in Laufwerk C: hat keine Bezeichnung
Seriennummer des Datenträgers: 1380-0FE3
Verzeichnis von C:\Java\uebung1

.                <DIR>          17.10.99  18:51  .
..               <DIR>          17.10.99  18:51  ..
HALLOW~1  JAV           129      17.10.99  18:53  HalloWelt.java
           1 Datei(en)                129 Bytes
           2 Verzeichnis(se)         3.812,89 MB frei

C:\Java\uebung1>

```

4. Als Nächstes müssen Sie den Quelltext kompilieren, d. h. mit Hilfe eines Kompilers in eine Binärdatei umwandeln, die vom System ausgeführt werden kann.

Öffnen Sie zu diesem Zweck das DOS-Fenster, wechseln Sie in das Verzeichnis, in dem „HalloWelt.java“ gespeichert wurde und geben Sie auf der Kommandozeile (wie im abgebildeten Beispiel) den Kompilierbefehl 'C:\jdk\bin\javac HalloWelt.java' ein.



```

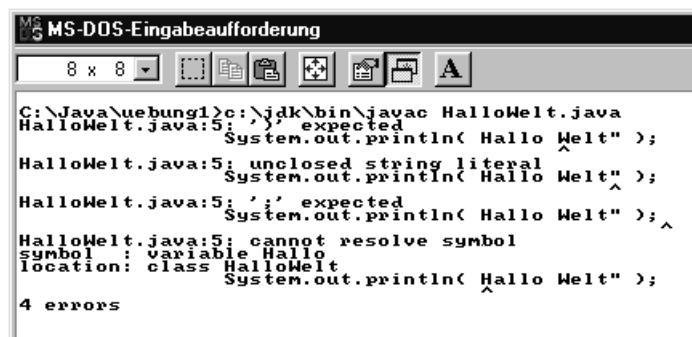
MS-DOS-Eingabeaufforderung
Auto
C:\Java\uebung1>c:\jdk\bin\javac HalloWelt.java

```

Hinweis: Falls das Java-SDK bei Ihnen nicht unter C:\jdk installiert ist, müssen Sie den Programmaufruf natürlich an Ihre Verhältnisse anpassen.

Wenn der Kompiler keine Fehler in der Quelldatei entdeckt, erzeugt er die Datei „HalloWelt.class“ und kehrt ohne weitere Meldung zum Kommandozeilen-Prompt zurück.

Wenn Fehler im Quelltext vorhanden sind, gibt der Kompiler entsprechende Fehlermeldungen unter Nennung der fehlerhaften Quelltextzeilen aus (siehe nachfolgende Abbildung).²



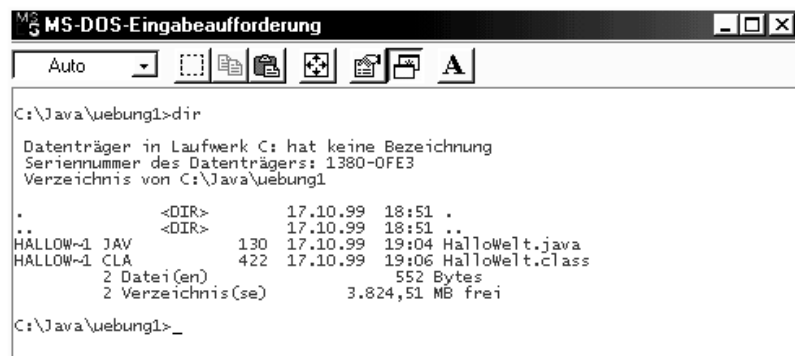
```

MS-DOS-Eingabeaufforderung
8 x 8
C:\Java\uebung1>c:\jdk\bin\javac HalloWelt.java
HalloWelt.java:5:    }, expected
                  System.out.println( Hallo Welt" );
HalloWelt.java:5: unclosed string literal
                  System.out.println( Hallo Welt"
^
HalloWelt.java:5:    ', expected
                  System.out.println( Hallo Welt" );^
HalloWelt.java:5: cannot resolve symbol
symbol  : variable Hallo
location: class HalloWelt
                  System.out.println( Hallo Welt" );
4 errors

```

5. Und nun zum Finale! Der Kompiler hat für Sie das fertige Programm „HalloWelt.class“ im aktuellen Verzeichnis erstellt (siehe Abbildung).

²Leider nur in englischer Sprache, aber Englisch können wir doch, oder?



```
MS-DOS-Eingabeaufforderung
Auto
C:\Java\uebung1>dir
Datenträger in Laufwerk C: hat keine Bezeichnung
Seriennummer des Datenträgers: 1380-0FE3
Verzeichnis von C:\Java\uebung1
.                <DIR>          17.10.99  18:51  .
..               <DIR>          17.10.99  18:51  ..
HALLOW~1  JAV           130    17.10.99  19:04  HalloWelt.java
HALLOW~1  CLA           422    17.10.99  19:06  HalloWelt.class
           2 Datei(en)                552 Bytes
           2 Verzeichnis(se)       3.824,51 MB frei
C:\Java\uebung1>
```

Durch die Kompilierung wurde eine Class-Datei mit sogenanntem Byte-Code geschaffen. Mit Hilfe eines betriebssystemspezifischen Bytecode-Interpreters (Java Virtual Maschine) können Sie nun das „Hallo Welt“-Programm zum Laufen bringen.

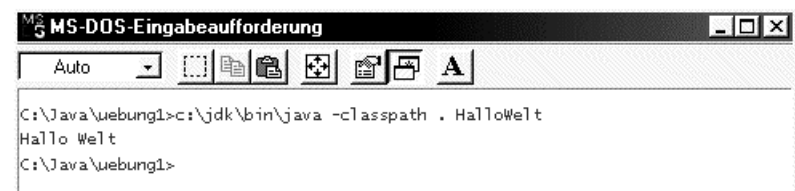
Geben Sie dazu auf der Kommandozeile (im DOS-Fenster) den Aufrufbefehl für den Java-Interpreter ein:

„C:\jdk\bin\java -classpath . HalloWelt“



```
MS-DOS-Eingabeaufforderung
Auto
C:\Java\uebung1>c:\jdk\bin\java -classpath . HalloWelt
```

Wenn alles ordnungsgemäß läuft, dann sollte ihr Computer nun auf dem Bildschirm die Meldung „Hallo Welt“ ausgeben und danach zum DOS-Prompt zurückkehren.



```
MS-DOS-Eingabeaufforderung
Auto
C:\Java\uebung1>c:\jdk\bin\java -classpath . HalloWelt
Hallo Welt
C:\Java\uebung1>
```

6. Bravo!! Egal wie lange Sie bis hierhin gebraucht haben, Beifall haben Sie sich verdient.

1.3 Hinweise - Vom Quelltext zur Programmausführung

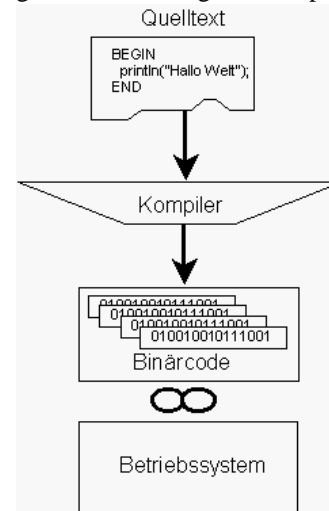
1.3.1 Kompilersprachen

Bei den klassischen Kompilersprachen (z. B. C, C++, Delphi, Visual Basic, Pascal) wandelt der Compiler den Quelltext in Binärcode, der in einer selbständig ausführbaren Datei (z. B. EXE-Datei) dauerhaft gebunden wird (siehe Abbildung 1.1).

Merkmale:

- hohe Ausführungsgeschwindigkeit
- Quelltext bleibt dem Nutzer verborgen
- Abhängigkeit vom Betriebssystem
- große Programmdateien

Abbildung 1.1: Vom Quelltext zur Programmausführung bei Kompilersprachen



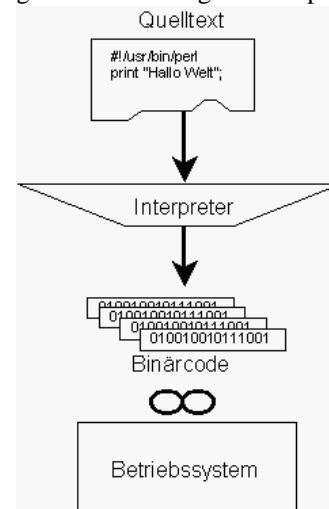
1.3.2 Interpretersprachen

Bei den Interpretersprachen (z. B.: Perl, Basic) wandelt der Interpreter zum Zeitpunkt der Programmausführung den Quelltext zeilenweise in Binärcode (siehe Abbildung 1.2).

Merkmale:

- Veränderbarkeit des Quelltextes auch durch Nutzer
- geringe Ausführungsgeschwindigkeit
- i. d. R. Abhängigkeit vom Betriebssystem
- kleine Quelltextdateien

Abbildung 1.2: Vom Quelltext zur Programmausführung bei Interpretersprachen



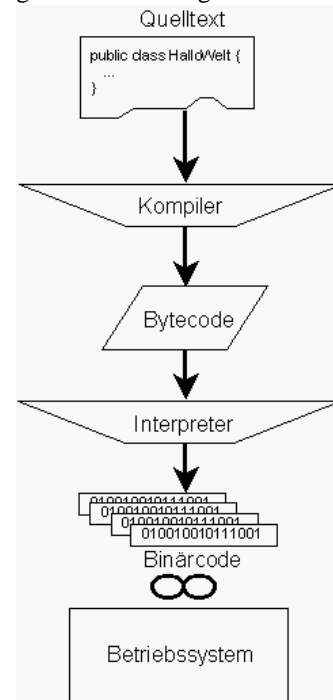
1.3.3 Javaansatz - „Write Once, Run Anywhere“

Bei Java handelt es sich um eine Hybridsprache, bei der sowohl der Compileransatz als auch der Interpreteransatz Verwendung finden. Zunächst wird der Quelltext durch den Java-Kompiler in Bytecode (CLASS-Datei) überführt. Zum Zeitpunkt der Programmausführung sorgt dann die sogenannte Java Virtual Machine (JVM) als Interpreter für die betriebssystemspezifische Umsetzung in Binärcode (siehe Abbildung 1.3).

Merkmale:

- gute Ausführungsgeschwindigkeit durch Optimierung der CLASS-Datei
- Quelltext bleibt dem Nutzer verborgen
- vollkommene Unabhängigkeit vom Betriebssystem (JVM vermittelt zwischen Bytecode und Betriebssystem)
- kleine Programmdateien

Abbildung 1.3: Vom Quelltext zur Programmausführung bei Java



Übung 2

Nekje - Nur ein kleiner Javaeditor

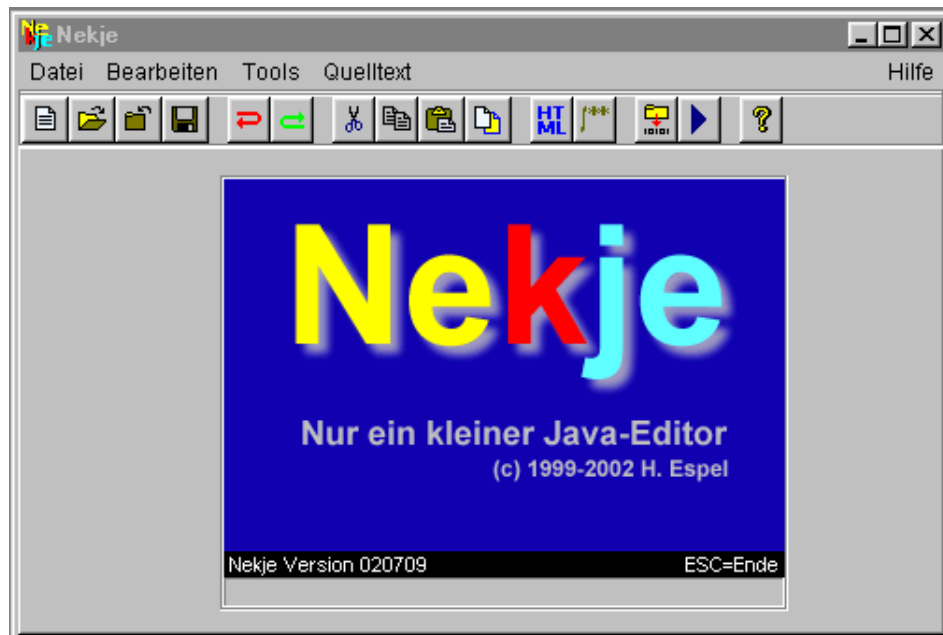
2.1 Aufgaben

Sie haben es natürlich gemerkt - so kann's nicht weitergehen.

Laufend ellenlange Kompiler- und Programmaufrufzeilen eintippen, bei jedem noch so kleinen Tippfehler nervige Fehlermeldungen präsentiert bekommen, immer wieder zwischen Editor und Kommandooberfläche hin- und herschalten ...

Was fehlt ist eine Entwicklungsumgebung (IDE¹), die die lästigen Kleinigkeiten übernimmt und deren Bedienung schnell zu erlernen ist.

Was machen? Unsere Lösung für Sie: NEKJE - eine einfache Java-Entwicklungsumgebung für Programmieranfänger.



Damit Sie sich mit NEKJE vertraut machen können, greifen wir das Problem aus Übung 1 nochmals auf.

Schreiben, kompilieren und testen Sie ein Programm mit dem Dateinamen „HalloWelt2.java“ mit Hilfe von NEKJE.

¹IDE steht für Integrated Development Environment. Für Java existieren eine Reihe von komplexen professionellen IDEs. Nach Abschluß dieses Grundkurses lohnt ein Blick auf die Produkte der Firmen SUN, Borland und IBM.

Ergänzende Aufgaben:

1. Schreiben Sie ein Programm „Javanauten“, das die folgende Bildschirmausgabe erzeugt:

```
*****  
* Hallo liebe Javanauten *  
*****
```

2. Untersuchen Sie, welche Wirkung Sie mit den folgenden Programmanweisungen erzielen:

- (a) `System.out.println("Java ist gar nicht so schwer,\n wie ich gedacht habe!");`
- (b) `System.out.println("Am Ende des Kurses bin ich " + "bestimmt annaeherd perfekt in Java");`
- (c) `System.out.print("Java macht Spass!");`

3. Wenn Sie mit Nekje arbeiten, stehen Ihnen die Methoden `println()` und `print()` aus der Klasse `javakurs.io.Ausgabe` zur Verfügung (siehe Dokumentation).

Schreiben Sie ein Programm, mit dem „Hallo Welt“ unter Benutzung der Methode `Ausgabe.println()` in einem gesonderten Bildschirmfenster ausgegeben wird. Beachten Sie dabei die Hinweise, die in der Dokumentation zur Klasse `javakurs.io.Ausgabe` gegeben werden.

















4. Schreiben Sie unter Verwendung der Methode `javakurs.io.Ausgabe.showImage()` einen Bildbetrachter, der in der Lage ist, eine Bilddatei vom Typ GIF oder JPEG zu laden und deren Inhalt anzuzeigen. Beachten Sie dabei die Hinweise, die in der Dokumentation zur Klasse `javakurs.io.Ausgabe` gegeben werden.

Unterstützende Materialien:

- 2.2 Hinweise - Kurzwahlleiste von Nekje
- 2.3 Dokumentation - Die Methoden der Klasse `Ausgabe`.
- 2.4 Lösungen zu den Aufgaben

2.2 Hinweise

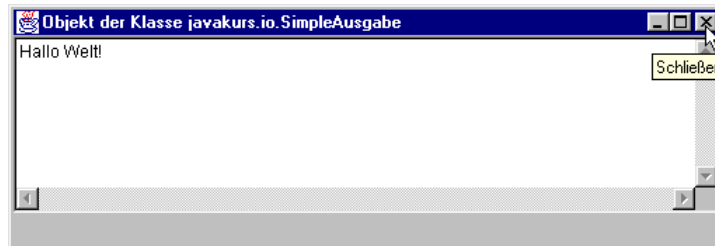
2.2.1 Kurzwahlleiste von Nekje

	Ein neues Quelltextfenster öffnen.
	Eine bestehende Quelltextdatei öffnen.
	Ein geöffnetes Quelltextfenster schließen.
	Einen geänderten Quelltext in Datei speichern.
	Letzte Editieraktion wird rückgängig gemacht.
	Rückgängig gemachte Editieraktion wird wiedergeholt.
	Den markierten Text im aktiven Fenster ausschneiden.
	Den markierten Text in die Zwischenablage kopieren
	Kopierten oder ausgeschnittenen Text aus der Zwischenablage in den Text einfügen.
	Eine Kopie des aktiven Quelltextes in neuem Fenster anlegen (duplizieren).
	Fügt ein Paar geschweifeter Klammern als Blockabgrenzung in den aktiven Quelltext ein.
	Fügt einen Kommentarblock ein.
	Generiert passend zum aktiven Quelltext eine HTML-Datei.
	Der Quelltext im aktiven Fenster wird kompiliert.
	Das Javaprogramm im aktiven Fenster wird ausgeführt.
	Die Kontexthilfe wird aufgerufen.

2.3 Dokumentation

2.3.1 Die Methoden der Klasse `javakurs.io.Ausgabe`

Die Klasse `javakurs.io.Ausgabe` stellt ein elementares Ausgabefenster für die Anzeige von Text- oder Bilddaten bereit (siehe Abbildung).



Die Klasse `javakurs.io.Ausgabe` gehört nicht zur Standardbibliothek von Java. Sie wird vielmehr zusammen mit dem Programm Nekje zur Verfügung gestellt und dient vornehmlich Lernzwecken.

Wichtig: Methoden der Klasse `Ausgabe` können Sie in einer Applikation nur dann nutzen, wenn Sie die Anweisung `import javakurs.io.*;` vor dem ersten Gebrauch einer Ausgabe-Methode im Quelltext einfügen. Alternativ können Sie die Methoden auch in Langformat nach dem Muster `javakurs.io.Ausgabe.print("Text");` aufrufen.

Beispiel:

```
import javakurs.io.*;

public class BeispielApp {
    public static void main( String[] argv ) {
        Ausgabe.println( "Hallo" );
    }
}
```

Alternativbeispiel:

```
public class BeispielApp {
    public static void main( String[] argv ) {
        javakurs.io.Ausgabe.println( "Hallo" );
    }
}
```

Die Klasse `javakurs.io.Ausgabe` stellt die folgenden Methoden zur Verfügung:

`Ausgabe.print()`

Diese Methode gibt Text in einem Ausgabefenster aus. Es erfolgt kein Zeilenumbruch nach der Textausgabe.

Anwendungsbeispiel: `Ausgabe.print("Guten Morgen");`

`Ausgabe.println()`

Diese Methode gibt Text in einem Ausgabefenster aus. Die Ausgabe wird mit einem Zeilenumbruch abgeschlossen.

Anwendungsbeispiel: `Ausgabe.println("Guten Tag");`

Ausgabe.showImage()

Diese Methode zeigt Bilder im GIF- und JPEG-Format in einem separaten Fenster an. Pfad bzw. URL der Bilddatei müssen als Parameter übergeben werden.

Anwendungsbeispiel:

```
Ausgabe.showImage("c:/bilder/test.gif");  
Ausgabe.showImage("http://irgendwo.de/test.gif");
```

Anmerkung:

Der Zugriff auf Web-Ressourcen funktioniert nicht o. W., wenn ein Proxy-Server den Zugriff vermittelt. In diesem Fall müssen Sie zunächst die folgenden Zeilen (mit den für Ihre Konfiguration gültigen Serverdaten) einbinden.

```
System.getProperties().put("proxySet", "true");  
System.getProperties().put("proxyHost", "meinProxyServer");  
System.getProperties().put("proxyPort", "8080");
```

Ausgabe.showWindow()

Diese Methode zeigt das Ausgabefenster und alle verbundenen Fenster an, wenn sie zuvor verborgen waren.

Anwendungsbeispiel: `Ausgabe.showWindow();`

Ausgabe.hideWindow()

Diese Methode verbirgt das Ausgabefenster und alle verbundenen Fenster, wenn sie zuvor sichtbar waren.

Anwendungsbeispiel: `Ausgabe.hideWindow();`

Ausgabe.clear()

Diese Methode löscht den gesamten Inhalt des Ausgabefensters und schließt verbundene Fenster. Das Ausgabefenster bleibt geöffnet.

Anwendungsbeispiel: `Ausgabe.clear();`

2.4 Lösungen

2.4.1 Lösung zu „HalloWelt2.java“

Notwendige Schritte:

1. Nekje über Aufruf von nekje.bat starten.
2. Über Menüpunkt „Datei | Neu“ oder entsprechenden Button in der Kurzwahlleiste ein neues Quelltextfenster öffnen.
3. Quelltext nach dem Muster der Übung 1 eingeben. Einzige Änderung: Der Name der Klasse sollte nach „HalloWelt2“ geändert werden.
4. Quelltext über Menüpunkt „Datei | Speichern“ oder „Datei | Speichern unter ...“ bzw. über Button in der Kurzwahlleiste mit dem Dateinamen „HalloWelt2.java“ im Verzeichnis für die Quelltexte speichern. Der Suffix „.java“ wird bei Java-Quelltextdateien automatisch vergeben, er kann aber auch explizit bei der Vergabe des Dateinamens angegeben werden.
5. Quelltext kompilieren über Menüpunkt „Tools | Kompilieren“ oder über den entsprechenden Button in der Kurzwahlleiste.
6. Programm testen über Menüpunkt „Tools | Ausführen“ oder über den entsprechenden Button in der Kurzwahlleiste.

2.4.2 Lösungen zu den ergänzenden Aufgaben

- Die geforderte Bildschirmausgabe lässt sich wie folgt erzeugen (Zeilennummern natürlich nicht mit eingeben):

```

1 public class Javanauten {
2     public static void main( String[] args ){
3
4         System.out.println( "*****" );
5         System.out.println( "*_Hallo_liebe_Javanauten_*" );
6         System.out.println( "*****" );
7
8     }
9 }

```

- Mit der Kontrollsequenz „\n“ kann ein Zeilenumbruch innerhalb eines Textbereichs (String) erzeugt werden.
 - Mit „+“ können Strings aneinander gekettet werden. So können Sie auch überlange Strings übersichtlich generieren.
 - `System.out.print()` erzeugt eine Bildschirmausgabe (genauer: eine Schreiben in die Standardausgabe des Betriebssystems) ohne abschließenden Zeilenumbruch.
- Nachfolgend zwei Alternativen als Lösung der Aufgabe:

Alternative 1: Methodenaufruf in Langformat

```

1 public class HalloWeltAusgabe1 {
2
3     public static void main( String[] args ) {
4
5         javakurs.io.Ausgabe.println("_Hallo_Welt" );
6
7     }
8 }

```

Alternative 2: Importanweisung

```

1 import javakurs.io.*;
2
3 public class HalloWeltAusgabe2 {
4
5     public static void main( String[] args ) {
6
7         Ausgabe.println("_Hallo_Welt" );
8
9     }
10 }

```

- Das Bildbetrachtungsprogramm kann das folgende Aussehen haben:

```

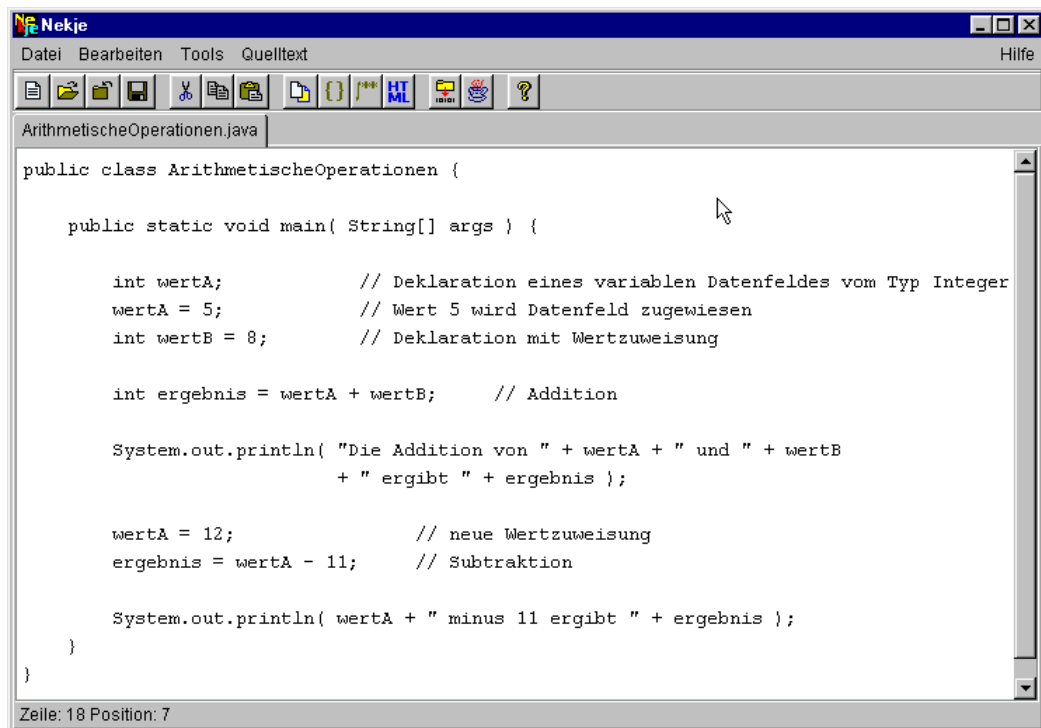
1 import javakurs.io.*;
2
3 public class BildBetrachter {
4
5     public static void main( String[] args ) {
6
7         // Anwendungscode
8         Ausgabe.showImage( "D:\\uebung2\\schule.jpg" );
9     }
10 }

```

Übung 3

Von Zahlen und Werten

3.1 Aufgaben



The screenshot shows the Nekje IDE window titled 'ArithmetischeOperationen.java'. The code defines a class 'ArithmetischeOperationen' with a 'main' method. It demonstrates two arithmetic operations: addition and subtraction. The first part declares 'wertA' as 5 and 'wertB' as 8, then calculates 'ergebnis = wertA + wertB' and prints the result. The second part updates 'wertA' to 12 and calculates 'ergebnis = wertA - 11', then prints the result. The status bar at the bottom indicates 'Zeile: 18 Position: 7'.

```
public class ArithmetischeOperationen {  
  
    public static void main( String[] args ) {  
  
        int wertA;           // Deklaration eines variablen Datenfeldes vom Typ Integer  
        wertA = 5;          // Wert 5 wird Datenfeld zugewiesen  
        int wertB = 8;      // Deklaration mit Wertzuweisung  
  
        int ergebnis = wertA + wertB;    // Addition  
  
        System.out.println( "Die Addition von " + wertA + " und " + wertB  
                               + " ergibt " + ergebnis );  
  
        wertA = 12;         // neue Wertzuweisung  
        ergebnis = wertA - 11;    // Subtraktion  
  
        System.out.println( wertA + " minus 11 ergibt " + ergebnis );  
  
    }  
}
```

Der obige Bildschirm-Schnappschuss zeigt zwei einfache arithmetische Operationen (Addition und Subtraktion) unter Nutzung variabler Datenfelder mit Daten vom Typ Integer.

1. Ergänzen Sie das Programm „ArithmetischeOperationen“ um Beispiele für Multiplikations-, Divisions- und Modulo-Operationen.
2. Verändern Sie das Programm dahingehend, dass Dezimalwerte korrekt verarbeitet werden.
3. Wenn Sie mit Nekje arbeiten, steht Ihnen die Methoden `Eingabe.read()` und `Eingabe.getDouble()` zur Verfügung. Schreiben Sie unter Verwendung dieser Methoden und der Methode `Ausgabe.println()` ein Programm, das alle arithmetischen Operationen auf zwei einzulesende Zahlenwerte anwendet.
4. Entwickeln Sie einen Währungsrechner, welcher beliebige Eurobeträge in US-Dollars und in japanische Yen umrechnet.

Unterstützende Materialien:

- 3.2 Hinweise - Einfache Datentypen, Arithmetische Operationen, Operationen mit Zeichenketten

- 3.3 Dokumentation - Die Methoden der Klasse Eingabe
- 3.4 Lösungen

3.2 Hinweise

3.2.1 Einfache Datentypen

Typ	Bezeichner	Länge	Wertebereich	Beispiel
ganze Zahlen	byte	1 Byte	-128 bis +127	byte wert = 5;
	short	2 Bytes	-32.768 bis+32.767	short wert = 255;
	int	4 Bytes	-2.147.483.648 bis +2.147.483.647	int wert = 123000;
	long	8 Bytes	-9.223.372.036.854.775.808 bis +9.223.372.036.854.775.807	long wert = 3456L;
Gleitkomma- zahlen	float	4 Bytes	1,40239846E-45 bis 3,40282347E+38	float wert = 5.2F;
	double	8 Bytes	4,94065645841246544E-324 bis 1,79769313486231570E+308	double wert = 23.9;
Alpha- numerischer Typ	char	2 Bytes	Alle Unicode-Zeichen	char wert = 'a';
Logischer Typ	boolean	1 Byte	true oder false	boolean wert = true;

Mit einer Anweisung nach dem Muster „`int varWert;`“ wird ein variables Datenfeld deklariert. Einem variablen Datenfeld können Daten, die dem Typ des Datenfeldes entsprechen, zugewiesen werden (Beispiel: `varWert = 18;`). Deklaration und Wertzuweisung können in einer Anweisung erfolgen.

Wenn Datenfeld und Datenwert im Typ nicht übereinstimmen, führt dies zu einem Kompilerfehler („Incompatible type for ... Explicit cast needed to convert ... to ...“). Soweit dies sinnvoll erscheint, kann die Inkompatibilität durch den Gebrauch eines Cast-Operators behoben werden (Beispiel: `int wert = (int)23.6;`).

Mit Hilfe des Modifizierers „`final`“ kann ein konstantes Datenfeld deklariert werden. Konstante Datenfelder können einmal bei der Deklaration mit einem Wert belegt werden. Danach führt jeder Veränderungsversuch zu einem Kompilerfehler.

Beispiel für die Deklaration einer Konstanten:

```
final int konstWert = 450;
```

Datenfelder sind nur innerhalb des gleichen Codeblocks adressierbar. Ein Codeblock wird durch die öffenden und schließenden geschweiften Klammern abgegrenzt.

Beispiel:

```

1 public class TestApplikation {
2
3     public static void main( String[] args ) {
4
5         int wertA = 15;
6
7         {
8             int wertB = 14;
9             int wertC = wertB + wertA;
10        }
11
12        {
13            wertB = 12;           // <---- Fehler!!!
14            int wertC = wertA;  // <---- korrekt
15        }
16    }
17 }
```

3.2.2 Arithmetische Operationen

Operator	Bedeutung	Beispiel
+	Addition von Zahlenwerten	<pre>int summe = 100 + 500;</pre>
++	Erhöhung um eine Einheit	<pre>int summe = wertA + wertB; int wertA = 5; int wertB = wertA++; // (wertB nun 5, wertA nun 6) wertB = ++wertA; // (wertB und wertA nun 7)</pre>
--	Minderung um eine Einheit	<pre>int wertA = 5; int wertB = wertA--; // (wertB nun 5, wertA nun 4) wertB = --wertA; // (wertB und wertA nun 3)</pre>
-	Subtraktion von Zahlenwerten	<pre>int differenz = 500 - 100;</pre>
*	Multiplikation von Zahlenwerten	<pre>int produkt = 5 * 4;</pre>
/	Division von Zahlenwerten	<pre>int quotient = 10 / 2;</pre>
%	Modulo-Operation (Rest von Divisionsoperationen)	<pre>int Modulowert = 13%5; (Ergebnis: 3)</pre>

3.2.3 Operationen mit Zeichenketten (Strings)

Zeichenketten (Beispiel: „Hallo Welt“) gehören in Java nicht zu den einfachen Datentypen. Gleichwohl können sie wie ein einfacher Datentyp mit dem „=“-Operator als Ganzes einem String-Datenfeld zugewiesen werden. Gespeichert wird dabei im Datenfeld allerdings nicht der gesamte String, sondern nur ein Verweis (Fachbegriff: Referenz) auf den Bereich im Arbeitsspeicher, in dem der String abgelegt ist.

Anwendungsbeispiel: `String str = "Hallo Welt";`

Im weiteren Programmablauf kann mittels der Referenz auf den Inhalt des Strings zugegriffen werden.

Anwendungsbeispiel: `System.out.println(str);`

Strings können mittels des „+“-Operators aneinander gekettet werden.

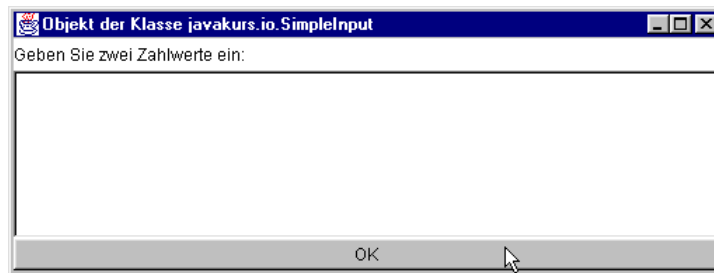
Anwendungsbeispiele:

```
String str1 = "Hallo " + "Welt!";
String str2 = str1 + "Ich gruesse alle, die mich kennen.";
String str3 = "Hello World!";
String str4 = str1 + " " + str3;
```


3.3 Dokumentation

3.3.1 Die Methoden der Klasse `javakurs.io.Eingabe`

Die Klasse `javakurs.io.Eingabe` stellt ein elementares Eingabefenster zur Verfügung, mit dem Daten in ein Programm eingelesen werden können (siehe Abbildung).



Die Klasse `javakurs.io.Eingabe` gehört nicht zur Standardbibliothek von Java. Sie wird vielmehr zusammen mit dem Programm `Nekje` zur Verfügung gestellt und dient vornehmlich Lernzwecken.

Wichtig: Methoden der Klasse `Eingabe` können Sie in einer Applikation nur dann nutzen, wenn Sie die Anweisung `import javakurs.io.*;` vor dem ersten Gebrauch einer Eingabe-Methode im Quelltext einfügen. Alternativ können Sie die Methoden auch in Langformat nach dem Muster `javakurs.io.Eingabe.read();` aufrufen.

Wenn Sie nur Methoden der Klasse `Eingabe` benutzen, dann müssen Sie das Programm mit der Anweisung `System.exit(0);` beenden.

Die Klasse `javakurs.io.Eingabe` stellt die folgenden Methoden zur Verfügung:

`char getChar(int index);`

Die Methode gibt den Char-Wert (Buchstabe) zurück, der im Eingabefenster an der Position `index` eingegeben wurde.

Anwendungsbeispiel:

```
Eingabe.read("Geben Sie zwei einzelne Buchstaben ein:");
char wert1 = Eingabe.getChar(0);
char wert2 = Eingabe.getChar(1);
```

`double getDouble(int index);`

Die Methode gibt den Double-Wert zurück, der im Eingabefenster an der Position `index` eingegeben wurde.

Anwendungsbeispiel:

```
Eingabe.read("Geben Sie zwei Zahlwerte ein:");
double wert1 = Eingabe.getDouble(0);
double wert2 = Eingabe.getDouble(1);
```

`float getFloat(int index);`

Die Methode gibt den Float-Wert zurück, der im Eingabefenster an der Position `index` eingegeben wurde.

Anwendungsbeispiel:

```
Eingabe.read("Geben Sie zwei Zahlwerte ein:");
float wert1 = Eingabe.getFloat(0);
float wert2 = Eingabe.getFloat(1);
```

int getInt(int index);

Die Methode gibt den Integer-Wert zurück, der im Eingabefenster an der Position *index* eingegeben wurde.

Anwendungsbeispiel:

```
Eingabe.read("Geben Sie zwei Ganzzahlwerte ein:");
int wert1 = Eingabe.getInt(0);
int wert2 = Eingabe.getInt(1);
```

String getText(int index);

Die Methode gibt den Text als String zurück, der im Eingabefenster an der Position *index* eingegeben wurde.

Anwendungsbeispiel:

```
Eingabe.read("Geben Sie zwei Worte ein:");
String str1 = Eingabe.getText(0);
String str2 = Eingabe.getText(1);
```

String getText();

Die Methode gibt die gesamte Eingabe als String zurück.

Anwendungsbeispiel:

```
Eingabe.read("Geben Sie Text ein:");
String str = Eingabe.getText();
```

String read(String prompt);

Die Methode öffnet ein Eingabefenster. Der Programmablauf wird bis zum Schließen des Fensters durch den OK-Button unterbrochen. Als Parameter *prompt* wird der Methode ein String übergeben, der als Text oberhalb des Eingabefeldes ausgegeben wird. Als Rückgabewert wird die Eingabe des Benutzers als String geliefert.

Anwendungsbeispiel:

```
String str = Eingabe.read("Geben Sie hier Ihre Meinung ein:");
```

String read();

Die Methode öffnet ein Eingabefenster. Der Programmablauf wird bis zum Schließen des Fensters durch den OK-Button unterbrochen. Als Rückgabewert wird die Eingabe des Benutzers als String geliefert.

Anwendungsbeispiel: `String str = Eingabe.read();`

void setDelimiter(String delim);

Die Methode setzt die Zeichen, die als Trennzeichen zwischen verschiedenen Eingabewerten fungieren. Standardmäßig werden Leerzeichen und Zeilenumbrüche erkannt.

Anwendungsbeispiel: `Eingabe.setDelimiter("\n");`

In diesem Fall werden nur Zeilenumbrüche als Trennzeichen anerkannt.

int `size()`;

Die Methode gibt die Anzahl der eingelesenen Wert zurück.

Anwendungsbeispiel:

`Eingabe.read("Geben Sie Werte ein:");`

`int anzahl = Eingabe.size();`

3.4 Lösungen

1. Die erweiterte Applikation kann folgendes Aussehen haben:

```
1 import javakurs.io.*;
2
3 public class ArithmetischeOperationen2 {
4
5     public static void main( String[] args ) {
6
7         int wertA; // Deklaration eines var. Integer-Datenfeldes
8         wertA = 5; // Wert 5 wird Datenfeld zugewiesen
9         int wertB = 8; // Deklaration mit Wertzuweisung
10
11        int ergebnis = wertA + wertB; // Addition
12        Ausgabe.println( "Die_Addition_von_" + wertA
13        + "_und_" + wertB + "_ergibt_" + ergebnis );
14
15        wertA = 6; // neue Wertzuweisung
16        ergebnis = wertA * 3; // Multiplikation
17        Ausgabe.println( wertA
18        + "_multipliziert_mit_3_ergibt_" + ergebnis );
19
20        ergebnis = wertA / 3; // Division
21        Ausgabe.println( wertA
22        + "_dividiert_durch_3_ergibt_" + ergebnis );
23
24        wertA = 13; // neue Wertzuweisung
25        ergebnis = wertA % 11; // Modulo
26        Ausgabe.println( wertA
27        + "_modulo_11_ergibt_" + ergebnis );
28    }
29 }
```

2. Ersetzen Sie die Integer-Datenfelder durch Datenfelder vom Typ `double`. Hinweis: Rechnerische Ungenauigkeiten sind systembedingt.

```

1 import javakurs.io.*;
2
3 public class ArithmetischeOperationen3 {
4
5     public static void main( String[] args ) {
6
7         double wertA; // Deklaration eines variablen Double-Datenfeldes
8         wertA = 5.5; // Wert 5 wird Datenfeld zugewiesen
9         double wertB = 8; // Deklaration mit Wertzuweisung
10
11        double ergebnis = wertA + wertB; // Addition
12        Ausgabe.println( "Die Addition von " + wertA
13        + " und " + wertB + " ergibt " + ergebnis );
14
15        wertA = 6.2; // neue Wertzuweisung
16        ergebnis = wertA * 3; // Multiplikation
17        Ausgabe.println( wertA
18        + " multipliziert mit 3 ergibt " + ergebnis );
19
20        ergebnis = wertA / 3; // Division
21        Ausgabe.println( wertA
22        + " dividiert durch 3 ergibt " + ergebnis );
23
24        wertA = 13.7; // neue Wertzuweisung
25        ergebnis = wertA % 11; // Modulo
26        Ausgabe.println( wertA
27        + " modulo 11 ergibt " + ergebnis );
28    }
29 }

```

3. Die Eingabe-Applikation kann folgendes Aussehen haben:

```

1 import javakurs.io.*;
2
3 public class ArithmetischeOperationen4 {
4
5     public static void main( String[] args ) {
6
7         /* Eingabe */
8         Eingabe.read( "Geben Sie zwei Zahlwerte ein:" );
9         double wertA = Eingabe.getDouble(0);
10        double wertB = Eingabe.getDouble(1);
11
12        /* Verarbeitung und Ausgabe */
13
14        double ergebnis = wertA + wertB; // Addition
15        Ausgabe.println( "Die Addition von " + wertA
16        + " und " + wertB + " ergibt " + ergebnis );
17
18        ergebnis = wertA * wertB; // Multiplikation
19        Ausgabe.println( wertA
20        + " multipliziert mit " + wertB + " ergibt " + ergebnis );
21
22        ergebnis = wertA / wertB; // Division
23        Ausgabe.println( wertA
24        + " dividiert durch " + wertB + " ergibt " + ergebnis );
25
26        ergebnis = (int)wertA % (int)wertB; // Modulo

```

```
27     Ausgabe.println( "Die_Division_von_" + (int)wertA
28     + "_durch_" + (int)wertB + "_ergibt_einen_Rest_von_" +
29     (int)ergebnis );
30 }
31 }
```

4. Die Applikation „Währungsrechner“ kann folgendes Aussehen haben:

```
1  import javakurs.io.*;
2
3  public class Waehrungsrechner {
4
5      public static void main( String[] args ) {
6
7          double kursDollar = 0.8916;
8          double kursYen = 96.2763;
9
10         // Eingabe
11         Eingabe.read("Geben_Sie_einen_Eurobetrag_ein:");
12
13         double euroBetrag = Eingabe.getDouble(0);
14
15         // Verarbeitung + Ausgabe
16         double fremdBetrag = euroBetrag / kursDollar;
17
18         Ausgabe.println( euroBetrag + "_Euros_entsprechen_"
19         + fremdBetrag + "_Dollars");
20
21         fremdBetrag = euroBetrag / kursYen;
22
23         Ausgabe.println( euroBetrag + "_Euros_entsprechen_"
24         + fremdBetrag + "_Yen" );
25     }
26 }
```

Übung 4

Machen Sie's mit Methode(n)

4.1 Aufgaben

```
1 import javakurs.io.*;
2
3 public class Potenzierung {
4
5     public static long hochZwei( long wert ) {
6         long resultat = wert * wert;
7         return resultat;
8     }
9
10    public static long hochDrei( long wert ) {
11        long resultat = hochZwei( wert ) * wert;
12        return resultat;
13    }
14
15    public static void main( String[] args ) {
16        Eingabe.read( "Potenzierung_-_Geben_Sie_einen_Basiswert_ein:" );
17        int wertA = Eingabe.getInt(0);
18
19        long wertB = hochZwei( wertA );
20        long wertC = hochDrei( wertA );
21
22        Ausgabe.println( wertA + "_hoch_Zwei_=" + wertB );
23        Ausgabe.println( wertA + "_hoch_Drei_=" + wertC );
24    }
25 }
```

Der obige Quelltext demonstriert ein Beispiel für die Verwendung eigendefinierter Methoden (Zeile 5 bis 8 und Zeile 10 bis 12) in einem Javaprogramm.

1. Ergänzen Sie die Applikation um eine weitere eigendefinierte Methode `hochVier()`.
2. Schreiben Sie einen einfachen Taschenrechner, der in der Lage ist, Eingaben nach dem Muster „6 + 9“ auszuwerten. Implementieren Sie die vier Grundrechenarten Addition, Subtraktion, Multiplikation und Division. Definieren Sie dazu in der Taschenrechner-Klasse für jede arithmetische Operation eine eigene Methode.

Lösen Sie die Aufgabe unter Verwendung der bedingten Verzweigung (siehe Hinweise).

Hilfe: Mit der Methode `Eingabe.getChar()` aus dem Paket `javakurs.io` können Sie einzelne Buchstaben (z.B. '+') in ein Char-Datenfeld einlesen.

3. Schreiben Sie eine Applikation „PotenzenApp“, die neben der Methode `main()` noch über die Methoden `potenzieren()` und `bildschirmAusgabe()` verfügt und die Potenzwerte im Bereich 'i = 0 ... n' ermittelt. Lassen Sie den Basiswert und den oberen Grenzwert n vom Benutzer eingeben.

Die Methoden sollen die folgende Schnittstellen-Beschreibung erfüllen:

public static long potenzieren(long eingabeWert, int n)

Die Methode gibt das Ergebnis der Operation eingabeWert^n als Long-Integer zurück.

public static void bildschirmAusgabe(long eingabeWert, int n)

Die Methode gibt das Ergebnis der Operation eingabeWert^n auf dem Bildschirm aus.

Lösen Sie die Aufgabe unter Verwendung der bedingten Wiederholung (siehe Hinweise).

4. Schreiben Sie eine Applikation „Fakultaeten“, die die Ergebnisse von $n!$ für $i = 1 \dots n$ ausgibt. Lösen Sie die Aufgabe unter Verwendung der bedingten Wiederholung (siehe Hinweise). Lassen Sie den oberen Grenzwert vom Benutzer eingeben.

Hilfe: In der Mathematik bedeutet z.B. $5!$ (gesprochen „fünf Fakultät“) das Produkt von $1 * 2 * 3 * 4 * 5$, also $5! = 120$. Entsprechend kann die Fakultät für jede beliebige ganze positive Zahl ermittelt werden.

Unterstützende Materialien:

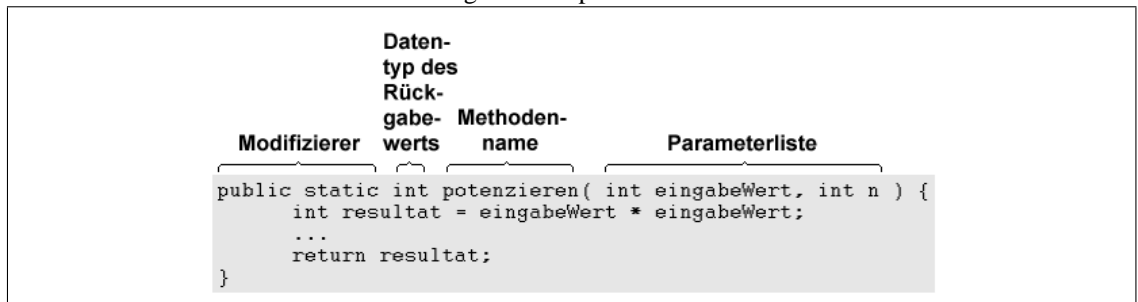
- 4.2 Hinweise - Methoden, Vergleichsoperatoren, logische Operatoren, bedingte Verzweigungen, bedingte Wiederholungen
- 4.3 Lösungen

4.2 Hinweise

4.2.1 Methoden

Funktional zusammenhängende Programmanweisungen werden in Methoden zusammengefaßt (siehe Abbildung 4.1).

Abbildung 4.1: Beispiel einer Methode



Allgemein werden Methoden in Java nach folgendem Muster definiert:

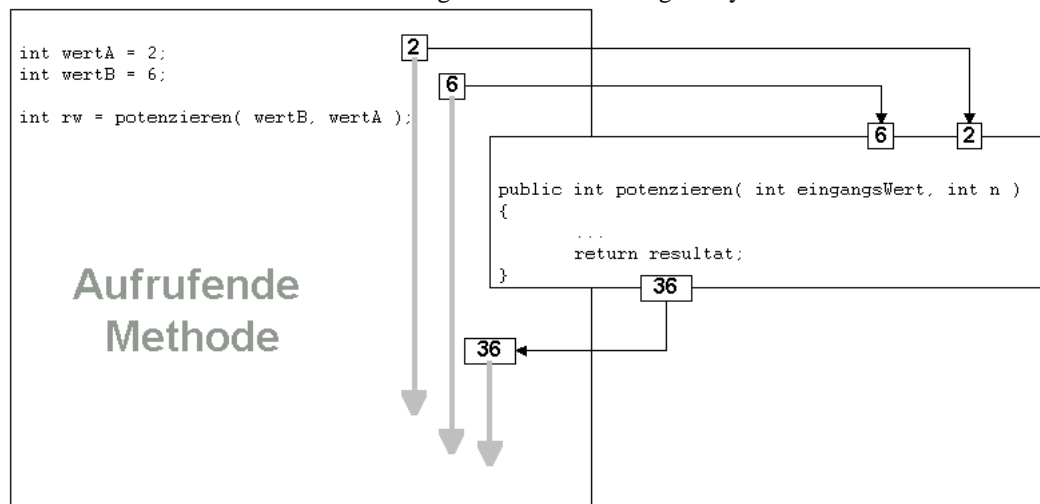
```
[Modifizierer] <Typ d. Rückgabewerts> <Methodenname>( [Parameterliste] ) {
    [Anweisungen]
    [Returnanweisung]
}
```

Methoden können mit ihrem Namen in Programmanweisungen aufgerufen werden.

Beispiel für den Aufruf einer Methode `potenzieren()`, die der Definition in Abbildung 4.1 entspricht:

```
...
int wertA = 6;
int anzahl = 3;
int rueckgabe = potenzieren( wertA, anzahl );
...
```

Abbildung 4.2: Parameterübergabe by value



Über die Parameterliste können einer Methode Daten übergeben werden. Daten werden mit ihren Werten in methodeneigene Datenfelder übertragen (Übergabe „by value“ - siehe Abbildung 4.2). Eine Veränderung der Inhalte dieser Datenfelder innerhalb der Methode hat keinerlei Einfluß auf den Inhalt der ursprünglichen Datenfelder außerhalb der Methode.

Methoden können über die Return-Anweisung einen Rückgabewert bereitstellen. Der Datentyp des Rückgabewertes wird im Methodenkopf festgelegt. Wird kein Wert zurückgegeben, so ist als Datentyp des Rückgabewerts „void“ angegeben.

Beispiel für Methode ohne Rückgabewert:

```
public static void bildschirmAusgabe() {
    System.out.println("Hallo_Welt");
}
```

Grundsätze (zur gefälligen Beachtung) bei der Definition von Methoden:

1. Jede Methode erfüllt nur eine abgegrenzte Funktion („Keep it simple“).
2. Der Methodename beschreibt die Funktion der Methode.
3. Methodennamen beginnen mit einem Kleinbuchstaben, neue Wortteile werden mit einem Großbuchstaben begonnen (z.B. `diesIstEineTolleMethode()`);
4. Eine Methode besteht aus einer überschaubaren Anzahl von Anweisungen.
5. Der Rücksprung aus einer Methode erfolgt nur am Ende des Methodenblocks.

4.2.2 Vergleichsoperatoren

Operator	Bedeutung	Beispiel
==	gleich	<code>boolean ergebnis = (wertA == wertB);</code>
!=	ungleich	<code>boolean ergebnis = (wertA != wertB);</code>
<	kleiner als	<code>boolean ergebnis = (wertA < wertB);</code>
<=	kleiner oder gleich	<code>boolean ergebnis = (wertA <= wertB);</code>
>	größer als	<code>boolean ergebnis = (wertA > wertB);</code>
>=	größer oder gleich	<code>boolean ergebnis = (wertA >= wertB);</code>

4.2.3 Logische Operatoren (Boolsche Operatoren)

Operator	Bedeutung	Beispiel
&&	Und-Verknüpfung	<code>boolean x = (wertA == wertB) && (wertA > wertC);</code>
	Oder-Verknüpfung	<code>boolean x = (wertA != wertB) (wertB > wertC);</code>
!	Logische Umkehrung	<code>boolean x = !(wertA < wertB);</code>

4.2.4 Bedingte Verzweigung mit if-else

Eine unverzichtbare Kontrollstruktur in jeder Programmiersprache ist die bedingte Verzweigung (vgl. nebenstehende Abbildung). Abhängig von der Erfüllung oder Nichterfüllung einer bestimmten Bedingung werden unterschiedliche Anweisungen ausgeführt.

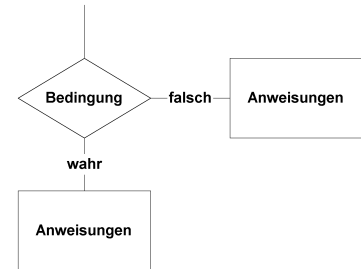
In Java kann eine bedingte Verzweigung mit Hilfe der `if-else`-Struktur und den verfügbaren Vergleichsoperatoren realisiert werden.

Beispiel einer bedingten Verzweigung:

```

1 ...
2 int wert = 4;
3
4 if( wert > 3 ) {
5   System.out.println("Der_gesuchte_Wert_ist_groesser_als_3");
6 }
7 else {
8   System.out.println("Der_gesuchte_Wert_ist_nicht_groesser_als_3");
9 }
10 ...

```



Die Bedingungsprüfung (hier Zeile 4: `wert > 3`) muss als Ergebnis `true` oder `false` (Datentyp `boolean`!) ergeben. Der `else`-Block (hier Zeile 7 bis 9) kann ggfs. entfallen.

Mehrere sich gegenseitig ausschließende Bedingungsabfragen können wie folgt gestaffelt werden:

```

1 ...
2 int wert = 4;
3
4 if( wert > 4 ) {
5   System.out.println("Der_Inhalt_des_Datenfeldes_ist_groesser_als_4");
6 }
7 else if( wert > 3 ){
8   System.out.println("Der_Inhalt_des_Datenfeldes_ist_groesser_als_3");
9 }
10 else if( wert > 2 ) {
11   System.out.println("Der_Inhalt_des_Datenfeldes_ist_groesser_als_2");
12 }
13 else {
14   System.out.println("Der_Inhalt_ist_nicht_groesser_als_2");
15 }
16 ...

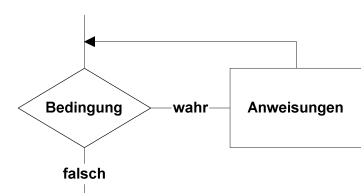
```

4.2.5 Schleifen mit while

Eine weitere wichtige Kontrollstruktur stellt die bedingte Wiederholung (Schleife) dar (vgl. nebenstehende Abbildung). Abhängig von der Erfüllung oder Nichterfüllung einer Bedingung werden bestimmte Anweisungen mehrfach ausgeführt.

In Java kann eine bedingte Wiederholung unter anderem mit Hilfe der `while`-Struktur und den verfügbaren Vergleichsoperatoren realisiert werden.

Beispiel einer bedingten Wiederholung:



```

1 ...
2 int wert = 0;
3
4 while( wert < 2 ) {
5   System.out.println("Anweisung_wird_ausgefuehrt.");
6   wert++;
7 }
8 ...

```

Die Bedingungsprüfung (hier Zeile 4: `wert < 2`) muss als Ergebnis `true` oder `false` (Datentyp `boolean`!) liefern. Solange das Ergebnis der Bedingungsprüfung `true` lautet, wird der folgende Anweisungsblock ausgeführt.

4.2.6 Weitere Kontrollstrukturen

Mit `if-else` und `while` können alle Programmstrukturen umgesetzt werden. Java bietet darüber hinaus mit der `switch-case`-Verzweigung, der `for`-Schleife und der `do-while`-Schleife drei weitere Kontrollstrukturen, die für bestimmte Problemstellungen eine komfortablere Lösung darstellen.

Verzweigungen mit `switch-case`

Struktur:

```

switch( ganzzahliger Ausdruck ) {
  case wert1:
    Anweisungen;
    break;
  case wert1:
    Anweisungen;
    break;
  ...
  default:
    Anweisungen;
}

```

Beispiel:

```

1 switch( wert ) {
2   case 'C':
3     System.out.print(
4       "Buchstabe_C");
5     break;
6   case 'D':
7     System.out.print(
8       "Buchstabe_D");
9     break;
10  ...
11  default:
12    System.out.print(
13      "Kein_gueltiger_Wert");
14 }

```

Schleifen mit `for`

Struktur:

```

for( Startanweisungen;
  Bedingungspruefung;
  Schleifenendanweisungen ) {
  Anweisungen;
}

```

Beispiel:

```

1 for( int i = 0; i < grenzWert; i++) {
2   System.out.println(
3     "Schleifendurchlauf_" + (i+1) );
4 }

```

Schleifen mit do-while

Struktur:

```
do {  
    Anweisungen;  
} while ( Bedingungspruefung );
```

Beispiel:

```
1 int i = 0;  
2  
3 do {  
4     System.out.println("i=_ " + i);  
5     i++;  
6 } while ( i < 100 );
```

Schleifenabbruch mit break und continue

Mit der Anweisung `continue`; kann man einen Schleifendurchgang vorzeitig beenden und den nächsten Schleifendurchgang bereits vor dem Durchlauf des gesamten Schleifendurchgangs beginnen.

Mit `break`; kann man eine Schleife oder einen `case`-Block verlassen und zum Statement nach Ende der Schleife bzw. des `switch`-Blocks springen.

Mit `labelname`: kann man eine Sprungmarke vor ein `for`-, `while`- oder `do`-Statement setzen und dann mit `continue labelname`; bzw. `break labelname`; eine mehrfach geschachtelte Schleife vorzeitig beenden und zur Ebene der Sprungmarke springen.

4.3 Lösungen

1. Die um die Methode hochVier() ergänzte Applikation kann folgendes Aussehen haben:

```
1 public class Potenzierung2 {
2
3     public static long hochZwei( long wert ) {
4         long resultat = wert * wert;
5         return resultat;
6     }
7
8     public static long hochDrei( long wert ) {
9         long resultat = hochZwei(wert) * wert;
10        return resultat;
11    }
12
13    public static long hochVier( long wert ) {
14        return hochDrei( wert ) * wert;
15    }
16
17    public static void main( String[] args ) {
18        long wertA = 3;
19
20        long wertB = hochZwei( wertA );
21        long wertC = hochDrei( wertA );
22        long wertD = hochVier( wertA );
23
24        System.out.println( wertA + "_hoch_Zwei=_ " + wertB );
25        System.out.println( wertA + "_hoch_Drei=_ " + wertC );
26        System.out.println( wertA + "_hoch_Vier=_ " + wertD );
27    }
28 }
```

2. Die Applikation „Taschenrechner“ kann folgendes Aussehen haben:

```
1 import javakurs.io.*;
2
3 public class Taschenrechner {
4
5     public static double addiere( double a, double b ) {
6         double ergebnis = a + b;
7         return ergebnis;
8     }
9
10    public static double subtrahiere( double a, double b ) {
11        double ergebnis = a - b;
12        return ergebnis;
13    }
14
15    public static double multipliziere( double a, double b ) {
16        double ergebnis = a * b;
17        return ergebnis;
18    }
19    public static double dividiere( double a, double b ) {
20        double ergebnis = a / b;
21        return ergebnis;
22    }
23
24    public static void main( String[] args ) {
25
26        Eingabe.read("Geben_Sie_eine_Formel_ein:");
27
28        double wertA = Eingabe.getDouble(0);
29        double wertB = Eingabe.getDouble(2);
30
31        char operator = Eingabe.getChar(1);
32
33        double ergebnis = 0;
34
35        if( operator == '+' ){
36            ergebnis = addiere( wertA, wertB );
37        }
38        else if( operator == '-' ) {
39            ergebnis = subtrahiere( wertA, wertB );
40        }
41        else if( operator == '*' ) {
42            ergebnis = multipliziere( wertA, wertB );
43        }
44        else if( operator == '/' ) {
45            ergebnis = dividiere( wertA, wertB );
46        }
47        else {
48            Ausgabe.println("Keinen_gueltigen_Operator_gefunden");
49        }
50
51        Ausgabe.println( wertA + "_" + operator + "_" + wertB +
52            "_=" + ergebnis);
53    }
54 }
```

3. Die Applikation „PotenzenApp“ kann folgendes Aussehen haben:

```
1 import javakurs.io.*;
2
3 public class PotenzenApp {
4
5     /**
6     * Gibt das Ergebnis der Operation eingabeWert^n auf dem
7     * Bildschirm aus.
8     */
9     public static void bildschirmAusgabe( long eingabeWert, int n ){
10         long resultat = potenzieren( eingabeWert, n );
11
12         if( resultat < 0 ) {
13             Ausgabe.println(eingabeWert + "^" +
14                 n + "_konnte_nicht_ermittelt_werden.");
15         }
16         else {
17             Ausgabe.println(eingabeWert + "^" +
18                 n + "_ergibt_" + resultat + "." );
19         }
20     }
21
22     /**
23     * Ermittelt Potenzen im Bereich 0...n
24     */
25     public static long potenzieren( long eingabeWert, int n ){
26
27         long resultat = -1;
28
29         if( n >= 0 ){
30             // Ausgangswerte
31             resultat = 1;
32             int i = 1;
33
34             while( i <= n ){
35                 resultat = resultat * eingabeWert;
36                 i++;
37             }
38         }
39         return( resultat );
40     }
41
42     public static void main( String[] args ) {
43
44         // Einlesen der Benutzereingaben
45         Eingabe.read("Potenzierung_-_Geben_Sie_Basiswert_" +
46             "und_obere_Grenze_ein:");
47
48         long basis = Eingabe.getInt(0);
49         int grenze = Eingabe.getInt(1);
50         int n = 0;
51
52         // Schleife bis zur Erreichung des Grenzwertes
53         while( n < grenze ) {
54             bildschirmAusgabe( basis, n );
55             n++;
56         }
57     }
58 }
```

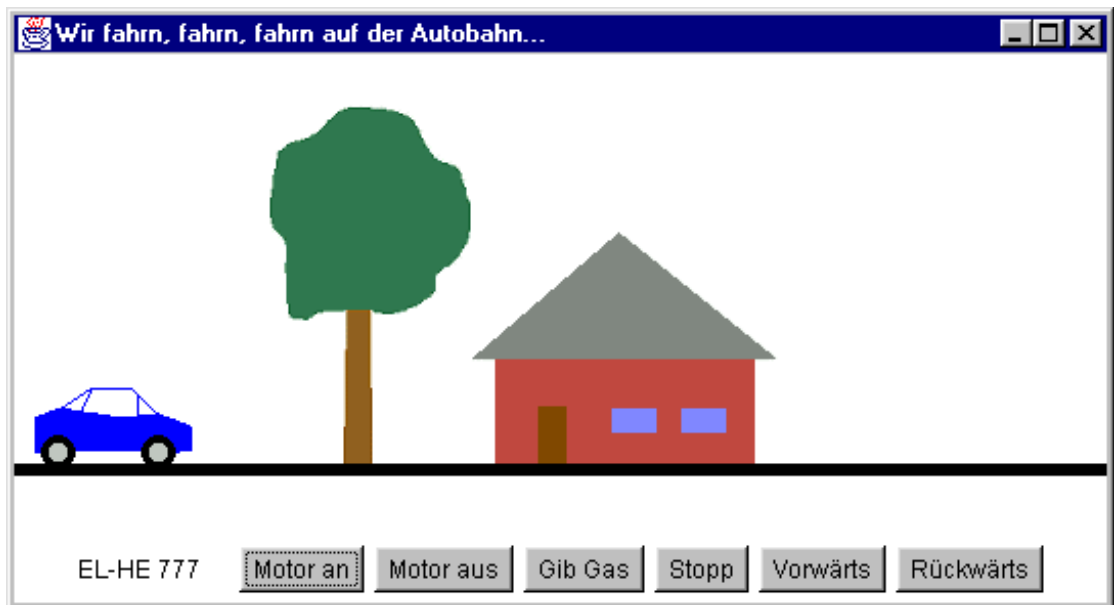

4. Die Applikation „Fakultaeten“ kann folgendes Aussehen haben:

```
1 import javakurs.io.*;
2
3 public class Fakultaeten {
4
5     /**
6     * Gibt das Ergebnis der Operation n! auf dem
7     * Bildschirm aus.
8     */
9     public static void bildschirmAusgabe( int n ){
10         long resultat = fakultaetErmitteln( n );
11
12         if( resultat < 0 ) {
13             Ausgabe.println(n + "!" +
14                 "_konnte_nicht_ermittelt_werden.");
15         }
16         else {
17             Ausgabe.println(n + "!" +
18                 "_ergibt_" + resultat + ".");
19         }
20     }
21
22     /**
23     * Ermittelt Fakultaeten.
24     */
25     public static long fakultaetErmitteln( int n ){
26
27         long resultat = 1;
28         int aktuellerWert = 1;
29
30         if( n > 0 ) {
31             while( aktuellerWert <= n ){
32                 resultat = resultat * aktuellerWert;
33                 aktuellerWert++;
34             }
35         }
36         else {
37             resultat = -1;
38         }
39
40         return( resultat );
41     }
42
43     public static void main( String[] args ) {
44         // Eingabe
45         Eingabe.read("Fakultaet_n!_-Geben_Sie_den_Basiswert_n_ein:");
46         int n = Eingabe.getInt(0);
47
48         // Verarbeitung und Ausgabe
49         int i = 1; // Zaehlerfeld
50         while( i <= n ) {
51             bildschirmAusgabe( i );
52             i++;
53         }
54     }
55 }
```

Übung 5

Objekte machen Java mächtig

5.1 Aufgaben



Java zeichnet im Vergleich zu anderen Programmiersprachen vor allem eines aus:

Java ist durch und durch objektorientiert.

Es ist daher dringend geboten, dass wir jetzt an einem Beispiel mit dem Konzept der objektorientierten Programmierung vertraut werden. Erst dann kann Java für uns seine volle Kraft entfalten.

Als Beispiel für objektorientierte Anwendungsprogrammierung soll eine Straßenszene nach dem Vorbild der obenstehenden Abbildung programmiert werden.

Keine Angst - es ist leichter, als der erste Blick auf das Ziel ahnen läßt. Im Kapitel Hilfen finden Sie eine schrittweise Anleitung zur Lösung der Aufgabe.

Ergänzende Aufgaben:

1. Im Paket `javakurs.io` finden Sie die Klasse `AusgabeFenster`. Benutzen Sie ein Objekt dieser Klasse zur Ausgabe der Lösungen in der Aufgabe 2 des vorangehenden Kapitels (Taschenrechner).
2. Erzeugen Sie in einer Applikation mehrere Objekte vom Typ `AusgabeFenster` und benutzen Sie die Objekte selektiv für die Textausgabe.
3. Zur Schnittstelle der Klasse `AusgabeFenster` gehören die Methoden `setTextColor()` und `setFont()`. Benutzen Sie diese Methoden, um die Farbe und die Schriftart der Textausgabe in Objekten vom Typ `AusgabeFenster` zu verändern.

4. Schreiben Sie eine Applikation, die einen Textstring mit der Klassenmethode `Eingabe.read()` einliest und anschließend den Textstring komplett
- (a) in Kleinbuchstaben,
 - (b) in Großbuchstaben und
 - (c) in umgekehrter Zeichenfolge
- in einem Objekt vom Typ `AusgabeFenster` ausgibt.
Hilfe: Suchen Sie in der Dokumentation zur Klasse `String` nach geeigneten Methoden.

Unterstützende Materialien:

- 5.2 Hilfen - Anleitung Schritt für Schritt
- 5.3 Hinweise - Klassen und Objekte bedingte Wiederholungen
- 5.4 Dokumentationen
- 5.5 Lösungen

5.2 Hilfen

Hier nun die Anleitung zur Aufgabenlösung Schritt für Schritt.

1. Schreiben Sie zunächst nach dem bekannten „Strickmuster“ das Grundgerüst einer Java-Applikation.

```
1 public class StrassenSzene {
2     public static void main( String[] args ) {
3
4     }
5 }
```

2. Schaffen Sie nun innerhalb der Methode `main()` ein Programmfenster mit der Abbildung einer Straße. Hierzu können Sie die fertige Klasse `Strasse` benutzen, die im Unterverzeichnis „javakurs\auto“ innerhalb der Datei „Strasse.class“ bereitliegt (`javakurs.auto.Strasse`).

Der Dokumentation für die Klasse `Strasse` können Sie entnehmen, dass die Klasse über einen Konstruktor `public Strasse()` verfügt.

Den Konstruktor benutzen Sie, um eine Instanz aus der Klasse `Strasse` zu erzeugen.

```
new Strasse();
```

Mit dieser Anweisung wird eine neue Instanz (im Folgenden auch kurz als „Objekt“ bezeichnet) vom Typ `Strasse` deklariert. Um einen dauerhaften Zugriff auf dieses Objekt zu erhalten, sollten Sie außerdem noch eine Referenz auf das Objekt in einem Datenfeld ablegen. Der Konstruktor liefert bei der Deklaration eines jeden Objekts die Referenz auf das Objekt (Zugriffsadresse) zurück.

```
Strasse strassenRef = new Strasse();
```

Über die im Datenfeld „strassenRef“ abgelegte Referenz können Sie im weiteren Programmablauf auf das Objekt zugreifen.

Wichtig: Ohne Speicherung der Referenz in einem Datenfeld ist das Objekt für Sie anonym, d.h. im weiteren Verlauf der Applikation nicht ansprechbar.

Die Benennung des Datenfeldes (hier: „strassenRef“) ist Sache des Programmiers. Machen Sie sie möglichst selbsterklärend, wählen Sie „sprechende“ Bezeichner.

Ihr Quelltext hat nun das folgende Aussehen:

```
1 public class StrassenSzene {
2     public static void main( String[] args ) {
3         Strasse strassenRef = new Strasse();
4     }
5 }
```

3. Die Dokumentation der Klasse `Strasse` beschreibt außerdem die Schnittstellen-Methoden (Public-Methoden), über die auf Objekte vom Typ `Strasse` Einfluß genommen werden kann.

Die Methode `setVisible()` sorgt dafür, dass ein Objekt vom Typ `Strasse` auf dem Bildschirm abgebildet wird. Schnittstellen-Methoden benutzen Sie nach dem folgenden Muster:

```
objektReferenz.methodenName( parameterliste );
```

Die Methode `setVisible()` erwartet gemäß Dokumentation einen Wert vom Typ `boolean` als Parameter. Fügen Sie also die Anweisung

```
strassenRef.setVisible( true );
```

der `main`-Methode hinzu.

Quelltext nunmehr:

```

1 public class StrassenSzene {
2     public static void main( String[] args ) {
3         Strasse strassenRef = new Strasse();
4         strassenRef.setVisible( true );
5     }
6 }

```

4. Bevor Sie nun den Quelltext kompilieren können, müssen Sie noch sicherstellen, dass die Klasse `Strasse` vom Kompiler und von der Java Virtual Machine gefunden werden.

Der bequemste Weg ist, die Anweisung

```
import javakurs.auto.Strasse;
```

oder noch besser

```
import javakurs.auto.*;
```

an den Anfang des Quelltextes zu stellen. Mit der ersten Variante stellen Sie sicher, dass die Datei „`Strasse.class`“ gefunden wird. Mit der zweiten Variante stellen Sie sicher, dass alle im Verzeichnis „`javakurs\auto`“ gespeicherten Class-Dateien bei Bedarf gefunden werden.

Quelltext:

```

1 import javakurs.auto.*;
2
3 public class StrassenSzene {
4     public static void main( String[] args ) {
5         Strasse strassenRef = new Strasse();
6         strassenRef.setVisible( true );
7     }
8 }

```

5. Kompilieren Sie nun den Quelltext und testen Sie die Applikation.
6. Noch ist die Applikation wenig eindrucksvoll. Doch was an dieser Stelle viel wichtiger ist - das grundlegende Prinzip objektorientierter Anwendungsprogrammierung in Java haben Sie erfolgreich angewendet. Aus einer bereits vorhandenen Klasse haben Sie ein funktionfähiges Objekt erzeugt und mit der Speicherung der Referenz auf das Objekt in einem variablen Datenfeld den Zugriff auf die Methoden des Objekts für Ihr Projekt gesichert.

Wichtig: Halten Sie einen Moment ein und überlegen Sie, ob Sie die Begriffe „Klasse“, „Objekt“ und „Referenz“ eindeutig voneinander abgrenzen können (siehe dazu auch Abbildung 5.1).

7. Um nun ein Fahrzeug in die vorhandene Strassenszene einzufügen, wenden Sie das soeben Gelernte erneut an.

Der Dokumentation der Klasse `Strasse` können Sie entnehmen, dass zur Schnittstelle der Klasse eine Methode `add()` gehört, die die Funktion hat, Fahrzeuge in die Strassenszene einzufügen. Dieser Methode muß als Parameter eine Referenz auf ein Objekt vom Typ `Auto` übergeben werden.

Deklarieren Sie aus der verfügbaren Klasse `Auto` (die Datei „`Auto.class`“ befindet sich im Verzeichnis „`javakurs\auto`“) also zunächst durch Aufruf eines Konstruktors eine Instanz.

```
new Auto();
```

Vergessen Sie nicht, die Referenz auf die Objektinstanz in einem Datenfeld zu speichern.

```
Auto autoRef = new Auto();
```

Fügen Sie nun das neu erschaffene `Auto` der Strassenszene hinzu, indem Sie die Referenz auf das `Auto`-Objekt an die Methode `add()` übergeben.

```
strasseRef.add( autoRef );
```

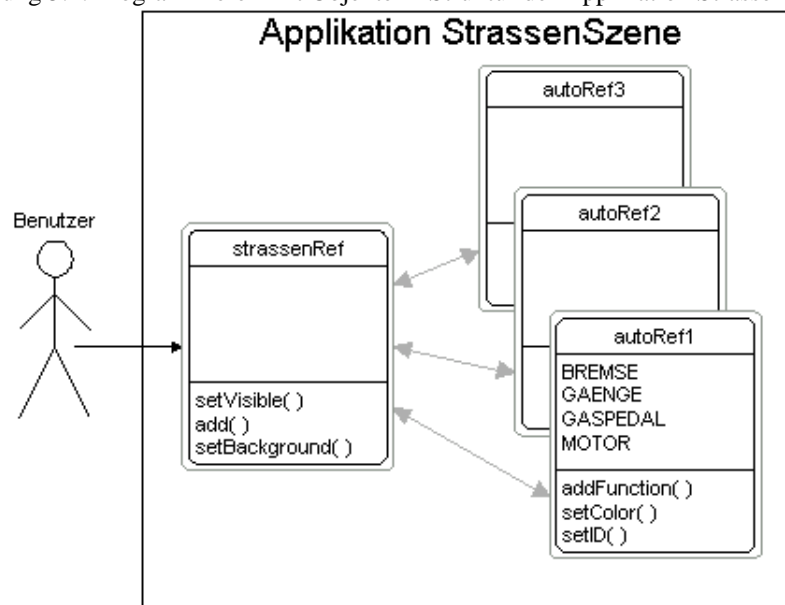
Quelltext:

```

1 import javakurs.auto.*;
2
3 public class StrassenSzene {
4     public static void main( String[] args ) {
5         Strasse strassenRef = new Strasse();
6
7         Auto autoRef = new Auto();
8         strassenRef.add( autoRef );
9
10        strassenRef.setVisible( true );
11    }
12 }

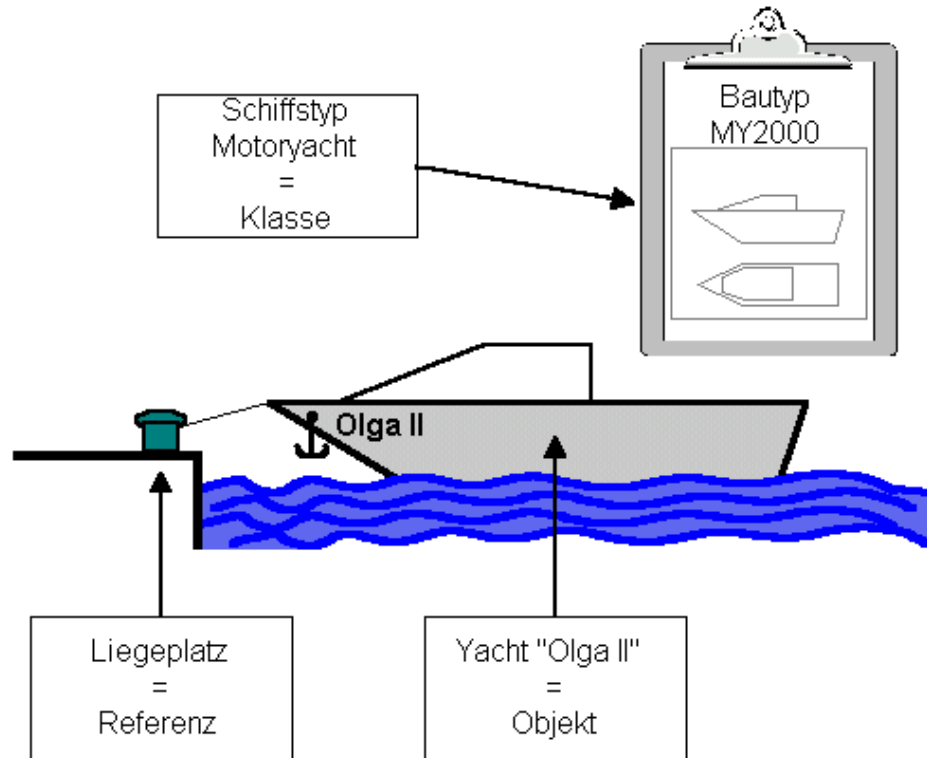
```

8. Kompilieren und testen Sie die Applikation.
9. Um das Auto in Bewegung zu setzen, müssen Sie mit Hilfe der Schnittstellen-Methode `addFunction()` das Objekt mit entsprechenden Fähigkeiten ausstatten. Die Dokumentation zur Klasse `Auto` gibt Hinweise, wie Sie vorgehen müssen.
10. Es fehlt noch eine schöne Landschaftskulisse. In der Klasse `Strasse` ist die Methode `setBackgroundImage()` definiert. Diese können Sie benutzen, um ein Bild Ihrer Wahl einzufügen.
11. Sie können zur Übung weitere Objekte vom Typ `Auto` deklarieren und sie ebenfalls in die Straßenszene einfügen.

Abbildung 5.1: Programmieren mit Objekten - Struktur der Applikation `StrassenSzene`

5.3 Hinweise

Abbildung 5.2: Klasse, Objekt und Referenz veranschaulicht am Beispiel einer Motoryacht



5.3.1 Deklaration einer Objektinstanz

Grundlage für die Deklaration eines Objekts bildet eine Klasse (Class). Jede Java-Implementierung stellt in der Standard-Klassenbibliothek eine Fülle von fertigen Klassen zur Verfügung, aus denen Objekte erzeugt werden können (siehe hierzu die Dokumentation zum JDK von Sun). Darüber hinaus können Sie eigene Klassen und Klassen von Drittanbietern in ihren Projekten nutzen.

In einer Klasse sind alle Eigenschaften und Fähigkeiten eines Objekttyps in Form von Datenfeldern und Methoden definiert. Die Klasse stellt somit den „Bauplan“ für die Erzeugung eines Objekts bereit.

Eine konkrete Instanz einer Klasse, das Objekt, wird durch Aufruf einer Konstruktormethode deklariert. Eine Klasse kann mehrere Konstruktoren aufweisen, die sich aber in der Liste der Übergabeparameter unterscheiden müssen.

Beispiel:

```
Konstruktoren: public Fahrzeug(){...}
                public Fahrzeug( int wert ){...}
                public Fahrzeug( int wertA, double wertB ){...}
```

Instanzen der Klasse `Fahrzeug` können im vorliegenden Fall somit in dreifacher Weise deklariert werden:

```
new Fahrzeug();
new Fahrzeug( 4 );
```

```
new Fahrzeug( 2, 30.5 );
```

5.3.2 Referenz auf ein Objekt

Unverzichtbar ist in der Regel, dass bei der Deklaration eines Objekts eine Referenz auf das Objekt in einem Datenfeld gespeichert wird.

```
Fahrzeug fRef = new Fahrzeug();
```

Über die Referenz kann im weiteren Programmablauf auf Schnittstellen-Methoden und Eigenschaften des Objekts zugegriffen werden. Zur Veranschaulichung siehe Abbildung 5.1.

Sie können ein Datenfeld, das erst zu einem späteren Zeitpunkt eine Referenz auf ein Objekt aufnehmen soll, auch vorab deklarieren. Weisen Sie einem solchen Datenfeld zunächst eine Null-Referenz zu.

```
Fahrzeug fRef = null;
```

5.3.3 Schnittstelle eines Objekts (Interface)

Die Eigenschaften und Funktionen eines Objekts können über die Schnittstellen-Methoden, die in der Dokumentation einer Klasse bekanntgemacht werden, angesprochen und beeinflusst werden. U. U. gehören auch Datenfelder zur Schnittstelle, obwohl der direkte Zugriff auf die Eigenschaften eines Objekts nach Möglichkeit verhindert werden sollte.

Methoden und Datenfelder der Objektschnittstelle werden durch den Modifizierer `public` gekennzeichnet.

Der Zugriff auf eine Objektmethode erfolgt nach der Notation:

```
objektReferenz.methodenName(parameter);
```

Beispiel: `fRef1.setAnzahlRaeder(4);`

Auf Datenfelder, die zur Schnittstelle eines Objekts gehören wird von aussen wie folgt zugegriffen:

```
objektReferenz.feldBezeichner
```

Beispiel: `fRef1.anzahlRaeder = 4;`

5.3.4 Zerstörung eines Objekts

Objekte, die im weiteren Programmablauf nicht mehr benötigt werden, werden von der Speicherverwaltung der Java Virtual Machine (Garbage Collector) automatisch zerstört, sobald der von ihnen belegte Speicherplatz anderweitig benötigt wird. Der Java-Programmierer braucht sich also um die Speicherverwaltung (anders z. B. als in C und C++) nicht zu kümmern.

5.3.5 Statische Methoden, statische Datenfelder

Grundsätzlich muss aus einer Klasse zunächst ein Objekt deklariert werden, bevor auf Methoden und Datenfelder der Objekt-Schnittstelle zugegriffen werden kann. Ausnahmen davon bilden Methoden und Datenfelder, die mit dem Modifizierer `static` belegt sind. Auf diese kann ohne Deklaration eines Objekts

direkt unter Benutzung des Klassennamens zugegriffen werden. Sie werden häufig als „Klassenmethoden“ bzw. „Klasseneigenschaften“ im Gegensatz zu „Objektmethoden“ bzw. „-eigenschaften“ bezeichnet.

Beispiel:

```
Konstruktor: public Double(double wert)
Datenfelder: public static double MAX_VALUE
              public static double MIN_VALUE
Methode:     public static double parseDouble( String s )
```

Die oben dokumentierte Methode `parseDouble()` und die Datenfelder `MAX_VALUE` und `MIN_VALUE` der Klasse `Double` (sie ist Bestandteil der Java-Standardbibliothek) können wie folgt benutzt werden:

```
1 double wertA = Double.parseDouble("2345.67");
2
3 System.out.println( wertA + "_liegt_zwischen_" + Double.MIN_VALUE +
4   "_und_" + Double.MAX_VALUE );
```

Möglich ist aber auch die folgende Verwendung:

```
1 Double dRef = new Double(4.0);
2 double wertA = dRef.parseDouble("2345.67");
3
4 System.out.println( wertA + "_liegt_zwischen_" + dRef.MIN_VALUE +
5   "_und_" + dRef.MAX_VALUE );
```

In der Regel wird man der ersten Variante den Vorzug geben. Häufig ist sogar die Deklaration eines Objekts für eine Klasse, die nur aus statischen Datenfeldern und Methoden besteht, dadurch unterbunden, dass der einzige Konstruktor nicht `public` sondern `private` gesetzt ist (siehe als Beispiel die Dokumentationen zur Klasse `Math` und zur Klasse `System`).

Statische Datenfelder und Methoden belegen nur einmal Platz im Arbeitsspeicher. Sie werden von allen deklarierten Objekten gemeinsam genutzt.

5.4 Dokumentation

5.4.1 Die Klasse `javakurs.auto.Strasse`

Die Klasse `javakurs.io.Strasse` stellt Konstruktoren und Methoden zur Verfügung, die eine elementare Straßenszene in einem Bildschirmfenster darstellen.

Konstruktoren

```
public javakurs.auto.Strasse( )
```

Erzeugt ein Objekt vom Typ `Strasse`.

Methoden

```
public void add( javakurs.auto.Auto autoref )
```

Fügt der Straßenszene ein Objekt vom Typ `Auto` hinzu. Diese Methode sollte erst aufgerufen werden, wenn das Objekt vom Typ `Auto` mit allen Funktionen versehen ist (siehe Methode `addFunction()` der Klasse `Auto`).

Parameter: `autoref` - Referenz auf ein Objekt vom Typ `Auto`

```
public void setBackgroundImage( java.lang.String source )
```

Setzt ein Hintergrundbild für die Straßenszene.

Parameter: `src` - Path oder URL zur Bilddatei (GIF oder JPEG)

```
public void setVisible( boolean value );
```

Macht die Straßenszene sichtbar oder verbirgt sie.

Parameter: `value` - `true` macht sichtbar, `false` macht unsichtbar

5.4.2 Die Klasse `javakurs.auto.Auto`

Die Klasse `javakurs.auto.Auto` stellt Konstruktoren, Konstanten und Methoden zur Verfügung, die ein Autoobjekt für die Verwendung in Verbindung mit einem Objekt vom Typ `javakurs.auto.Strasse` erzeugen.

Konstruktoren

```
public javakurs.auto.Auto( )
```

Erzeugt ein Objekt vom Typ `Auto`.

Eigenschaften/Felder

```
public static final short BREMSE
public static final short GAENGE
public static final short GASPEDAL
public static final short MOTOR
```

Konstanten, die in Verbindung mit der Methode `addFunction()` die ausführbaren Funktionen des Objekts festlegen.

Methoden

```
public void addFunction( short func )
```

Fügt eine bestimmte Funktion hinzu.

Parameter: func - Konstante Kennzahl für die Funktion.

Auto.BREMSE stattet das Objekt mit einer Bremsfunktion aus.

Auto.GAENGE stattet das Objekt mit Vorwärts- und Rückwärtsgang aus.

Auto.GASPEDAL stattet das Objekt mit einem Gaspedal aus.

Auto.MOTOR stattet das Objekt mit einem Motor aus.

```
public void setColor( java.lang.String farbe )
```

Legt die Farbe des Autos fest.

Parameter: farbe - String mit Farbe („gruen“, „rot“, „blau“)

```
public void setID( java.lang.String kennzeichen )
```

Fügt dem Auto eine Kennzeichnung hinzu.

Parameter: kennzeichen - Beliebiger Textstring

5.4.3 Die Klasse `javakurs.io.AusgabeFenster`

Die Klasse `javakurs.io.AusgabeFenster` stellt Konstruktoren und Methoden zur Verfügung, die die Ausgabe von Text in einem Bildschirmfenster ermöglichen.

Konstruktoren

```
public javakurs.io.AusgabeFenster( )
```

Erzeugt ein Objekt vom Typ `AusgabeFenster`.

Das Objekt wird erst durch den Aufruf der Methode `setVisible()` auf dem Bildschirm angezeigt. Positionierung und Größe entsprechen Standardwerten.

```
public javakurs.io.AusgabeFenster( int x, int y, int width, int height )
```

Erzeugt ein Objekt vom Typ `AusgabeFenster`.

Parameter: x - Horizontale Position des Fensters

y - Vertikale Position des Fensters

width - Breite des Fensters in Pixeln

height - Höhe des Fensters in Pixeln

Methoden

public void **addStatus**()

Fügt am Fuß des Fenster eine Statuszeile ein.

public void **clear**();

Löscht den Inhalt des Fensters.

public void **print**(*java.lang.String str*)

Gibt eine Textzeile aus. Es erfolgt kein Zeilenumbruch.

Parameter: *str* - Textstring mit Zeileninhalt

public void **println**(*java.lang.String str*)

Gibt eine Textzeile aus. Am Ende erfolgt ein Zeilenumbruch.

Parameter: *str* - Textstring mit Zeileninhalt

public void **setStatus**(*java.lang.String str*)

Gibt eine Meldung in der Statuszeile aus, wenn diese vorher durch `addStatus` erzeugt wurde.

Parameter: *str* - Textstring mit Zeileninhalt

public void **setTextColor**(*java.awt.Color c*)

Ändert die Farbe der Textausgabe. Die Änderung bezieht sich auf alle Textausgaben des Fensters.

Parameter: *c* - Referenz auf ein Objekt vom Typ `java.awt.Color`, das die Farbinformationen enthält

public void **setFont**(*java.awt.Font f*)

Ändert die Schriftart der Textausgabe. Die Änderung bezieht sich auf alle Textausgaben.

Parameter: *f* - Referenz auf ein Objekt vom Typ `java.awt.Font`, das Schriftart-Informationen enthält

public void **setVisible**(*boolean value*)

Zeigt das Fenster aus dem Bildschirm an oder setzt es unsichtbar.

Parameter: *value* - `true` zeigt das Fenster an, `false` verbirgt es

public void **sleep**(*int millisecs*)

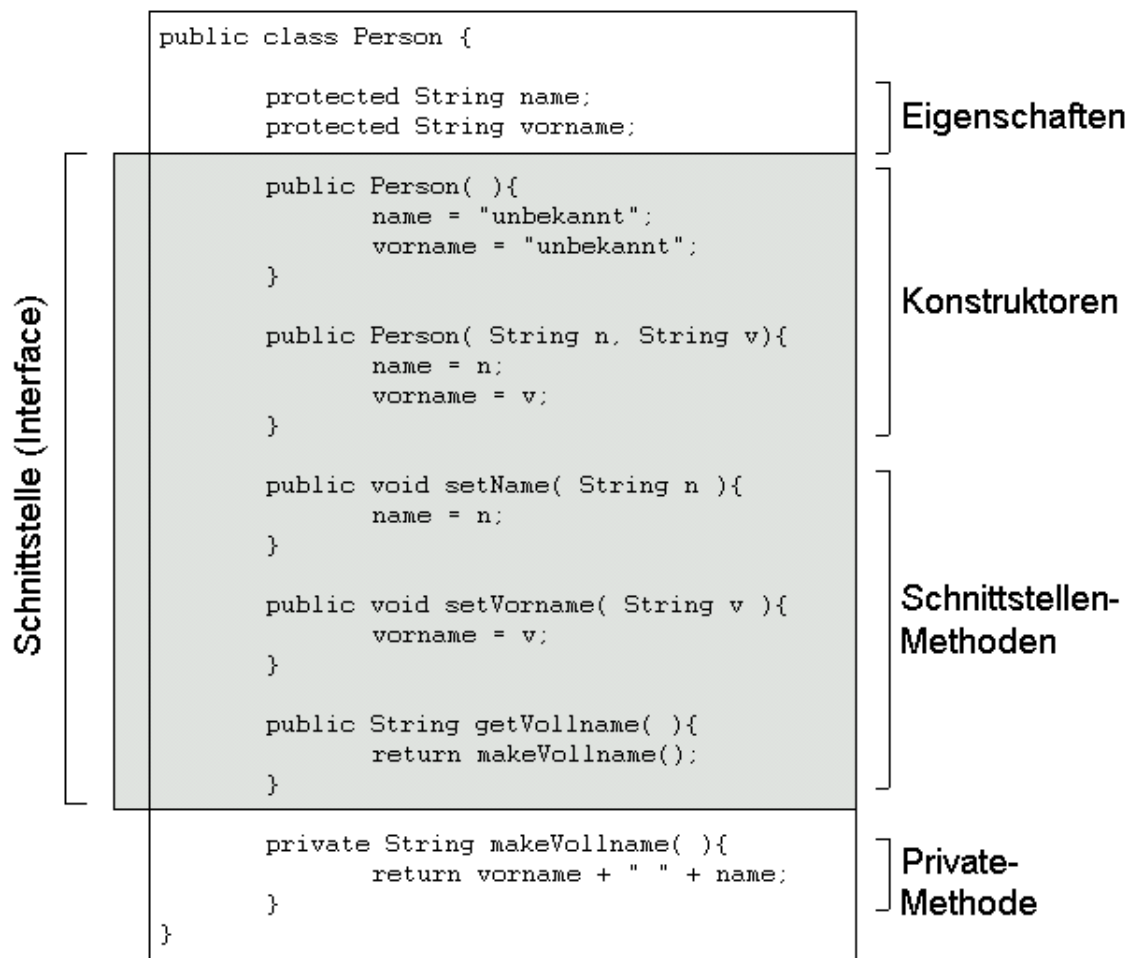
Unterbricht den Programmablauf.

Parameter: *millisecs* - Unterbrechungsdauer in Millisekunden

Übung 6

Do it yourself

6.1 Aufgaben



Oben sehen Sie ein Beispiel für die Definition einer Java-Klasse.

- Ergänzen Sie die abgebildete Klasse `Person` um die Eigenschaften Geburtsjahr, Geburtsmonat, Geburtstag, Geburtsort.
Definieren Sie geeignete Schnittstellen-Methoden (Set- und Get-Methoden), die auf diese Eigenschaften zugreifen.
Definieren Sie auch eine Methode

```
int berechneAlter( int yy, int mm, int dd ),
```

 welche für ein einzugebendes Stichtagsdatum das Alter der Person berechnet.
- Schreiben Sie eine Applikation, in der mehrere Objekte vom Typ `Person` deklariert und für die Bildschirmausgabe spezifischer Personendaten benutzt werden.

3. Schreiben Sie nach dem Muster der Klasse `Person` eine eigene Klasse `Fahrzeug` und testen Sie die Klasse mit einer geeigneten Applikation.
4. Schreiben Sie das Programm „HalloWelt“ objektorientiert.

Unterstützende Materialien:

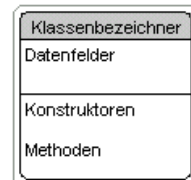
- 6.2 Hinweise - Struktur einer Klasse, Definition der Konstruktoren, Definition der Eigenschaften, Definition der Methoden, Kapselung, Schnittstelle, Selbstreferenzierung mit `this`, Pakete (Packages), Mehrere Klassen in einer Quelltextdatei, Innere Klassen, Applikationen objektorientiert schreiben
- 6.3 Lösungen

6.2 Hinweise

6.2.1 Struktur einer Klasse

Eine Klasse setzt sich zusammen aus den Elementkategorien

- Konstruktoren,
- Datenfelder und
- Methoden.



Konstruktoren werden bei der Erzeugung (Deklaration) eines Objekts aufgerufen.

Datenfelder und Methoden bilden die Eigenschaften und Fähigkeiten eines Objekttyps ab.

Durch die Modifizierer `public`, `protected` und `private` kann der Zugriff auf die Datenfelder und Methoden beeinflusst werden.

Es gilt demnach die folgende Zuordnung:

Objekt	Klasse	
	Elemente	Zugriffsbeschränkungen
Deklaration	Konstruktoren	public <kein Modifizierer> protected private
Eigenschaften	konstante und variable Datenfelder	public <kein Modifizierer> protected private
Fähigkeiten	Methoden	public <kein Modifizierer> protected private

6.2.2 Definition der Konstruktoren

Die Erzeugung (Deklaration) eines Objekts geschieht über den Aufruf eines Konstruktors. In Konstruktoren sollten nur notwendige Initialisierungen (Wertzweisung für Datenfeldern, Deklaration interner Objekte, Verbindungsaufbau zu Datenbanken u. ä.) vorgenommen werden.

Konstruktoren sind in der Bezeichnung immer identisch mit der Klasse, in der sie definiert werden.

Beispiel:

```
public class Person {
    public Person() {
        ...
    }
}
```

In einer Klasse können mehrere Konstruktoren definiert werden. Sie müssen sich in der Parameterliste unterscheiden.

Beispiel:

```
public class Person {  
  
    public Person(){  
        ...  
    }  
  
    public Person( String vname, String nname ){  
        ...  
    }  
}
```

Wird für eine Klasse kein Konstruktor definiert, so ist implizit ein leerer Standardkonstruktor festgelegt, mit dem aus der Klasse dann Objekte deklariert werden können.

Beispiel:

```
public class Person {  
    String vorname;  
    String nachname;  
  
    public void setName( String vname, String nname ) {  
        vorname = vname;  
        nachname = nname;  
    }  
}  
  
class PersonTester {  
    public static void main( String[] args ) {  
  
        Person p = new Person();  
        p.setName( "Otto", "Schmidt" );  
    }  
}
```

6.2.3 Definition der Eigenschaften

Die Eigenschaften eines Objekttyps werden in der Klasse in Form von konstanten und variablen Datenfeldern abgebildet. Die Datenfelder können als einfache Datentypen oder als Objekttypen jeder Art deklariert werden.

Der Zugriff auf Datenfelder ist immer nur innerhalb des durch geschweifte Klammern abgegrenzten Codeblocks möglich. Datenfelder, die außerhalb einer Methode definiert werden, sind dementsprechend global innerhalb der gesamten Klasse adressierbar.

Beispiel:

```
public class Person {  
  
    private String vorname;  
    private String nachname;  
  
    public Person(){  
        ...  
    }  
  
    public void setNachname( String name ) {
```



```
        nachname = name;
    }

    public void setVorname( String name ) {
        vorname = name;
    }

    public void setName( String vname, String nname ) {
        nachname = nname;
        vorname = vname;
    }
}
```

6.2.4 Definition der Methoden

Methoden bilden die Fähigkeiten eines Objekts ab. Grundsätzlich sollte nur über entsprechende Methoden (Set- bzw. Get-Methoden) ein Zugriff auf die Eigenschaften eines Objekt möglich sein (Data-Hiding).

In einer Klasse können mehrere Methoden mit gleicher Bezeichnung definiert werden. Sie müssen sich im Rückgabewert oder in der Parameterliste unterscheiden.

```
public class Person {
    ...

    public int berechneAlter ( int yy, int mm, int dd ){
        ...
    }

    public int berechneAlter ( Date current ) {
        ...
    }

    public int berechneAlter() {
        ...
    }
}
```

6.2.5 Kapselung, Schnittstelle

Nur auf Konstruktoren, Datenfelder und Methoden, die explizit durch den Modifizier `public` gekennzeichnet sind, kann vom Benutzer der Klasse unmittelbar zugegriffen werden.

Die Public-Elemente bilden die Schnittstelle (das Interface) der Klasse bzw. der aus der Klasse erzeugten Objekte. Nur über das Interface ist die Kommunikation mit einem Objekt möglich.

Elemente, die ohne Zugriffsmodifizierer definiert werden oder die mit den Modifizierern `protected` bzw. `private` gekennzeichnet sind, sind außerhalb der Klasse nur eingeschränkt oder gar nicht nutzbar (Näheres dazu in der folgenden Übung).

6.2.6 Selbstreferenzierung mit `this`

Mitunter ist es notwendig, dass ein Objekt Zugriff auf die eigene Referenz nimmt. Innerhalb einer Klasse kann der Bezeichner `this` benutzt werden, um eine solche Selbstreferenzierung vorzunehmen.

Beispiel:

```
public class Person {

    private String vname;
    private String nname;

    public Person( String vname, String nname ) {

        this.vname = vname; // Eindeutigkeit bei Bezeichnerkollision
        this.nname = nname; // wird durch 'this' hergestellt.
    }

    public String getVorname() {

        return this.vname; // Hier ist 'this' nicht unbedingt notwendig
    }

}
```

6.2.7 Pakete (Packages)

Zusammengehörige Klassen können in Paketen (Packages) gebündelt werden. Dazu muß in den jeweiligen Quelltextdateien als erste Anweisung `package <paketpfad>;` eingefügt werden.

Beispiel:

```
package meine.klassen;

public class Maus() {
    ...
}
```

Alle Class-Dateien des Package `meine.klassen` müssen in einem Unterverzeichnis „meine\klasse“ installiert werden. Der Java-Kompiler richtet ein solches Verzeichnis automatisch ein.

Für den Zugriff auf die Klassen eines Package müssen die Klassen entweder mit dem vollen Klassennamen (z.B. `meine.klassen.Maus`) adressiert werden oder es muss vor dem ersten Zugriff auf eine Package-Klasse eine Importanweisung erfolgen (z.B. `import meine.klassen.*;`).

6.2.8 Mehrere Klassen in einer Quelltextdatei

Ursprünglich (Java 1.0) galt für Java die Regel: Jede Klasse hat ihre eigene Quelltextdatei. Mehrere Klassen konnten nicht in einer Quelltextdatei definiert werden.

Seit der Java-Version 1.1 ist diese Beschränkung aufgehoben. Sie können in ein Quelltextdatei mehrere Klassen definieren.

Allerdings: Sie können nur die Klasse, die der Quelltextdatei den Namen gibt, mit dem Modifizierer `public` kennzeichnen. Nur diese Klasse ist damit uneingeschränkt für fremde Projekte nutzbar. Alle an-

deren in der Quelltextdatei definierten Klassen können nur innerhalb desselben Pakets (Package) benutzt werden.

Beispiel:

```
public class Maus() {
    ...
}

class MausWohnung() {
    ...
}
```

6.2.9 Innere Klassen

Innerhalb einer Klasse können weitere Klassen definiert werden (innere Klassen). Innere Klassen sind innerhalb der sie umgebenden Klasse (mit Ausnahme in Static-Methoden) wie jede andere Klasse adressierbar.

Beispiel:

```
public class Maus() {

    public Maus() {
        MausWohnung mw = new MausWohnung();
    }

    class MausWohnung() {
        ...
    }
}
```

Wollen Sie eine innere Klasse in einer Static-Methode der umgebenden Klasse oder in einer fremden Klasse benutzen, dann müssen Sie mittels einer Methode ein Objekt vom Typ der inneren Klasse erzeugen und eine Referenz darauf nach außen liefern (zum Vorgehen siehe Beispiel).

Beispiel:

```
public class Maus() {

    public Maus() {
        MausWohnung mw = new MausWohnung();
    }

    class MausWohnung() {
        ...
    }

    public Mauswohnung adresse() {
        return new Mauswohnung();
    }

    public static main( String[] args ) {

        Maus m = new Maus();
        Maus.Mauswohnung mw = m.adresse();
    }
}
```

Sie können auch innerhalb einer Methode oder innerhalb eines durch geschweifte Klammern abgegrenzten Codeblocks eine Klasse definieren. Diese ist nur innerhalb der Methode oder des Codeblocks nutzbar.

6.2.10 Applikationen objektorientiert schreiben

Schreiben Sie auch Applikationen möglichst objektorientiert! Wie? Hier kommt ein Muster:

```
public class OOApplikation{
    // Konstruktor
    public OOApplikation() {

    }

    public void init() {
        ...
    }

    public void start() {
        methode_1();
        ...
    }

    public void methode_1() {
        ...
    };

    ...

    public void methode_n() {
        ...
    }

    public static void main( String[] args ) {

        OOApplikation app = new OOApplikation();
        app.init();
        app.start();
    }
}
```

6.3 Lösungen

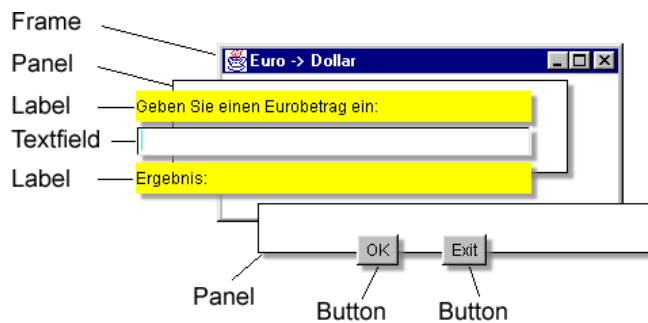
4. „Hallo Welt“ objektorientiert bedeutet zwar „mit Kanonen auf Spatzen schießen“ - als Übung ist es aber sicherlich sehr dienlich.

```
1 public class HalloWeltOO {
2
3   String message = "Hallo_Welt!!";
4
5   // Konstruktor
6   public HalloWeltOO() {
7   }
8
9   public void setMessage( String msg ) {
10    message = msg;
11  }
12
13  public String getMessage() {
14    return( message );
15  }
16
17  public void printMessage() {
18    System.out.println( getMessage() );
19  }
20
21  public static void main( String[] args ) {
22    // Objekt vom Typ HalloWeltOO deklarieren
23    HalloWeltOO appRef = new HalloWeltOO();
24    // Botschaft ausgeben
25    appRef.printMessage();
26  }
27 }
```

Übung 7

Benutzeroberfläche mit AWT

7.1 Aufgaben



1. Schreiben Sie eine Applikation, deren Benutzeroberfläche (GUI) aus einem Fenster mit zwei Schaltknöpfen (Buttons) besteht. Die Buttons tragen die Aufschrift „OK“ und „Exit“. Bei Betätigung des Exit-Buttons soll die Anwendung beendet und das Fenster geschlossen werden.



2. Erweitern Sie die Anwendung dahingehend, dass bei Betätigung des OK-Buttons eine Textausgabe im Fenster erfolgt. In der Textausgabe soll die Anzahl der bislang aufgelaufenen OK-Ereignisse gemeldet werden. Benutzen Sie für die Textausgabe ein Objekt vom Typ `java.awt.Label`.
3. Sorgen Sie dafür, dass sich die Button-Applikation auch durch Betätigung des X-Buttons (oben rechts) schließt. Hinweis: Sie müssen dazu beim Programmfenster (Objekttyp `java.awt.Frame`) ein Objekt vom Interfacetyp `WindowListener` registrieren, welches in der Methode `windowClosing()` einen Programmabbruch hervorruft.
4. Schreiben Sie einen Währungsrechner nach dem Vorbild der obigen Abbildung.
 - (a) Die Anwendung soll in der Lage sein, Eurobeträge in US-Dollarbeträge umzurechnen.
 - (b) Die Anwendung soll in der Lage sein, nach Wahl des Benutzers Eurobeträge in verschiedene andere Währungen umzurechnen. Mit einem Objekt vom Typ `java.awt.Choice` können Sie mehrere Alternativen zur Auswahl stellen.

Unterstützende Materialien:

- 7.2 Hinweise - Interfaces, AWT-Package
- 7.3 Lösungen

7.2 Hinweise

7.2.1 Interfaces

Interfaces legen Teile der Schnittstelle für eine Klasse fest, ohne dass der dazugehörige Programmcode zunächst definiert wird.

Beispiel:

```
public interface KlickListener {  
  
    public void klickPerformed();  
}
```

Im Beispiel wird die Schnittstellen-Methode `klickPerformed()` als verbindlich für Klassen festgelegt, die Bedingungen des Interfacetyps `KlickListener` erfüllen wollen.

Klassen, die ein bestimmtes Interface implementieren, müssen den Programmcode für alle Methoden der im Interface vordefinierten Schnittstelle bereitstellen.

Beispiel:

```
public class BeepApp implements KlickListener {  
  
    private void beep() {  
        ...  
    }  
  
    public void klickPerformed(){  
        beep();  
        beep();  
    }  
}
```

Im Beispiel wird die Methode `klickPerformed()` definiert, die als Schnittstelle für Objekte vom Typ `KlickListener` verbindlich festgelegt wurde.

7.2.2 Das AWT-Package

Grafische Benutzeroberflächen (Graphical User Interfaces - GUI) gehören zum Standard jeder Anwendung. Java unterstützt die Erstellung von GUIs u. a. durch das „Abstract Windowing Toolkit“ (AWT). Alle Klassen des AWT sind in dem Package „java.awt“ zusammengefasst.

GUI-Elemente im AWT

class Button	Schaltknöpfe (Buttons halt)
class Canvas	Zeichenfläche für die Ausgabe von Grafik
class Checkbox	Markierungsfeld
class CheckboxGroup	Fasst mehrere Checkbox-Elemente zusammen.
class Choice	Aufklappbare Optionsauswahl
class Dialog	Dialogfenster
class FileDialog	Dateiauswahl-Dialog
class Frame	Leeres Oberflächenfenster
class Label	Einzeiliges Textelement
class List	Auswahlliste
class Menu class MenuBar class MenuItem	Menüelemente
class Panel	Leeres Element zur Aufnahme weiterer Elemente
class TextArea	Mehrzeilige Texteingabe
class TextField	Einzeilige Texteingabe

Sie finden Quelltextbeispiele zum Gebrauch der Klassen in der Java-Dokumentation.

Layout-Management

Die Anordnung der GUI-Elemente innerhalb eines Frames oder eines Panels erfolgt in der Regel durch einen Layout-Manager, der dem Frame oder Panel zugeordnet wird. Das AWT stellt die Layout-Manager `BorderLayout`, `CardLayout`, `FlowLayout`, `GridBagLayout` und `GridLayout` bereit. In der API-Dokumentation zu Java finden Sie ausführliche Erläuterungen zu den verfügbaren Layout-Managern mit Abbildungen und Quelltext-Beispielen.

Anwendungsbeispiel:

```
import java.awt.*;

public class LayoutBeispiel{

    private Frame frameRef;

    public LayoutBeispiel() {

        frameRef = new Frame();
        FlowLayout lRef = new FlowLayout();
        frameRef.setLayout( lRef );
        ...
    }

    public static void main( String[] argv ) {
        LayoutBeispiel app = new LayoutBeispiel();
    }
}
```

```

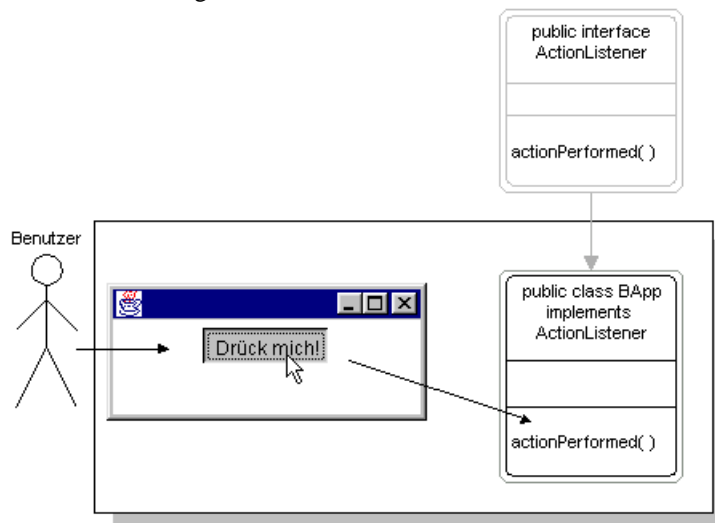
    }
}

```

Ereignisbehandlung

Das AWT-Package implementiert ein ereignisorientiertes Programmiermodell. Bei der Interaktion des Users mit den Elementen der Benutzeroberfläche werden von den GUI-Elementen (z.B. Buttons) Ereignisse (Events) ausgelöst. Diese können dann in der Anwendung mit sinnvolle Aktionen verbunden werden.

Abbildung 7.1: GUI-Element und ActionListener



Seit Java 1.1 ist empfohlen, zu diesem Zweck bei den GUI-Elementen sogenannte Listener-Objekte zu registrieren, an die die Verarbeitung der Events delegiert wird. Abhängig vom Typ des GUI-Elements können verschiedenartige Listener-Objekte registriert werden. Die Klasse `java.awt.Button` stellt z.B. eine Methode `addActionListener()` bereit, mit der Listener-Objekte vom Typ `ActionListener` registriert werden.

Die Listener-Objekte müssen spezifische Schnittstellenmethoden bereitstellen, an die das Event weitergeleitet wird. Ein `ActionListener`-Objekt muss z.B. die Methode `actionPerformed()` definieren, welcher als Parameter eine Referenz auf ein vom GUI-Element erzeugtes Objekt vom Typ `ActionEvent` übergeben wird.

Das GUI-Element leitet die Events an alle bei ihm registrierten Listener-Objekte durch Aufruf der entsprechenden Schnittstellen-Methode weiter (siehe Abbildung 7.1). In der Schnittstellen-Methode erfolgt die Verarbeitung des Events durch geeignete Anwendungsaktionen.

Die Listener-Typen des AWT-Packets sind im Package `java.awt.event` als Interfaces definiert.

Anwendungsbeispiel 1 - Applikation implementiert selbst das Interface ActionListener :

```
1 import java.awt.*;
2 import java.awt.event.*;
3
4 public class ListenerBeispiel1 implements ActionListener {
5
6     private Frame frameRef;
7     private Button buttonRef;
8
9     public ListenerBeispiel1() {
10         frameRef = new Frame();
11
12         buttonRef = new Button( "Exit" );
13         buttonRef.addActionListener( this );
14
15         frameRef.add( buttonRef );
16     }
17
18     public void run() {
19         frameRef.setSize(100,100);
20         frameRef.setVisible( true );
21     }
22
23     public void stop() {
24         System.exit(0);
25     }
26
27     public void actionPerformed( ActionEvent e ) {
28         stop();
29     }
30
31     public static void main( String[] args ) {
32         ListenerBeispiel1 app = new ListenerBeispiel1();
33         app.run();
34     }
35 }
```

Anwendungsbeispiel 2 - Innere Klasse stellt das Interface ActionListener bereit:

```
1 import java.awt.*;
2 import java.awt.event.*;
3
4 public class ListenerBeispiel2 {
5
6     private Frame frameRef;
7     private Button buttonRef;
8
9     public ListenerBeispiel2() {
10         frameRef = new Frame();
11
12         buttonRef = new Button( "Exit" );
13
14         ExitButtonListener ebl = new ExitButtonListener();
15         buttonRef.addActionListener( ebl );
16
17         frameRef.add( buttonRef );
18     }
19
20     public void run() {
21         frameRef.setSize(100,100);
22         frameRef.setVisible( true );
23     }
24
25     public void stop() {
26         System.exit(0);
27     }
28
29     public static void main( String[] args ) {
30         ListenerBeispiel2 app = new ListenerBeispiel2();
31         app.run();
32     }
33
34     class ExitButtonListener implements ActionListener {
35         public void actionPerformed( ActionEvent e ) {
36             stop();
37         }
38     }
39 }
```

Anwendungsbeispiel 3 - Public-Klasse stellt das Interface ActionListener bereit:

```
1 import java.awt.*;
2
3 public class ListenerBeispiel3 {
4
5     private Frame frameRef;
6     private Button buttonRef;
7
8     public ListenerBeispiel3() {
9         frameRef = new Frame();
10
11         buttonRef = new Button( "Exit" );
12
13         ExitButtonListener ebl = new ExitButtonListener( this );
14         buttonRef.addActionListener( ebl );
15
16         frameRef.add( buttonRef );
17     }
18
19     public void run() {
20         frameRef.setSize(100,100);
21         frameRef.setVisible( true );
22     }
23
24     public void stop() {
25         System.exit(0);
26     }
27
28     public static void main( String[] args ) {
29         ListenerBeispiel3 app = new ListenerBeispiel3();
30         app.run();
31     }
32 }
```



```
1 import java.awt.event.*;
2
3 public class ExitButtonListener implements ActionListener {
4
5     ListenerBeispiel3 parentApp;
6
7     public ExitButtonListener( ListenerBeispiel3 application ) {
8         parentApp = application;
9     }
10
11     public void actionPerformed( ActionEvent e ) {
12         parentApp.stop();
13     }
14 }
```

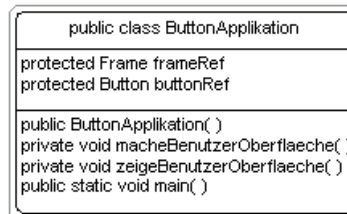
Ausgewählte Listener-Interfaces des AWT

ActionListener	Dient der Verarbeitung von Events, die durch bestimmte vordefinierte User-Aktionen ausgelöst werden (z.B. Drücken eines Buttons, Abschluss einer Texteingabe etc.). Ein <code>ActionListener</code> verarbeitet Events vom Typ <code>ActionEvent</code> .
ItemListener	Dient der Verarbeitung von Events, die ein bestimmtes Element einer Liste betreffen (z.B. bei Auswahllisten, Menüs etc.). Ein <code>ItemListener</code> verarbeitet Events vom Typ <code>ItemEvent</code> .
KeyListener	Dient der Verarbeitung einzelner Tastaturreignisse (z.B. Drücken einer bestimmten Tastaturtaste). Ein <code>KeyListener</code> verarbeitet Events vom Typ <code>KeyEvent</code> .
MouseListener	Dient der Verarbeitung von Mausaktionen durch Tastendruck. Ein <code>MouseListener</code> verarbeitet Events vom Typ <code>MouseEvent</code> .
MouseMotionListener	Dient der Verarbeitung von Mausbewegungen. Ein <code>MouseMotionListener</code> verarbeitet Events vom Typ <code>MouseMotionEvent</code> .
WindowListener	Dient der Verarbeitung von Events, die ein Oberflächenfenster betreffen (z.B. Verkleinerung zum Symbol, Schließen eines Fenster etc.). Ein <code>WindowListener</code> verarbeitet Events vom Typ <code>WindowEvent</code> .

Details zu den einzelnen Listener-Interfaces finden Sie in der Java-Dokumentation.

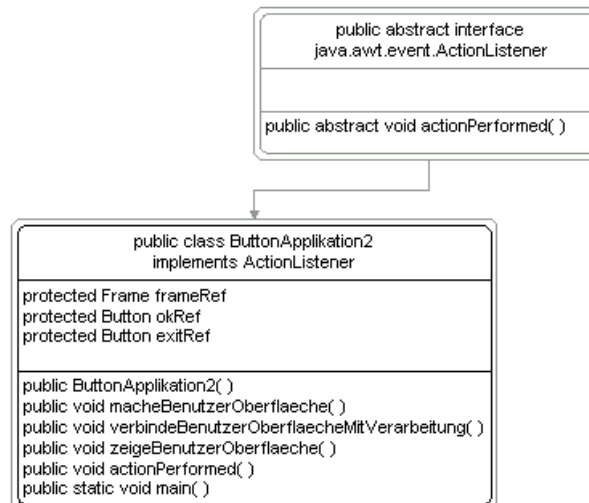
7.3 Lösungen

1. Zunächst ein Beispiel für ein Ein-Button-Fenster ohne Event-Auswertungen. Vorsicht dieses Fenster können Sie unter MS Windows nur mit Hilfe des Task-Manager schließen!



```
1 import java.awt.*;
2
3 public class ButtonApplikation {
4
5     protected Frame frameRef;
6     protected Button buttonRef;
7
8     public ButtonApplikation() {
9
10    }
11
12    private void macheBenutzerOberflaeche() {
13        frameRef = new Frame();
14        FlowLayout flRef = new FlowLayout();
15        frameRef.setLayout( flRef );
16
17        buttonRef = new Button();
18        buttonRef.setLabel("Drueck_mich!");
19
20        frameRef.add( buttonRef );
21    }
22
23    private void zeigeBenutzerOberflaeche() {
24        frameRef.setSize( 200, 100 );
25        frameRef.setLocation( 30,50 );
26        frameRef.setVisible( true );
27    }
28
29
30    public static void main( String args[] ) {
31        ButtonApplikation appRef = new ButtonApplikation();
32        appRef.macheBenutzerOberflaeche();
33        appRef.zeigeBenutzerOberflaeche();
34    }
35 }
```

Jetzt die eigentliche Lösung mit Event-Auswertung des Exit-Buttons. Die Applikationsklasse `ButtonApplikation2` implementiert selbst das Interface `ActionListener`, indem die Methode `actionPerformed()` definiert wird. In der Methode `verbindeBenutzerOberflaecheMitVerarbeitung()` wird das Applikationsobjekt beim Button als `ActionListener` angemeldet.



```

1 import java.awt.*;
2 import java.awt.event.*;
3
4 public class ButtonApplikation2 implements ActionListener {
5
6     protected Frame frameRef;
7     protected Button okRef;
8     protected Button exitRef;
9
10    /**
11     * Konstruktor - leer, deshalb nicht unbedingt noetig
12     */
13    public ButtonApplikation2() {
14    }
15
16    /**
17     * Erstellt die Benutzeroberflaeche
18     */
19    private void macheBenutzerOberflaeche() {
20        // Erzeuge Programmfenster
21        frameRef = new Frame();
22
23        // Setze Layoutmanager
24        FlowLayout flRef = new FlowLayout();
25        frameRef.setLayout( flRef );
26
27        // Erzeuge OKButton
28        okRef = new Button();
29        okRef.setLabel("OK");
30
31        // Erzeuge ExitButton
32        exitRef = new Button("Exit");
33
34        // Fuege Buttons in Programmfenster ein
35        frameRef.add( okRef );
  
```



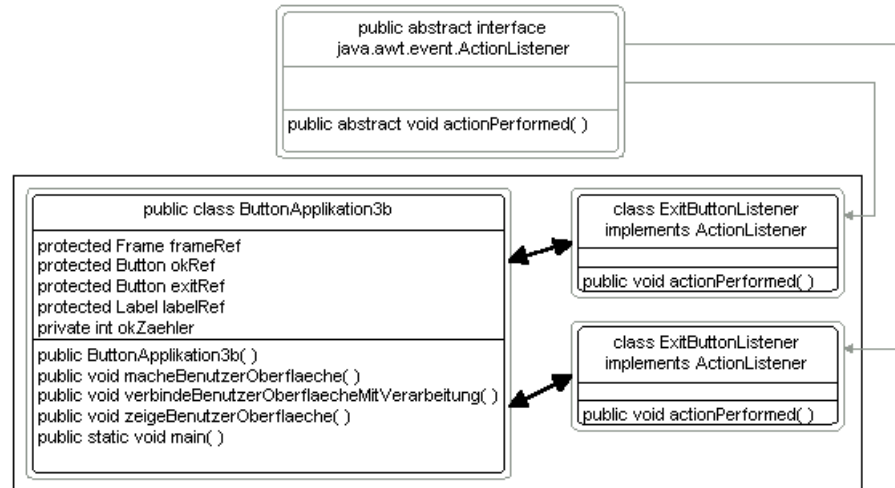
```
36     frameRef.add( exitRef );
37 }
38
39 /**
40  * Registriert die ActionListener fuer die GUI-Elemente
41  */
42 private void verbindeBenutzerOberflaecheMitVerarbeitung() {
43
44     // ExitButton wird mit diesem Objekt als Listener verbunden
45     exitRef.addActionListener( this );
46 }
47
48 /**
49  * Sorgt fuer die Anzeige der Benutzeroberflaeche
50  */
51 private void zeigeBenutzerOberflaeche() {
52
53     // Setze Fenstergroesse
54     frameRef.setSize( 200, 100 );
55
56     // Setze Fensterposition
57     frameRef.setLocation( 30,50 );
58
59     // Mache Fenster mit allen Elementen sichtbar
60     frameRef.setVisible( true );
61 }
62
63 /**
64  * ActionListener-Methode
65  */
66 public void actionPerformed( ActionEvent e ) {
67
68     // Bei Klick schliesse Programm
69     System.exit(0);
70 }
71
72 public static void main( String args[] ) {
73     ButtonApplikation2 appRef = new ButtonApplikation2();
74     appRef.macheBenutzerOberflaeche();
75     appRef.verbindeBenutzerOberflaecheMitVerarbeitung();
76     appRef.zeigeBenutzerOberflaeche();
77 }
78 }
```

2. Die Zwei-Button-Applikation ergänzt um ein Textausgabefeld.

```
1 import java.awt.*;
2 import java.awt.event.*;
3
4 public class ButtonApplikation3 implements ActionListener {
5
6     protected Frame frameRef;
7     protected Button okRef;
8     protected Button exitRef;
9     protected Label labelRef;
10
11     private int okZaehler;
12     /**
13      * Konstruktor
14      */
15     public ButtonApplikation3() {
16         okZaehler = 0;
17     }
18
19     /**
20      * Erstellt die Benutzeroberflaeche
21      */
22     public void macheBenutzerOberflaeche() {
23         // Erzeuge Programmfenster
24         frameRef = new Frame();
25
26         // Setze Layoutmanager
27         FlowLayout flRef = new FlowLayout();
28         frameRef.setLayout( flRef );
29
30         // Erzeuge OKButton
31         okRef = new Button();
32         okRef.setLabel("OK");
33
34         // Erzeuge ExitButton
35         exitRef = new Button("Exit");
36
37         // Erzeuge Textlabel
38         labelRef = new Label("OK-Klicks:_ " + okZaehler);
39
40         // Fuege Buttons in Programmfenster ein
41         frameRef.add( okRef );
42         frameRef.add( exitRef );
43         frameRef.add( labelRef );
44     }
45
46     /**
47      * Registriert die ActionListener fuer die GUI-Elemente
48      */
49     public void verbindeBenutzerOberflaecheMitVerarbeitung() {
50
51         // ExitButton wird mit diesem Objekt als Listener verbunden
52         exitRef.addActionListener( this );
53         // OKButton wird mit diesem Objekt als Listener verbunden
54         okRef.addActionListener( this );
55     }
56
57     /**
58      * Sorgt fuer die Anzeige der Benutzeroberflaeche
59      */
```

```
60 public void zeigeBenutzerOberflaeche() {
61
62     // Setze Fenstergroesse
63     frameRef.setSize( 200, 100 );
64
65     // Setze Fensterposition
66     frameRef.setLocation( 30,50 );
67
68     // Mache Fenster mit allen Elementen sichtbar
69     frameRef.setVisible( true );
70 }
71
72 /**
73  * ActionListener-Methode
74  */
75 public void actionPerformed( ActionEvent e ) {
76
77     if( e.getSource() == exitRef ) {
78         // Bei Klick schliesse Programm
79         System.exit(0);
80     }
81     else {
82         // erhoehe Zaehler und zeige neuen Stand
83         okZaehler++;
84         labelRef.setText("OK-Klicks_ " + okZaehler );
85     }
86
87 }
88
89 public static void main( String args[] ) {
90     ButtonApplikation3 appRef = new ButtonApplikation3();
91     appRef.macheBenutzerOberflaeche();
92     appRef.verbindeBenutzerOberflaecheMitVerarbeitung();
93     appRef.zeigeBenutzerOberflaeche();
94 }
95 }
```

Alternativlösung: Die Zwei-Button-Applikation ergänzt um ein Textausgabefeld. Diesmal werden zwei innere Klassen als Listener für die Buttons definiert.



```

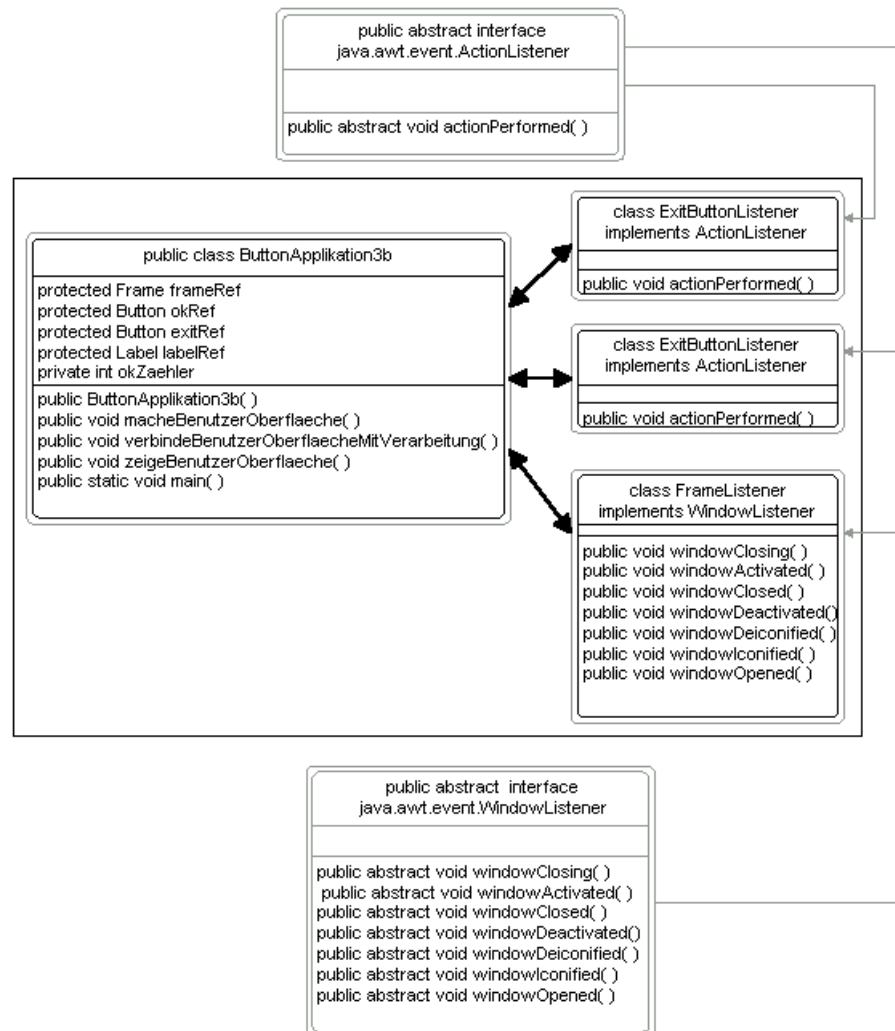
1 import java.awt.*;
2 import java.awt.event.*;
3
4 public class ButtonApplikation3b {
5
6     protected Frame frameRef;
7     protected Button okRef;
8     protected Button exitRef;
9     protected Label labelRef;
10
11     private int okZaehler;
12     /**
13      * Konstruktor
14      */
15     public ButtonApplikation3b() {
16         okZaehler = 0;
17     }
18
19     /**
20      * Erstellt die Benutzeroberflaeche
21      */
22     public void macheBenutzerOberflaeche() {
23         // Erzeuge Programmfenster
24         frameRef = new Frame();
25
26         // Setze Layoutmanager
27         FlowLayout flRef = new FlowLayout();
28         frameRef.setLayout( flRef );
29
30         // Erzeuge OKButton
31         okRef = new Button();
32         okRef.setLabel("OK");
33
34         // Erzeuge ExitButton
35         exitRef = new Button("Exit");
36
37         // Erzeuge Textlabel
38         labelRef = new Label("OK-Klicks:_" + okZaehler);
39

```

```
40     // Fuege Buttons in Programmfenster ein
41     frameRef.add( okRef );
42     frameRef.add( exitRef );
43     frameRef.add( labelRef );
44 }
45
46 /**
47  * Registriert die ActionListener fuer die GUI-Elemente
48  */
49 public void verbindeBenutzerOberflaecheMitVerarbeitung() {
50
51     // ExitButton wird mit Listener-Objekt verbunden
52     ExitButtonListener exitLRef = new ExitButtonListener();
53     exitRef.addActionListener( exitLRef );
54
55     // OKButton wird mit diesem Objekt als Listener verbunden
56     OKButtonListener okLRef = new OKButtonListener();
57     okRef.addActionListener( okLRef );
58 }
59
60 /**
61  * Sorgt fuer die Anzeige der Benutzeroberflaeche
62  */
63 public void zeigeBenutzerOberflaeche() {
64
65     // Setze Fenstergroesse
66     frameRef.setSize( 200, 100 );
67
68     // Setze Fensterposition
69     frameRef.setLocation( 30,50 );
70
71     // Mache Fenster mit allen Elementen sichtbar
72     frameRef.setVisible( true );
73 }
74
75 public static void main( String args[] ) {
76     ButtonApplikation3b appRef = new ButtonApplikation3b();
77     appRef.macheBenutzerOberflaeche();
78     appRef.verbindeBenutzerOberflaecheMitVerarbeitung();
79     appRef.zeigeBenutzerOberflaeche();
80 }
81
82 /**
83  * Innere Klasse als ActionListener fuer den Exit-Button
84  */
85 class ExitButtonListener implements ActionListener {
86     public void actionPerformed( ActionEvent e ) {
87         System.exit(0);
88     }
89 }
90
91 /**
92  * Innere Klasse als ActionListener fuer den OK-Button
93  */
94 class OKButtonListener implements ActionListener {
95     public void actionPerformed( ActionEvent e ) {
96
97         // erhoehe Zaehler und zeige neuen Stand
98         okZaehler++;
99         labelRef.setText("OK-Klicks_" + okZaehler );
```

```
100     }  
101     }  
102 }
```

3. Die gleiche Zwei-Button-Applikation wie zuvor. Diesmal wurde eine innere Klasse `FrameListener` zusätzlich definiert, die das Interface eines `WindowListener` implementiert. Ein Objekt vom Typ `FrameListener` wird beim Haupt-Frame registriert, um `WindowEvents` zu verarbeiten.



```

1 import java.awt.*;
2 import java.awt.event.*;
3
4 public class ButtonApplikation4 {
5
6     protected Frame frameRef;
7     protected Button okRef;
8     protected Button exitRef;
9     protected Label labelRef;
10
11     private int okZaehler;
12     /**
13      * Konstruktor
14      */
15     public ButtonApplikation4() {
16         okZaehler = 0;
17     }
18 }
  
```

```
19  /**
20  * Erstellt die Benutzeroberflaeche
21  */
22  public void macheBenutzerOberflaeche() {
23      // Erzeuge Programmfenster
24      frameRef = new Frame();
25
26      // Setze Layoutmanager
27      FlowLayout flRef = new FlowLayout();
28      frameRef.setLayout( flRef );
29
30      // Erzeuge OKButton
31      okRef = new Button();
32      okRef.setLabel("OK");
33
34      // Erzeuge ExitButton
35      exitRef = new Button("Exit");
36
37      // Erzeuge Textlabel
38      labelRef = new Label("OK-Klicks:_" + okZaehler);
39
40      // Fuege Buttons in Programmfenster ein
41      frameRef.add( okRef );
42      frameRef.add( exitRef );
43      frameRef.add( labelRef );
44  }
45
46  /**
47  * Registriert die ActionListener fuer die GUI-Elemente und
48  * registriert WindowListener beim Anwendungs-Frame
49  */
50  public void verbindeBenutzerOberflaecheMitVerarbeitung() {
51
52      // ExitButton wird mit Listener-Objekt verbunden
53      ExitButtonListener exitLRef = new ExitButtonListener();
54      exitRef.addActionListener( exitLRef );
55
56      // OKButton wird mit diesem Objekt als Listener verbunden
57      OKButtonListener okLRef = new OKButtonListener();
58      okRef.addActionListener( okLRef );
59
60      // Frame wird mit WindowListener verbunden
61      FrameListener frLRef = new FrameListener();
62      frameRef.addWindowListener( frLRef );
63  }
64
65  /**
66  * Sorgt fuer die Anzeige der Benutzeroberflaeche
67  */
68  public void zeigeBenutzerOberflaeche() {
69
70      // Setze Fenstergroesse
71      frameRef.setSize( 200, 100 );
72
73      // Setze Fensterposition
74      frameRef.setLocation( 30,50 );
75
76      // Mache Fenster mit allen Elementen sichtbar
77      frameRef.setVisible( true );
78  }
```

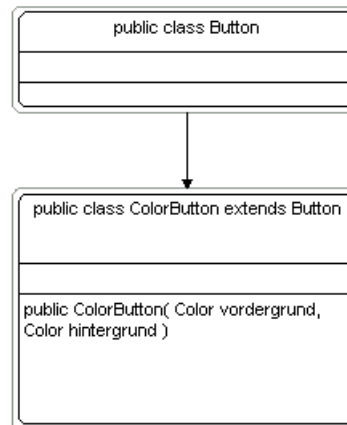


```
79
80 public static void main( String args[] ) {
81     ButtonApplikation4 appRef = new ButtonApplikation4();
82     appRef.macheBenutzerOberflaeche();
83     appRef.verbindeBenutzerOberflaecheMitVerarbeitung();
84     appRef.zeigeBenutzerOberflaeche();
85 }
86
87 /**
88  * Innere Klasse als ActionListener fuer den Exit-Button
89  */
90 class ExitButtonListener implements ActionListener {
91     public void actionPerformed( ActionEvent e ) {
92         System.exit(0);
93     }
94 }
95
96 /**
97  * Innere Klasse als ActionListener fuer den OK-Button
98  */
99 class OKButtonListener implements ActionListener {
100     public void actionPerformed( ActionEvent e ) {
101
102         // erhoehe Zaehler und zeige neuen Stand
103         okZaehler++;
104         labelRef.setText("OK-Klicks_" + okZaehler );
105     }
106 }
107 class FrameListener implements WindowListener {
108
109     // Wenn X-Button geklickt wird, dann beende Anwendung
110     public void windowClosing( WindowEvent event ){
111         System.exit(0);
112     }
113
114     // Keine Aktion definiert
115     public void windowActivated( WindowEvent event ){
116     }
117
118     // Keine Aktion definiert
119     public void windowClosed( WindowEvent event ){
120     }
121
122     // Keine Aktion definiert
123     public void windowDeactivated( WindowEvent event ){
124     }
125
126     // Keine Aktion definiert
127     public void windowDeiconified( WindowEvent event ){
128     }
129
130     // Keine Aktion definiert
131     public void windowIconified( WindowEvent event ){
132     }
133
134     // Keine Aktion definiert
135     public void windowOpened( WindowEvent event ){
136     }
137 }
138 }
```

Übung 8

Vererbung am Beispiel farbiger Buttons

8.1 Aufgaben



Sind Sie das graue Einerlei der Windowsbuttons auch leid? Schreiben wir uns doch eine Klasse für farbige Buttons.

Wie? Ganz einfach! Wir können uns eines der herausragenden Features von Java bedienen - der Möglichkeit auf der Basis einer vorhandenen Klasse (superclass) eine abgeleitete Klasse (subclass) zu schaffen. Die Subklasse erbt dabei alle Eigenschaften und Fähigkeiten der Superklasse. Darüber hinaus gewünschte Eigenschaften und Fähigkeiten werden in Form zusätzlicher Datenfelder und Methoden in der Subklasse definiert.

1. Nehmen Sie also die vorhandene Klasse `java.awt.Button` und erweitern Sie diese zu einer Klasse `ColorButton`, die über einen Konstruktor `public ColorButton(Color vordergrund, Color hintergrund)` verfügen soll.
Für die Setzung der Farben können Sie die Methoden `setForeground()` und `setBackground()` benutzen, über die alle GUI-Elemente des AWT-Package verfügen.
2. Testen Sie die Klasse `ColorButton` mit einer Applikation, die vier verschiedenfarbige Buttons in einem Fenster darstellt (siehe Abbildung oben).
3. Schreiben Sie eine einfache Klasse `Ball` und leiten Sie davon die Subklassen `Fussball` und `Rugbyball` ab. Testen Sie die Klassen in einer Applikation mit geeigneten Printausgaben.

Unterstützende Materialien:

- 8.2 Hinweise - Superklasse, Subklasse, Vererbung, Zugriffsmodifizierer, Überschreiben von Methoden, Polymorphismus, Abstrakte Klassen, Finale Klassen, finale Methoden
- 8.3 Lösungen

8.2 Hinweise

8.2.1 Superklasse, Subklasse, Vererbung

Java bietet die Möglichkeit vorhandene Klassen um neue Eigenschaften und Methoden zu erweitern. Dazu leiten sie mittels des Modifizierers `extends` aus der vorhandenen Klasse (Superklasse) eine neue Unterklasse (Subklasse) ab.

Beispiel:

a) Superklasse:

```
public class Fahrzeug {  
  
    protected int anzahlRaeder;  
  
    public Fahrzeug( int raeder ) {  
  
        anzahlRaeder = raeder;  
    }  
  
    public int getAnzahlRaeder() {  
        return( anzahlRaeder );  
    }  
  
    public void print() {  
        System.out.println("Dies_ist_ein_Fahrzeug_mit_" +  
            anzahlRaeder + "_Raedern." );  
    }  
}
```

b) Abgeleitete Klasse (Subklasse):

```
public class Automobil extends Fahrzeug {  
  
    protected int kWatt;  
  
    public Automobil( int raeder, int kw ) {  
        super( raeder );  
        kWatt = kw;  
    }  
  
    public int getKiloWatt() {  
        return( kWatt );  
    }  
}
```

Die Subklasse hat alle Eigenschaften und Methoden der Superklasse geerbt und stellt darüber hinaus noch die Eigenschaft `kWatt` und die Methode `getKiloWatt()` zur Verfügung.

Wenn Initialisierungen der Superklasse notwendig sind (im Beispiel muss die Anzahl der Räder als Parameter dem Konstruktor der Klasse `Fahrzeug` übergeben werden), so erfolgt dies im Konstruktor der Subklasse in der ersten Anweisung durch Referenzierung des Konstruktors der Superklasse (hier: `super(n)`). Der explizite Aufruf des Konstruktors der Superklasse kann entfallen, wenn keine Daten an die Superklasse weitergereicht werden müssen.

Die Superklasse kann aus der Subklasse heraus bei Bedarf immer mit der Referenz `super` angesprochen werden.

Java erlaubt nur eine Einfachvererbung (eine Subklasse kann nur aus einer Superklasse abgeleitet werden). Mehrfachvererbungen, wie sie in C++ oder Microsoft's Java-Plagiat C# möglich sind, sind in Java mit Absicht nicht implementiert.

8.2.2 Zugriffsmodifizierer

Modifizierer	Zugriff auf Methode bzw. Datenfeld möglich			
	in Klasse	in Subklasse	in Package	„weltweit“
private	x			
protected	x	x		
<kein>	x	x	x	
public	x	x	x	x

8.2.3 Überschreiben von Methoden, Polymorphismus

Sie können das Verhalten einer vorhandenen Klasse verändern, indem Sie Methoden der Superklasse in einer Subklasse überschreiben.

Beispiel:

Subklasse mit überschreiben einer Methode:

```
public class Automobil extends Fahrzeug {

    protected int kWatt;

    public Automobil( int raeder, int kw ) {
        super( raeder );
        kWatt = kw;
    }

    public int getKiloWatt() {
        return( kWatt );
    }

    public void print() {
        System.out.println("Dies ist ein Auto. Es hat " +
            anzahlRaeder + " Raeder und bringt eine Leistung von " +
            kWatt + " Kilowatt. ");
    }
}
```

In diesem Fall wurde neben der Erweiterung der Superklasse auch eine Überschreibung der Methode `print()` vorgenommen.

Anwendungsbeispiel:

```
public class AutoApp {

    public static void main( String[] argv ) {

        Fahrzeug fzRef = new Fahrzeug( 3 );
        Automobil autoRef = new Automobil( 4, 115 );

        fzRef.print();
    }
}
```

```
        autoRef.print();
    }
}
```

Wenn Sie dieses Beispiel testen, werden Sie feststellen, dass je nach Objekttyp (`Fahrzeug` oder `Automobil`) verschiedene Printausgaben erfolgen, obwohl der gleiche Methodenaufruf erfolgte (Polymorphismus).

8.2.4 Abstrakte Klassen

Werden in einer Klassendefinition eine oder mehrere Methoden zwar als Bestandteil der Schnittstelle (Interface) festgelegt, aber nicht mit Anwendungslogik gefüllt, so handelt es sich um eine abstrakte Klasse. Diese Klassen müssen mit dem Modifizierer `abstract` bezeichnet werden. Auch die entsprechenden „leeren“ Methoden werden durch `abstract` gekennzeichnet.

Beispiel:

```
public abstract class Fahrzeug {

    protected int anzahlRaeder;

    public Fahrzeug( int n ) {

        anzahlRaeder = n;
    }

    public int getAnzahlRaeder() {
        return( anzahlRaeder );
    }

    public abstract void print();
}
```

Aus abstrakten Klassen können keine Objekte deklariert werden, sie können aber als Superklasse fungieren. Wenn eine abstrakte Klasse erweitert wird, müssen die nicht vollständig definierten Methoden mit Anwendungslogik gefüllt werden.

8.2.5 Finale Klassen, finale Methoden

Wird eine Klasse mit dem Modifizierer `final` gekennzeichnet, kann sie nicht erweitert werden (Beispiel: `public final class java.lang.System`). Ebenso können Methoden mit dem Modifizierer `final` gekennzeichnet werden. In diesem Fall kann die Methode in Subklassen nicht überschrieben werden.

8.3 Lösungen

1. Die Klasse `ColorButton` als Subklasse von `java.awt.Button`.

```
1 import java.awt.*;
2
3 public class ColorButton extends Button {
4
5     public ColorButton( Color fg, Color bg ) {
6         setForeground( fg );
7         setBackground( bg );
8     }
9 }
10 }
```

2. Die Klasse `ColorButton` wird hier für die Deklaration von Objekten benutzt.

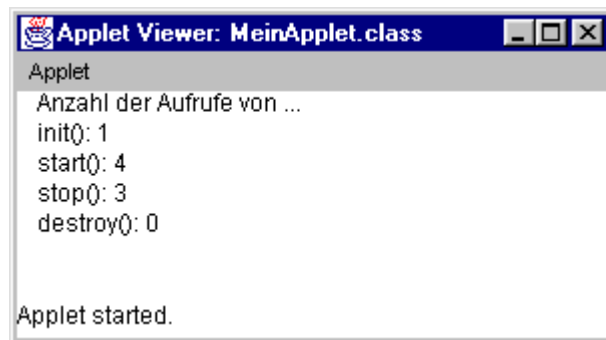
```
1 import java.awt.*;
2 import java.awt.event.*;
3
4 public class ColorButtonApplikation{
5
6     ColorButton cbRef1;
7     ColorButton cbRef2;
8     ColorButton cbRef3;
9     ColorButton cbRef4;
10    Frame frameRef;
11
12    /**
13     * Konstruktor
14     */
15    public ColorButtonApplikation() {
16        init();
17    }
18
19    /**
20     * Initialisiert die Applikation
21     */
22    private void init() {
23        // Erzeuge Programmfenster
24        frameRef = new Frame();
25        frameRef.setTitle("Farbige_Buttons");
26        GridLayout glRef = new GridLayout(2,2);
27        frameRef.setLayout(glRef);
28        // Erzeuge farbigen Button mit Text
29        cbRef1 = new ColorButton( Color.white, Color.black );
30        cbRef1.setLabel("Exit");
31
32        // Registriere ActionListener fuer ColorButton
33        ColorButtonListener cbL = new ColorButtonListener();
34        cbRef1.addActionListener( cbL );
35
36        // Erzeuge farbigen Button mit Text
37        cbRef2 = new ColorButton( Color.black, Color.white );
38        cbRef2.setLabel("Schwarz_auf_weiss");
39        // Erzeuge farbigen Button mit Text
40        cbRef3 = new ColorButton( Color.yellow, Color.red );
41        cbRef3.setLabel("Gelb_auf_rot");
42        // Erzeuge farbigen Button mit Text
43        cbRef4 = new ColorButton( Color.blue, Color.yellow );
44        cbRef4.setLabel("Blau_auf_Gelb");
```

```
45
46     frameRef.add( cbRef1 );
47     frameRef.add( cbRef2 );
48     frameRef.add( cbRef3 );
49     frameRef.add( cbRef4 );
50 }
51
52 /**
53  * Startet Applikation
54  */
55 public void run() {
56     frameRef.setSize(300,300);
57     frameRef.setVisible( true );
58 }
59
60 /**
61  * Innere Klasse als ActionListener
62  */
63 class ColorButtonListener implements ActionListener {
64     public void actionPerformed( ActionEvent e ) {
65         System.exit(0);
66     }
67 }
68
69 public static void main( String[] argv ) {
70     ColorButtonApplikation cbApp = new ColorButtonApplikation();
71     cbApp.run();
72 }
73
74 }
```

Übung 9

Der Lebenszyklus eines Applets

9.1 Aufgaben



Applets sind ein populäres Anwendungsgebiet für die Programmiersprache Java. Anders als Applikationen sind Applets nicht selbständig lauffähig, sondern benötigen ein Wirtprogramm, welches den Ablauf des Applets steuert. Populär ist die Verwendung von Applets als Ergänzung von HTML-Dokumenten im Internet und Intranet. In diesem Fall steuert der Webbrowser, der das HTML-Dokument anzeigt auch den Ablauf des Applets.

In dieser Übung stellen wir Ihnen am Beispiel (siehe Aufgaben 1 bis 5) den grundsätzlichen Aufbau eines Java-Applets und dessen Funktionsweise vor.

1. Übernehmen Sie den nachfolgend abgedruckten Quelltext für ein elementares Applet und kompilieren Sie die Klasse `MeinTestApplet`.

```
1 import java.awt.*;
2 import java.applet.*;
3
4 public class MeinTestApplet extends Applet {
5
6     int anzahlInit;
7     int anzahlStart;
8     int anzahlStop;
9     int anzahlDestroy;
10
11     public void init() {
12         anzahlInit++;
13     }
14
15     public void start() {
16         anzahlStart++;
17     }
18
19     public void stop() {
20         anzahlStop++;
21     }
22
23     public void destroy() {
```



```

24     anzahlDestroy++;
25     Toolkit.getDefaultToolkit().beep();
26 }
27
28 public void paint( Graphics g ) {
29     g.drawString( "Anzahl_der_Aufrufe_von_...", 10, 10 );
30     g.drawString( "init():_" + anzahlInit, 10, 20 );
31     g.drawString( "start():_" + anzahlStart, 10, 30 );
32     g.drawString( "stop():_" + anzahlStop, 10, 40 );
33     g.drawString( "destroy():_" + anzahlDestroy, 10, 50 );
34 }
35
36 }

```

2. Schreiben Sie ein HTML-Dokument zum Applet `MeinTestApplet` nach dem folgenden Muster:

```

1 <HTML>
2 <!-- HTML-Testdatei fuer Java-Applet MeinTestApplet-->
3 <BODY>
4 <APPLET CODE="MeinTestApplet.class" WIDTH="400" HEIGHT="200">
5 </APPLET>
6 </BODY>
7 </HTML>

```

Hinweis: Wenn Sie Nekje als Editor verwenden, können Sie durch Drücken des HTML-Buttons bei aktiviertem Java-Quelltextfenster eine HTML-Datei zum Applet generieren lassen.

3. Starten Sie das Applet, indem Sie das HTML-Dokument in den Appletviewer von Sun laden.

Hinweis: Wenn Sie Nekje verwenden, drücken Sie einfach den Ausführungsbutton bei aktiviertem HTML-Textfenster. Es sollte sich der Appletviewer mit dem Applet öffnen.

4. Verkleinern Sie das Ausführungsfenster des Appletviewers zum Symbol, vergrößern Sie das Fenster anschließend wieder, wiederholen Sie Verkleinerung und Vergrößerung mehrfach. Beenden Sie die Ausführung des Applets.

Welche Aussagen können Sie zum Aufruf der Methoden des Applets treffen? Vergleichen Sie die Aussagen mit den Hinweisen zu dieser Übung.

5. Laden Sie das HTML-Dokument in einen Webbrowser (Netscape Navigator oder MS Internet Explorer). Können Sie Unterschiede im Verhalten des Applets zwischen der Ausführung im Appletviewer und im Webbrowser feststellen?

Unterstützende Materialien:

- 9.2 Hinweise - Erläuterungen zur Klasse `java.applet.Applet`

9.2 Hinweise

9.2.1 Erläuterungen zur Klasse `java.applet.Applet`

Applets sind unselbständige Programme, die nur mit Hilfe eines Wirtes (z. B. Appletviewer oder Webbrowsers) zum Laufen gebracht werden können. Der Wirt erzeugt aus der Applet-Klasse eine Objektinstanz und ruft nach vorgegebenen Muster bestimmte Methoden des Appletobjekts auf.

Um ihr eigenes Applet zu schreiben, müssen Sie eine Subklasse aus der Superklasse `java.applet.Applet` ableiten und geeignete Methoden der Superklasse mit ihrer eigenen Definition überschreiben.

Das Grundmuster für die Definition eines Applets sieht wie folgt aus:

```
import java.awt.*;
import java.applet.*;

public class MeinApplet extends Applet {
    ...
}
```

Ein Blick in die Dokumentation zeigt, dass die Klasse `java.applet.Applet` ein Reihe von vordefinierten Methoden als Schnittstelle anbietet.

Elementare Methoden der Applet-Schnittstelle sind die Methoden `init()`, `start()`, `stop()` und `destroy()`, `paint()`. Diese Methoden werden vom Wirt zu festgelegten Zeitpunkten aufgerufen und eignen sich zum Überschreiben mit eigenen Methodendefinitionen. In der Superklasse `java.applet.Applet` sind diese Methoden zwar definiert, führen aber keinerlei Aktionen aus.

```
public void init()
```

Diese Methode wird vom Wirt einmal aufgerufen, um das Applet davon in Kenntnis zu setzen, dass es in das System geladen wurde.

Wenn Sie diese Methode überschreiben, sollten Sie hier grundlegende Initialisierungen vornehmen (Setzen von Variablenwerten, Deklaration von eigenen Objekten).

```
public void start()
```

Diese Methode wird vom Wirt aufgerufen, um das Applet davon in Kenntnis zu setzen, dass es mit der Ausführung beginnen soll. Ein erster Aufruf erfolgt automatisch nach der Methode `init()`. Zu späteren Zeitpunkten sollte die Methode vom Wirt aufgerufen werden, wenn das Applet nach einer Phase der Unsichtbarkeit wieder sichtbar wird.

Wenn Sie bestimmte Ausgangsinitialisierungen für das Applet vornehmen möchten, die vor jeder Appletausführung ausgeführt werden, können Sie diese durch Überschreiben der Methode `start()` definieren.

```
public void stop()
```

Diese Methode wird vom Wirt aufgerufen, um das Applet davon in Kenntnis zu setzen, dass es die Ausführung beenden muss. Ein Aufruf durch den Wirt erfolgt kurz vor der Zerstörung des Objekts und sollte auch immer dann erfolgen, wenn das Applet vorübergehend unsichtbar wird. Falls Sie nach jeder Applet-Ausführung aufräumen müssen - hier ist der geeignete Platz.

```
public void destroy()
```

Diese Methode wird vom Wirt aufgerufen, um das Applet davon in Kenntnis zu setzen, dass es nicht mehr benötigt wird und alle besetzten Ressourcen freigeben soll. Ein Aufruf kann jederzeit erfolgen, wenn die HTML-Seite verlassen wurde, in die das Applet eingebettet ist und der belegte Speicherbereich anderweitig benötigt wird.

Die Methode `destroy()` ist die letzte Gelegenheit, Aktionen vom Applet durchführen zu lassen. Typischerweise sollten Sie hier geöffnete Dateien schließen und Netzwerkverbindungen unterbrechen.

```
public void paint( Graphics g )
```

Diese Methode können Sie benutzen, um auf der Oberfläche eines Applets zu zeichnen. Die Methode `paint()` wird automatisch aufgerufen, sobald ein Neuzeichnen des Applets auf dem Bildschirm notwendig wird. Wie Sie der Dokumentation entnehmen können, wird diese Methode in der Klasse `java.awt.Container` definiert, von der die Klasse `java.applet.Applet` eine Subsubklasse ist. Ohne dass Sie sich darum kümmern müssen, wird durch das Applet der Methode `paint()` eine Referenz auf ein Objekt vom Typ `java.awt.Graphics` übergeben. Dieses Objekt kapselt Details zur Systemumgebung, in der das Applet zur Ausführung kommt und zum verwendeten Grafikkontext. Sie können Methoden des `Graphics`-Objekts für Text- und Grafikausgaben auf der Appletoberfläche benutzen.

Beispiel:

```
import java.awt.*;
import java.applet.Applet;

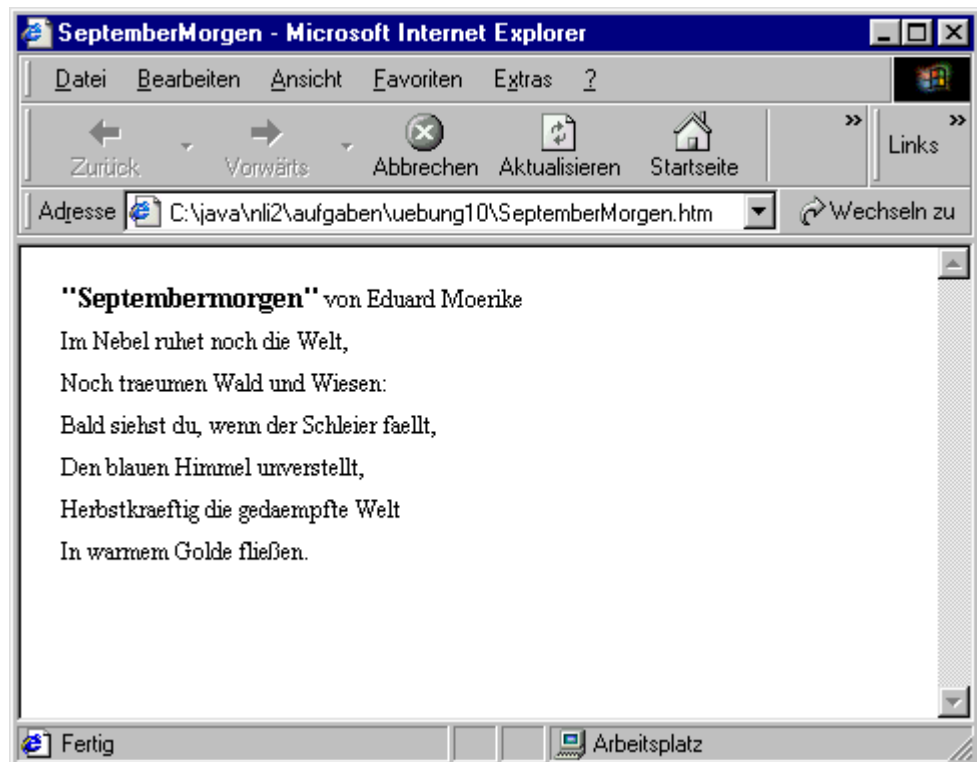
public class GanzKleines Applet {
    public void paint( Graphics g ) {

        g.drawString( "Alles_ganz_einfach", 10, 100 );
    }
}
```

Übung 10

Septembermorgen

10.1 Aufgaben



1. Schreiben Sie ein Applet, welches das Gedicht „Septembermorgen“ (siehe obige Abbildung) unter Berücksichtigung aller Text-Formatierungen (Schrifttyp Times Roman, Standard-Schriftgröße 12, Überschrift Schriftgröße 14 und fett) im Browser ausgibt.
2. Experimentieren Sie mit verschiedenen Textfarben, indem Sie die Methode `setColor()` der Klasse `java.awt.Graphics` aufrufen.
3. Wie können Sie den Hintergrund des Applets farblich verändern?
4. Gelingt es Ihnen, das Gedicht in einem Rahmen zu veröffentlichen?

Unterstützende Materialien:

- 10.2 Hinweise - Textformatierungen im Grafik-Kontext
- 10.3 Lösung

10.2 Hinweise

10.2.1 Textformatierungen im Grafik-Kontext

Sie finden Hinweise zur Textformatierung in der Dokumentation der Klassen `java.awt.Graphics` (Methoden `setFont()` und `setColor()`) und `java.awt.Font`, `java.awt.Color`.

Ein Beispiel, wie Sie in einem Applet Textformatierungen vornehmen können:

```
public void paint ( Graphics g ) {  
  
    Font f = new Font( "SansSerif", Font.BOLD, 20 );  
  
    g.setFont( f );  
  
    g.drawString("Nehmt_Abschied_Brueder,", 10, 10 );  
  
    f = new Font( "Serif", Font.PLAIN, 14 );  
  
    g.setFont( f );  
  
    g.drawString("ungewiss_ist_alle_Wiederkehr,", 10, 25 );  
  
}
```

Auf jeder Systemplattform verfügbare Standardzeichensätze:

Java-Bezeichnung	Windows-Äquivalent
Serif	Times Roman
SansSerif	Arial
Monospaced	Courier
Dialog	<systemabhängig>
DialogInput	<systemabhängig>

10.3 Lösung

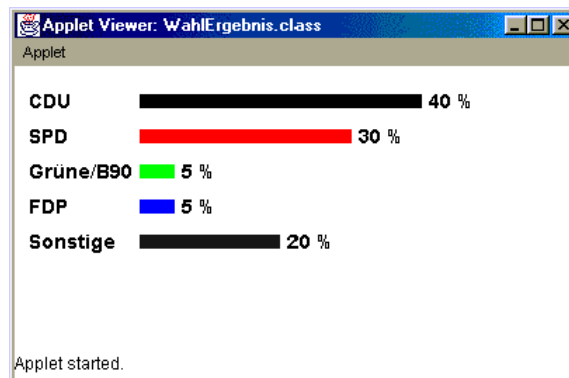
1. Mögliche Lösung:

```
1 import java.awt.*;
2 import java.applet.Applet;
3
4 public class SeptemberMorgen extends Applet {
5
6     private String titell = "\"Septembermorgen\"";
7     private String titel2 = "_von_Eduard_Moerike";
8
9     private String[] gedicht = {
10         "Im_Nebel_ruhet_noch_die_Welt,",
11         "Noch_traeumen_Wald_und_Wiesen:",
12         "Bald_siehst_du,_wenn_der_Schleier_faellt,",
13         "Den_blauen_Himmel_unverstellt,",
14         "Herbstkraeftig_die_gedaempfte_Welt",
15         "In_warmem_Golde_flieSSen."
16     };
17
18     public void paint( Graphics g ) {
19
20         // Variablen fuer genaue Positionierung
21         int xpos = 0;
22         int ypos = 0;
23
24         // Fontinfos fuer Ueberschrift Teil 1
25         Font fontRef = new Font( "Serif", Font.BOLD, 14 );
26         FontMetrics fmRef = getFontMetrics( fontRef );
27         g.setFont( fontRef );
28
29         // Ausgabe der Ueberschrift Teil 1
30         xpos = 10;
31         ypos = 15;
32         g.drawString(titell, xpos, ypos);
33
34         // Position fuer Ueberschrift Teil 2
35         xpos += fmRef.stringWidth(titell);
36
37         // Fontinfos fuer den Rest des Gedichts
38         fontRef = new Font("Serif", Font.PLAIN, 12);
39         fmRef = getFontMetrics( fontRef );
40
41         // Ausgabe Ueberschrift Teil 2
42         g.setFont( fontRef );
43         g.drawString(titel2, xpos, ypos);
44
45         // Ausgabe Gedicht
46         xpos = 10;
47         int zsprung = fmRef.getHeight() + 5;
48         ypos += zsprung;
49         for( int i = 0; i < gedicht.length; i++ ) {
50             g.drawString( gedicht[i], xpos, ypos );
51             ypos += zsprung;
52         }
53     }
54 }
```

Übung 11

Wahlergebnisse

11.1 Aufgaben



Wahlabend in einem kleinen Bundesland im Südwesten der Republik. Die Wahl ist gelaufen. Das vorläufige amtliche Endergebnis lautet:

CDU	45.5 %
SPD	44.4 %
Grüne/Bündnis 90	3.2 %
FDP	2.6 %
Sonstige	4.3 %

1. Für die Webseite des Landeswahlleiters soll ein Applet geschrieben werden, welches die Wahlergebnisse in ansprechender Form als Balkendiagramm mehrfarbig präsentiert (siehe Abbildung oben).
2. Neidvoll blickt der Landeswahlleiter Berlin auf das Applet Wahlergebnisse. Er möchte es für die bevorstehenden Wahlen auch haben.
Formulieren Sie diesmal den Quelltext so, dass das Applet in der Lage ist, für jedes mögliche Wahlergebnis gleich welchen Bundeslandes die Daten einzulesen und als Balkendiagramm auszugeben.
3. „Das Wappen unseres Landes muss unbedingt mit ins Applet!“ Der Kunde ist König, machen Sie mal!

Ergänzende Aufgabenstellungen:

Stellen Sie Wahlergebnisse als Linienchart über mehrere Jahren und als Kreisdiagramm dar.

Unterstützende Materialien:

- 11.2 Hinweise - Arrays
- 11.3 Hilfen
- 11.4 Lösungen

11.2 Hinweise

11.2.1 Arrays

Mehrere Datenfelder vom gleichen Typ können Sie in einem Verbund von Datenfeldern, einem sogenannten Array, zusammenfassen. Der Zugriff auf die einzelnen Werte des Arrays ist über Indizes von 0 bis n-1 möglich.

Beispiel für die Verwendung von Arrays:

```
String[] parteien = new String[3]; // Deklaration eines Arrays
// fuer Referenzen auf
// drei Stringobjekte

parteien[0] = "FDP";
parteien[1] = "SPD";
parteien[2] = "CDU";

double[] prozente = new double[3]; // Deklaration eines Arrays
// fuer die Aufnahme von
// drei Werten vom Typ double

prozente[0] = 6.8;
prozente[1] = 52.0;
prozente[2] = 38.7;

System.out.println( "Ergebnis_" + parteien[1] + ":_ " + prozente[1] );
```

Die Deklaration und die Wertzuweisung können bei Arrays auch in einer Anweisung erfolgen.

```
int[] Gehaelter = { 6800, 5200, 4800 };

String[] wochenTage = { "Sonntag", "Montag", "Dienstag", "Mittwoch",
    "Donnerstag", "Freitag", "Samstag" };
```

Die Länge eines Arrays kann über die für jeden Array definierte Konstante `length` ermittelt werden.

```
int arrayLaenge = wochenTage.length;
```

11.3 Hilfen

zu 1. Spezielles Wahlapplet

- Balkendiagramm

Im Package `java.awt` finden Sie in der Klasse `Graphics` eine Fülle von Methoden zum Zeichnen von Diagrammen.

Für ein Balkendiagramm bieten sich die Methoden `java.awt.Graphics.drawRect()`, `java.awt.Graphics.fillRect()` an.

- Exakte Positionierung

Der Punkt 0,0 ist für alle Grafikmethoden der Klasse `Graphics` in der oberen linken Ecke der Ausgabefläche definiert.

Die Abmessungen von Textausgaben können mit Methoden der Klasse `java.awt.FontMetrics` ermittelt werden. Ein Objekt der Klasse `FontMetrics` erzeugen Sie durch Aufruf der Applet-Methode `getFontMetrics()`.

- Verschiedene Farben

Mit der Methode `setColor()` der Klasse `Graphics` können Sie die Vordergrundfarbe für die Ausgabe von Grafikoperationen festlegen.

In der Klasse `java.awt.Color` sind viele Standardfarben als Konstanten festgelegt (z. B. `Color.red` oder `Color.blue`). Durch Aufruf des Konstruktors `Color(int r, int g, int b)` können Sie auch leicht eine individuelle Farbmischung nach dem RGB-Modell schaffen.

- Zahlwerte in String konvertieren

Die Wandelung von Zahlwerten in Strings (z. B. bei der Ausgabe der Wahlprozente im Diagramm benötigt) ist durch die statische Methode `<Datentyp>.toString()` möglich (Beispiel: `String text = Double.toString(30.5);`).

zu 2. Universelles Wahlapplet

Vorschlag:

Definieren Sie für die drei Gruppen `ParteiName`, `ParteiProzente` und `ParteiFarben` Arrays, in die Sie mit einer `for`-Schleife jeweils die Parameterwerte einlesen. Vorab müssen Sie noch die Anzahl der Werte ermitteln, entweder durch Übergabe eines entsprechenden Wertes an das Applet oder durch Scannen aller Parameterwerte eines Typs (z.B. `ParteiName`).

- Einlesen von Applet-Parametern

Parameterwerte können für ein Applet mit dem HTML-Tag `<PARAM>` innerhalb einer Applet-Definition übergeben werden.

Beispiel:

```
1 <APPLET CODE="WahlErgebnis2.class" WIDTH="400" HEIGHT="200">
2 <PARAM NAME="Partei0" VALUE="CDU">
3 <PARAM NAME="Prozent0" VALUE="30.0">
4 <PARAM NAME="Farbe0" VALUE="Schwarz">
5 </APPLET>
```

Im Java-Applet sorgt die Methode `getParameter()` für das Einlesen eines Parameterwertes als Stringobjekt.

Beispiel:

```
String wertRef = getParameter("Prozent0");
```

Achtung: Evtl. sind im Programm Konvertierungen in andere Datentypen notwendig.

- String-Objekte in Zahlenwerte konvertieren

Strings können Sie mit den statischen Methoden `Integer.parseInt()`, `Double.parseDouble()`, usw. in Werte vom Datentyp `int`, `double` usw. konvertieren.

Vorsicht Falle: Außer `Integer.parseInt()` sind die weiteren Parse-Methoden erst seit dem JDK 1.2 eingeführt. Die Browser bis Herbst 1999 verstehen aber nur den Java-Sprachumfang bis JDK 1.1, so dass Sie sich bei Applets, die mit einem Alt-Browser angezeigt werden sollen, behelfen müssen. Eine selbstdefinierte Konvertierungsmethode, die alle javafähigen Browser verarbeiten können, kann folgendermaßen aussehen:

```
private double String2Double( String str ) {
    double retVal = 0.0;

    try {
        Double temp = Double.valueOf(str);
        retVal = temp.doubleValue();
    }
    catch( NumberFormatException e ) {
        // Bei Fehler bleibt der Returnwert 0.0
    }

    return(retVal);
}
```

Die Methode `String2Double()` verwendet nur Methoden, die bereits seit JDK 1.0 verfügbar sind. Allerdings verlangt die Methode `Double.valueOf()`, dass eine mögliche `NumberFormatException` abgefangen wird (mehr zu Exceptions in Kapitel 13).

zu 3. Bilder einbinden

Speichern Sie die Bilder, die Sie im Applet verwenden wollen, am Besten im Ursprungsverzeichnis des Applets. Benutzen Sie für das Laden des Bildes die Applet-Methode `getImage()`.

Als URL geben Sie das Ursprungsverzeichnis des Applets an, welches Sie mit der Methode `getCodeBase()` ermitteln können. Als String brauchen Sie dann nur noch den Dateinamen des Bildes angeben.

Beispiel: `Image example = getImage(getCodeBase(), "beispiel.gif");`

In vielen Anwendungsfällen ist es ratsam mit einem `MediaTracker`-Objekt dafür Sorge zu tragen, dass Bilder in dem Moment, wo sie ausgegeben werden sollen, auch bereits vollständig geladen sind. Ein Beispiel für die Umsetzung finden Sie in der Dokumentation zur Klasse `java.lang.MediaTracker`.

11.4 Lösungen

1. So kann die Lösung aussehen:

```
1 import java.awt.*;
2 import java.applet.Applet;
3
4 public class WahlErgebnis extends Applet {
5
6     private String[] parteien = {
7         "CDU",
8         "SPD",
9         "Gruene/B90",
10        "FDP",
11        "Sonstige"
12    };
13
14    private Color[] parteiFarben = {
15        Color.black,
16        Color.red,
17        Color.green,
18        Color.blue,
19        Color.lightGray
20    };
21
22    private double[] prozente = {
23        40.0, 30.0, 5.0, 5.0, 20.0
24    };
25
26    private Font f;
27    private int height;
28    private int maxParteiLength;
29
30    public void init() {
31
32        // Einmalige Initialisierungen
33        f = new Font( "SansSerif", Font.BOLD, 14 );
34        setFont( f );
35        FontMetrics fm = getFontMetrics( f );
36        height = fm.getHeight();
37
38        for( int i = 0; i < parteien.length; i++ ) {
39            int len = fm.stringWidth( parteien[i] );
40            if( maxParteiLength < len )
41                maxParteiLength = len;
42        }
43
44    }
45
46    public void paint( Graphics g ) {
47        // Position Text
48        int xpos1 = 10;
49        // Position Balken
50        int xpos2 = xpos1 + maxParteiLength + 5;
51        // Vertikale Startposition
52        int ypos = 10 + height;
53        int scale = 5;
54
55        for( int i = 0; i < parteien.length; i++ ) {
56            // Textausgabe
57            g.setColor( Color.black );
```

```
58     g.drawString( parteien[i], xpos1, ypos );
59
60     // Balkenausgabe
61     int balkenLen = (int)( prozente[i]*scale );
62     g.setColor( parteiFarben[i] );
63     g.fillRect( xpos2, ypos-height/2, balkenLen, 10 );
64
65     // Prozentausgabe
66     g.setColor( Color.black );
67     g.drawString( Double.toString(prozente[i]) + "%",
68                 xpos2+balkenLen+5, ypos );
69
70     // Zeilensprung
71     ypos += height+5;
72 }
73 }
74 }
```

2. Ein universelles Wahlapplet kann das folgende Aussehen haben:

```

1 <HTML>
2 <!-- HTML-Testdatei fuer Java-Applet WahlErgebnis2 -->
3 <BODY>
4 <APPLET CODE="WahlErgebnis2.class" WIDTH="400" HEIGHT="200">
5 <PARAM NAME="Partei0" VALUE="CDU">
6 <PARAM NAME="Prozent0" VALUE="50">
7 <PARAM NAME="Farbe0" VALUE="Schwarz">
8 <PARAM NAME="Partei1" VALUE="SPD">
9 <PARAM NAME="Prozent1" VALUE="30">
10 <PARAM NAME="Farbe1" VALUE="Rot">
11 <PARAM NAME="Partei2" VALUE="B90/Grüne">
12 <PARAM NAME="Prozent2" VALUE="10">
13 <PARAM NAME="Farbe2" VALUE="Gruen">
14 </APPLET>
15 </BODY></HTML>

1 import java.awt.*;
2 import java.applet.Applet;
3
4 public class WahlErgebnis2 extends Applet {
5
6     private String[] parteien;
7     private Color[] parteiFarben;
8     private String[] prozente;
9
10    private Font f;
11    private int height;
12    private int maxParteiLength;
13
14    private int anzahlParteien;
15
16    public void init() {
17        // Parameter einlesen
18        anzahlParteien = getAnzahlParteien(); // Ermittelt Anzahl der Parteien
19        getParametersParteien();           // Liest Parteibezeichnungen ein
20        getParametersProzente();           // Liest Prozente ein
21        getParametersFarben();             // Liest Parteifarben ein
22
23        // Einmalige Initialisierungen
24        f = new Font("SansSerif", Font.BOLD, 14);
25        setFont(f);
26        FontMetrics fm = getFontMetrics(f);
27        height = fm.getHeight();
28
29        for( int i = 0; i < parteien.length; i++ ) {
30            int len = fm.stringWidth(parteien[i]);
31            if( maxParteiLength < len )
32                maxParteiLength = len;
33        }
34
35    }
36
37    private int getAnzahlParteien() {
38        int anzahl = 0;
39
40        // Mehr als 100 sollen es ja wohl nicht sein ...
41        for( ; anzahl < 100; anzahl++ ) {
42            String pWert = "Partei" + anzahl;
43            String partei = getParameter( pWert );

```

```
44
45     if( partei == null ) // Schluss!!!
46         break;
47     }
48
49     return( anzahl );
50 }
51
52 private void getParametersParteien() {
53     // Array deklarieren
54     parteien = new String[ anzahlParteien ];
55
56     for(int i=0; i < parteien.length; i++) {
57         String pWert = "Partei" + i;
58         parteien[i] = getParameter( pWert );
59         // Wenn Parameter nicht vorhanden, dann ist Schluss
60         if( parteien[i] == null )
61             break;
62     }
63 }
64
65 private void getParametersProzente() {
66     // Array deklarieren
67     prozente = new String[ anzahlParteien ];
68
69     for(int i=0; i < prozente.length; i++) {
70         String pWert = "Prozent" + i;
71         prozente[i] = getParameter( pWert );
72         // Wenn Parameter nicht vorhanden, dann ist Schluss
73         if( prozente[i] == null )
74             break;
75     }
76 }
77
78 private void getParametersFarben() {
79     // Array deklarieren
80     parteiFarben = new Color[ anzahlParteien ];
81
82     for(int i=0; i < parteiFarben.length; i++) {
83         String pWert = "Farbe" + i;
84
85         String farbe = getParameter( pWert );
86         // Wenn Parameter nicht vorhanden, dann ist Schluss
87         if( farbe == null )
88             break;
89         else {
90             if( farbe.equals("Rot") )
91                 parteiFarben[i] = Color.red;
92             else if( farbe.equals("Schwarz") )
93                 parteiFarben[i] = Color.black;
94             else if( farbe.equals("Gruen") )
95                 parteiFarben[i] = Color.green;
96             else if( farbe.equals("Blau") )
97                 parteiFarben[i] = Color.blue;
98             else
99                 parteiFarben[i] = Color.lightGray;
100        }
101    }
102 }
103
```

```
104 private double String2Double( String str ) {
105     double retVal = 0.0;
106
107     try {
108         Double temp = Double.valueOf(str);
109         retVal = temp.doubleValue();
110     }
111     catch( NumberFormatException e ) {
112     }
113
114     return(retVal);
115 }
116
117 public void paint( Graphics g ) {
118     // Position Text
119     int xpos1 = 10;
120     // Position Balken
121     int xpos2 = xpos1 + maxParteiLength + 5;
122     // Vertikale Startposition
123     int ypos = 10 + height;
124     int scale = 5;
125
126     for( int i = 0; i < parteien.length; i++ ) {
127         // Textausgabe
128         g.setColor(Color.black);
129         g.drawString( parteien[i], xpos1, ypos );
130
131         // Balkenausgabe
132
133         double wert = String2Double( prozente[i] );
134         int balkenLen = (int)(wert*scale);
135         g.setColor(parteiFarben[i]);
136         g.fillRect( xpos2, ypos-height/2, balkenLen, 10 );
137
138         // Prozentausgabe
139         g.setColor(Color.black);
140         g.drawString( prozente[i] + "_%",
141             xpos2+balkenLen+5, ypos );
142
143         // Zeilensprung
144         ypos += height+5;
145     }
146 }
147 }
```

Ein alternativer Lösungsansatz mit einer Klasse `Partei`, welche die Daten für jede Partei kapselt, und einem Objekt vom Typ `java.util.Vector`, welches die verschiedenen Parteiobjekte verwaltet.

```

1 <HTML>
2 <!-- HTML-Testdatei fuer Java-Applet WahlErgebnis2a -->
3 <BODY>
4 <APPLET CODE="WahlErgebnis2a.class" WIDTH="400" HEIGHT="200">
5 <PARAM NAME="Param0" VALUE="CDU,30.0,0,0,0">
6 <PARAM NAME="Param1" VALUE="SPD,35.7,255,0,0">
7 <PARAM NAME="Param2" VALUE="Grüne/B90,15.9,0,255,0">
8 </APPLET>
9 </BODY></HTML>

1 import java.awt.*;
2 import java.applet.Applet;
3 import java.util.*;
4
5 public class WahlErgebnis2a extends Applet {
6
7     Vector parteien;
8
9     private Font f;
10    private int height;
11    private int maxParteiLength;
12
13    public void init() {
14        // Parameter einlesen
15        getParameters();
16
17        // Einmalige Initialisierungen
18        f = new Font("SansSerif", Font.BOLD, 14);
19        setFont(f);
20        FontMetrics fm = getFontMetrics(f);
21        height = fm.getHeight();
22
23        for( Enumeration e = parteien.elements(); e.hasMoreElements(); ) {
24
25            Partei p = (Partei)e.nextElement();
26            int len = fm.stringWidth( p.getName() );
27            if( maxParteiLength < len )
28                maxParteiLength = len;
29        }
30    }
31 }
32
33 public void getParameters() {
34
35     parteien = new Vector();
36
37     int i = 0;
38     while( true ) {
39
40         String pWert = "Param"+i;
41         String paramString = getParameter( pWert );
42
43         if( paramString == null )
44             break;
45         else {
46             Partei temp = new Partei( paramString );
47             parteien.addElement( temp );

```



```
48     }
49
50     i++;
51 }
52 }
53
54 private double String2Double( String str ) {
55     double retVal = 0.0;
56
57     try {
58         Double temp = Double.valueOf(str);
59         retVal = temp.doubleValue();
60     }
61     catch( NumberFormatException e ) {
62     }
63
64     return(retVal);
65 }
66
67 public void paint( Graphics g ) {
68     // Position Text
69     int xpos1 = 10;
70     // Position Balken
71     int xpos2 = xpos1 + maxParteiLength + 5;
72     // Vertikale Startposition
73     int ypos = 10 + height;
74     int scale = 5;
75
76     for( Enumeration e = parteien.elements(); e.hasMoreElements(); ) {
77
78         Partei p = (Partei)e.nextElement();
79
80         // Textausgabe
81         g.setColor(Color.black);
82         g.drawString( p.getName(), xpos1, ypos );
83
84         // Balkenausgabe
85         double wert = String2Double( p.getProzente() );
86         int balkenLen = (int)(wert*scale);
87         g.setColor( p.getFarbe() );
88         g.fillRect( xpos2, ypos-height/2, balkenLen, 10 );
89
90         // Prozentausgabe
91         g.setColor(Color.black);
92         g.drawString( p.getProzente() + "%", xpos2+balkenLen+5, ypos );
93
94         // Zeilensprung
95         ypos += height+5;
96     }
97 }
98 }

```

```
1 import java.util.*;
2 import java.awt.*;
3
4 public class Partei {
5
6     private String parteiName;
7     private String parteiProzente;
8     private Color parteiFarbe;
9

```

```
10 public Partei( String name, String prozente, int r, int g, int b ) {
11     parteiName = name;
12     parteiProzente = prozente;
13     parteiFarbe = new Color(r,g,b);
14 }
15
16 public Partei( String paramStr ) {
17     tokenize( paramStr );
18 }
19
20 public void setName( String name ) {
21     parteiName = name;
22 }
23
24 public void setProzente( String prozente ) {
25     parteiProzente = prozente;
26 }
27
28 public void setFarbe( int r, int g, int b ) {
29     parteiFarbe = new Color( r, g, b );
30 }
31
32 public String getName() {
33     return( parteiName );
34 }
35
36 public String getProzente() {
37     return( parteiProzente );
38 }
39
40 public Color getFarbe() {
41     return( parteiFarbe );
42 }
43
44 private void tokenize( String paramStr ) {
45     int red = 0;
46     int green = 0;
47     int blue = 0;
48
49     StringTokenizer st = new StringTokenizer( paramStr, "," );
50
51     int num = st.countTokens();
52
53     for(int i = 0; i < num; i++) {
54         if( st.hasMoreTokens() ) {
55             String token = st.nextToken();
56
57             switch(i) {
58                 case 0:
59                     parteiName = token;
60                     break;
61                 case 1:
62                     parteiProzente = token;
63                     break;
64                 case 2:
65                     red = Integer.parseInt(token);
66                     break;
67                 case 3:
68                     green = Integer.parseInt(token);
69                     break;
```

```
70         case 4:
71             blue = Integer.parseInt(token);
72             break;
73         default:
74             break;
75     }
76 }
77 }
78
79     setFarbe( red, green, blue );
80 }
81
82 }
```

3. Nachfolgend das Wahlapplet mit einer zusätzlichen Bilddatei:

```

1 <HTML>
2 <!-- HTML-Testdatei fuer Java-Applet WahlErgebnis2-->
3 <BODY>
4 <APPLET CODE="WahlErgebnis2.class" WIDTH="400" HEIGHT="200">
5 <PARAM NAME="Partei0" VALUE="CDU">
6 <PARAM NAME="Prozent0" VALUE="50">
7 <PARAM NAME="Farbe0" VALUE="Schwarz">
8 <PARAM NAME="Partei1" VALUE="SPD">
9 <PARAM NAME="Prozent1" VALUE="30">
10 <PARAM NAME="Farbe1" VALUE="Rot">
11 <PARAM NAME="Partei2" VALUE="B90/Grüne">
12 <PARAM NAME="Prozent2" VALUE="10">
13 <PARAM NAME="Farbe2" VALUE="Gruen">
14 </APPLET>
15 </BODY></HTML>

1 import java.awt.*;
2 import java.applet.Applet;
3
4 public class WahlErgebnis3 extends Applet {
5
6     private String[] parteien;
7     private Color[] parteiFarben;
8     private String[] prozente;
9     private Image img;
10
11     private Font f;
12     private int height;
13     private int maxParteiLength;
14
15     private int anzahlParteien;
16
17     public void init() {
18         // Parameter einlesen
19         anzahlParteien = getAnzahlParteien(); // Ermittelt Anzahl der Parteien
20         getParametersParteien();           // Liest Parteibezeichnungen ein
21         getParametersProzente();           // Liest Prozente ein
22         getParametersFarben();             // Liest Parteifarben ein
23
24         String bildName;
25
26         if( (bildName = getParameter("Bild")) != null ) {
27             img = getImage( getCodeBase(), bildName );
28         }
29
30         // Einmalige Initialisierungen
31         f = new Font("SansSerif", Font.BOLD, 14);
32         setFont(f);
33         FontMetrics fm = getFontMetrics(f);
34         height = fm.getHeight();
35
36         for( int i = 0; i < parteien.length; i++ ) {
37             int len = fm.stringWidth(parteien[i]);
38             if( maxParteiLength < len )
39                 maxParteiLength = len;
40         }
41
42     }
43

```

```
44 public int getAnzahlParteien() {
45     int anzahl = 0;
46
47     // Mehr als 100 sollen es ja wohl nicht sein ...
48     for( ; anzahl < 100; anzahl++ ) {
49         String pWert = "Partei" + anzahl;
50         String partei = getParameter( pWert );
51
52         if( partei == null ) // Schluss!!!
53             break;
54     }
55
56     return( anzahl );
57 }
58
59 public void getParametersParteien() {
60     // Array deklarieren
61     parteien = new String[ anzahlParteien ];
62
63     for(int i=0; i < parteien.length; i++) {
64         String pWert = "Partei" + i;
65         parteien[i] = getParameter( pWert );
66         // Wenn Parameter nicht vorhanden, dann ist Schluss
67         if( parteien[i] == null )
68             break;
69     }
70 }
71
72 public void getParametersProzente() {
73     // Array deklarieren
74     prozente = new String[ anzahlParteien ];
75
76     for(int i=0; i < prozente.length; i++) {
77         String pWert = "Prozent" + i;
78         prozente[i] = getParameter( pWert );
79         // Wenn Parameter nicht vorhanden, dann ist Schluss
80         if( prozente[i] == null )
81             break;
82     }
83 }
84
85 public void getParametersFarben() {
86     // Array deklarieren
87     parteiFarben = new Color[ anzahlParteien ];
88
89     for(int i=0; i < parteiFarben.length; i++) {
90         String pWert = "Farbe" + i;
91
92         String farbe = getParameter( pWert );
93         // Wenn Parameter nicht vorhanden, dann ist Schluss
94         if( farbe == null )
95             break;
96         else {
97             if( farbe.equals("Rot") )
98                 parteiFarben[i] = Color.red;
99             else if( farbe.equals("Schwarz") )
100                 parteiFarben[i] = Color.black;
101             else if( farbe.equals("Gruen") )
102                 parteiFarben[i] = Color.green;
103             else if( farbe.equals("Blau") )
```

```
104         parteiFarben[i] = Color.blue;
105     else
106         parteiFarben[i] = Color.lightGray;
107     }
108 }
109 }
110
111 public void paint( Graphics g ) {
112     // Position Text
113     int xpos1 = 10;
114     // Position Balken
115     int xpos2 = xpos1 + maxParteiLength + 5;
116     // Vertikale Startposition
117     int ypos = 10 + height;
118     int scale = 5;
119     int imgx = 0;
120     int imgy = 0;
121     if( img != null ) {
122         g.drawImage( img, xpos1, ypos, this );
123         imgx = img.getWidth(this);
124         imgy = img.getHeight(this);
125     }
126
127     g.setColor( Color.blue );
128     g.drawString("Wahlergebnisse", xpos1 + imgx + 5, ypos + imgy/2);
129
130     ypos += imgy+height;
131     for( int i = 0; i < parteien.length; i++ ) {
132         // Textausgabe
133         g.setColor(Color.black);
134         g.drawString( parteien[i], xpos1, ypos );
135
136         // Balkenausgabe
137         double wert = Double.parseDouble( prozente[i] );
138         int balkenLen = (int)(wert*scale);
139         g.setColor(parteiFarben[i]);
140         g.fillRect( xpos2, ypos-height/2, balkenLen, 10 );
141
142         // Prozentausgabe
143         g.setColor(Color.black);
144         g.drawString( prozente[i] + "_%", xpos2+balkenLen+5, ypos );
145
146         // Zeilensprung
147         ypos += height+5;
148     }
149
150 }
151 }
```

Übung 12

Projekte: Farb-Composer und „Tipp die Zahl“

12.1 Aufgaben

1. Schreiben Sie ein Applet, welches über drei Eingabefelder für die Rot-, Grün- und Blauwerte einer Farbe nach dem RGB-Modell (zulässige Eingabewerte 0 bis 255) verfügt und die jeweils eingestellte Farbe auf einer gesonderten Fläche (`java.awt.Canvas`) abbildet.

Außerdem sollte der Hexadezimalwert der gewählten Farbe, wie er in HTML-Quellcode verwendet wird (Beispiel: #FF00A4), jeweils durch ein Label angezeigt werden.

Sie finden ein Lösungsmuster unter 12.2 Hinweise

2. Schreiben Sie ein Applet, mit dem Sie „Tipp die Zahl“ spielen können.

Beschreibung:

Das Programm wählt eine Nummer im Bereich 1 - 1000 durch eine Zufallsauswahl. Dann gibt das Programm die Meldung aus:

Ich habe eine Nummer zwischen 1 und 1000 ausgewählt. Können Sie die Nummer erraten?

Geben Sie Ihren Rateversuch ein.

Ein `TextField` sollte für die Eingabe von Nummer zur Verfügung stehen. Per Button kann der Rateversuch zur Auswertung geschickt werden.

Sobald ein Rateversuch ausgewertet wurde, sollte das Applet die Hintergrundfarbe nach Rot oder Grün ändern. Grün zeigt an, dass der Spieler der Lösung einen Schritt näher gekommen ist und Rot zeigt an, dass der letzte Rateversuch ihn von der Lösung entfernt hat. Ein zweites nicht-editierbares Textfeld zeigt an, ob der Benutzer „zu hoch“ oder „zu tief“ geraten hat. Wenn die richtige Nummer getroffen wurde, sollte die Meldung „Richtig!!“ erscheinen und das Eingabefeld sollte nicht mehr editierbar sein.

Eine zusätzliche Schaltfläche sollte zur Verfügung stehen, mit der ein neues Spiel gestartet werden kann.

Sie finden ein Lösungsmuster unter 12.2 Hilfen

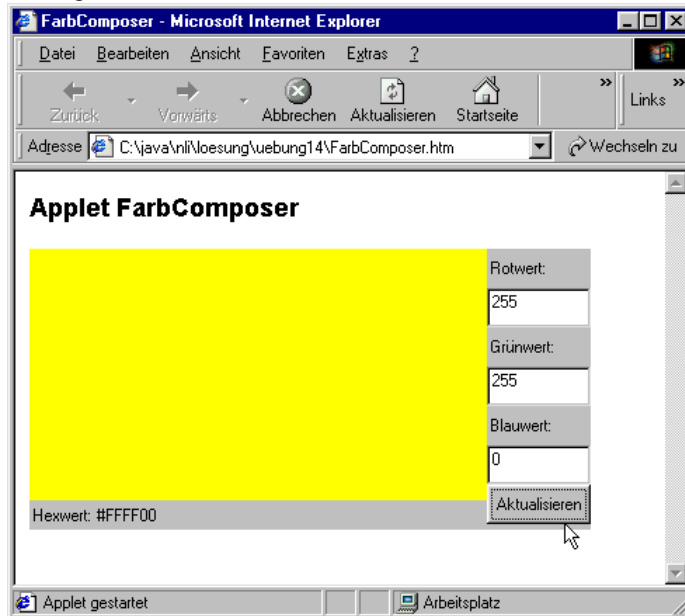
Unterstützende Materialien:

- 12.2 Hilfen

12.2 Hilfen

- zu 1. Sie müssen eine Subklasse von der Klasse `java.awt.Canvas` ableiten und die Methode `paint()` überschreiben, um ein Objekt vom Typ `Canvas` zu erzeugen (siehe auch Dokumentation).

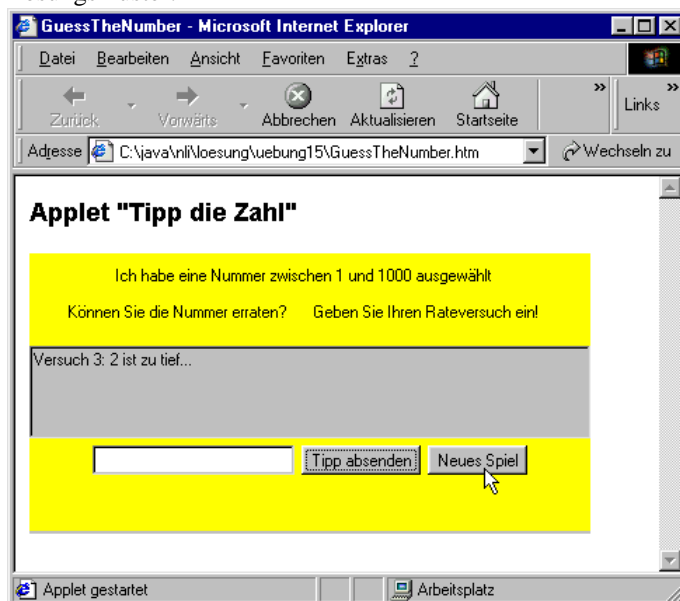
Lösungsmuster:



- zu 2. Zufallszahlen können Sie mit der Klasse `java.util.Random` generieren. Benutzen Sie die JDK-Dokumentation für Details.

Der Modulo-Operator ($10 \% 6 = 4$) bietet sich an, um den Wertebereich der Zufallszahlen einzuzugrenzen.

Lösungsmuster:



Übung 13

Behandlung von Ausnahmen - Java Exceptions

13.1 Aufgabe

Bearbeiten Sie die nachfolgenden Lernaufgaben in der angegebenen Reihenfolge.

1. Kopieren Sie das nachfolgend abgedruckte Java-Applet DivisionApp!

```
1 import java.awt.*;
2 import java.awt.event.*;
3 import java.applet.*;
4
5 public class DivisionsApp extends Applet {
6
7     Label prompt1, prompt2;
8     TextField input1, input2;
9     double result;
10
11    public void init() {
12        prompt1 = new Label("Geben_Sie_eine_Zahl_ein:");
13        input1 = new TextField( 10 );
14        prompt2 = new Label("Geben_Sie_eine_Teiler_ein_" +
15            "druecken_Sie_Enter:");
16        input2 = new TextField( 10 );
17        input2.addActionListener( new FieldActionListener() );
18        add( prompt1 );
19        add( input1 );
20        add( prompt2 );
21        add( input2 );
22    }
23
24    class FieldActionListener implements ActionListener {
25        public void actionPerformed( ActionEvent e ) {
26            int number1 = Integer.parseInt( input1.getText() );
27            input1.setText("");
28            int number2 = Integer.parseInt( input2.getText() );
29            input2.setText("");
30
31            double result = quotient( number1, number2 );
32
33            showStatus( number1 + "_/__" + number2 + "_=__" +
34                Double.toString( result ) );
35        }
36    }
37
38    public double quotient( int numerator, int denominator ) {
39        return( (double)numerator/((double)denominator) );
40    }
41 }
```

2. Testen Sie das Applet mit verschiedenen Ganzzahlen!

3. Geben Sie anstatt einer Zahl Buchstaben in einem Eingabefeld ein. Wie reagiert das Applet? Beheben Sie die Fehlerquelle unter Verwendung eines `try-catch`-Blocks
4. Benutzen Sie als Teiler die Zahl 0. Was passiert? Beheben Sie den Fehler durch Definition einer eigenen Exception-Klasse `DivideByZeroException`.
5. Testen Sie mit dem Applet die Wirkung einer `finally`-Klausel.

Unterstützende Materialien:

- 13.2 Hilfen

13.2 Hilfen

zu 2. Mit Ganzzahlen sollte Ihr Applet reibungslos funktionieren.

zu 3. Aus der Fehlermeldung, die vom Applet ausgegeben wird, können Sie ersehen, dass das Programm eine `java.lang.NumberFormatException` wirft, weil die Konvertierung in einen Zahlwert gescheitert ist. Mit Hilfe eines `try-catch`-Blocks können Sie diese Ausnahme abfangen.

```

class FieldActionListener implements ActionListener {
    public void actionPerformed( ActionEvent e ) {
        int number1, number2;
        try {
            number1 = Integer.parseInt( input1.getText() );
            input1.setText("");
            number2 = Integer.parseInt( input2.getText() );
            input2.setText("");
        }
        catch( NumberFormatException exception ) {
            showStatus( exception.toString() );
            return;
        }

        try {
            double result = quotient( number1, number2 );

            showStatus( number1 + "/" + number2 + "=" +
                Double.toString( result ) );
        }
        catch( DivideByZeroException exception ) {
            showStatus( exception.toString() );
        }
    }
}

```

zu 4. Eine Division durch Null ist nicht definiert. Ihr Applet behilft sich mit dem Resultat infinity (unendlich). Sie können diese Ausnahmesituation in Java durch das „Werfen eines Ausnahmefehlers“ abfangen.

So gehen Sie vor:

(a) Definieren Sie eine neue Klasse `DivideByZeroException`.

```

public class DivideByZeroException extends Exception {

    public DivideByZeroException() {
        super("Versuch_durch_Null_zu_teilen_gescheitert");
    }
}

```

(b) Erweitern Sie die Methode `quotient()` wie folgt:

```

public double quotient( int numerator, int denominator )
    throws DivideByZeroException {
    if( denominator == 0 )
        throw new DivideByZeroException();

    return( (double)numerator/(double)denominator );
}

```

(c) Schließen Sie die Benutzung der `quotient`-Methode innerhalb der Klasse `FieldActionListener` in einen `try-catch`-Block ein:

```
...
try {
    double result = quotient( number1, number2 );

    showStatus( number1 + " / " + number2 + " = " +
        Double.toString( result ) );
}
catch( DivideByZeroException exception ) {
    showStatus( exception.toString() );
}
}
```

...

(d) Speichern und Kompilieren Sie das veränderte Java-Programm. Testen Sie dann das Applet erneut.

zu 5. Ein try-catch-Block kann bei Bedarf um einen finally-Block ergänzt werden. Die Inhalte des finally-Blocks werden in jedem Fall (ob Ausnahme-Fehler oder auch nicht) abgearbeitet. Damit eignet sich der finally-Block für alle Arten von notwendigen Aufräum- und Abschlussaktionen.

Beispielsverwendung in der Klasse FieldActionListener:

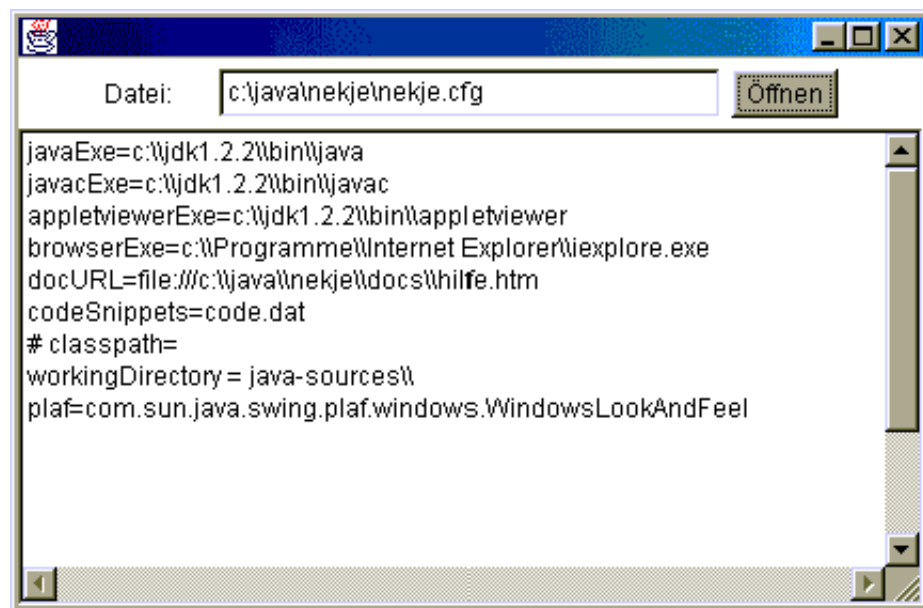
```
try {
    number1 = Integer.parseInt( input1.getText() );
    number2 = Integer.parseInt( input2.getText() );
}
catch( NumberFormatException exception ) {
    showStatus( exception.toString() );
    return;
}
finally {
    input1.setText( "" );
    input2.setText( "" );
    Toolkit.getDefaultToolkit().beep();
}
```

Übung 14

Von Dateien und Streams

14.1 Aufgaben

1. Schreiben Sie einen Text-Viewer nach dem abgebildeten Muster.



2. Erweitern Sie den Text-Viewer zu einem einfachen Editor-Programm, mit dem Sie auch Texte verändern, neu erstellen und dauerhaft extern speichern können.
3. Verwenden Sie ein Objekt der Klasse `java.awt.FileDialog` zum Öffnen und Abspeichern der Dateien.

Unterstützende Materialien:

- 14.2 Hilfen
- 14.3 Hinweise - Das Stream-Konzept, Dateien lesen, Dateien schreiben
- 14.4 Lösungen

14.2 Hilfen

zu 1. In der Anwendung (siehe Abbildung) werden die folgenden GUI-Elemente verwendet:

```

java.awt.TextArea
java.awt.Label
java.awt.Button
java.awt.TextField
    } java.awt.Panel
  
```

14.3 Hinweise

14.3.1 Das Stream-Konzept

Das Lesen und Schreiben von Dateien ist in Java Teil eines umfassenden Konzepts, welches die Kommunikation mit allen externen Quellen und Zielen umfasst.

Die Kommunikation mit externen Quellen und Zielen wird über Streams definiert. Zum Lesen aus einer externen Quellen muss grundsätzlich ein `InputStream` geöffnet werden.



Zum Schreiben in Richtung eines externen Ziels muss grundsätzlich ein `OutputStream` geöffnet werden.



In der Java-Bibliothek sind entsprechend auch die Klassen `java.io.InputStream` und `java.io.OutputStream` vorhanden.

Von diesen beiden Klassen abgeleitet sind darüber hinaus mehrere Subklassen definiert, die eine komfortable Arbeit mit Streams ermöglichen. Hinzukommen verschiedene Reader- und Writer-Klassen im Package `java.io`, die jeweils Stream-Objekte als Übergabeparameter erwarten und für eine leichte Handhabung von Schreib- und Leseoperationen sorgen.

14.3.2 Dateien lesen

Zum Lesen aus einer Textdatei sollten Sie ein Objekt der Klasse `java.io.BufferedReader` benutzen, welches ein Objekt der Klasse `java.io.FileReader` aufruft.

Beispiels-Deklaration:

```

BufferedReader infile = new BufferedReader( new FileReader( "c:\datei.txt" ) );
  
```

Verwenden Sie die Dokumentation zur Klasse `BufferedReader` für weitere Details.

Vergessen Sie nicht das `BufferedReader`-Objekt mit der Methode `close()` zu schließen, wenn Sie es nicht mehr gebrauchen.

14.3.3 Dateien schreiben

Sie finden Hinweise in der JDK-Dokumentation der Klassen `java.io.BufferedWriter`, `java.io.FileWriter` und `java.io.OutputStream`.

14.4 Lösungen

1. Ein einfacher Textviewer kann folgendes Aussehen haben:

```
1 import java.awt.*;
2 import java.awt.event.*;
3 import java.io.*;
4
5 public class SimpleTextViewer extends Frame {
6
7     protected TextField dateiFeld;
8     protected Button ladeButton;
9     protected TextArea textFeld;
10
11     public SimpleTextViewer() {
12
13         this.setLayout( new BorderLayout() );
14
15         Panel p = new Panel();
16         //p.setLayout( new GridLayout(1,3) );
17
18         Label prompt = new Label("Datei:_");
19         dateiFeld = new TextField(30);
20         ladeButton = new Button( "Öffnen" );
21         ladeButton.addActionListener( new LadeButtonListener() );
22
23         textFeld = new TextArea( 25, 2 );
24
25         p.add(prompt);
26         p.add(dateiFeld);
27         p.add(ladeButton);
28
29         add( p, BorderLayout.NORTH );
30         add(textFeld, BorderLayout.CENTER );
31
32         this.addWindowListener(
33             new WindowAdapter() {
34                 // Bei Programmexit-Event beende Programm
35                 public void windowClosing(WindowEvent e) {
36                     System.exit(0);
37                 }
38             }
39         );
40
41         this.setSize(600,400);
42         this.setVisible( true );
43     }
44
45     public static void main( String[] args ) {
46         new SimpleTextViewer();
47     }
48
49     class LadeButtonListener implements ActionListener {
50         public void actionPerformed( ActionEvent e ) {
51
52             String dateiName = dateiFeld.getText();
53
54             try {
55                 BufferedReader inputDatei = new BufferedReader(
56                     new FileReader(dateiName) );
57
```



```
58     try {
59         while( true ) {
60             String line = inputDatei.readLine();
61
62             if( line == null )
63                 break;
64             textFeld.append( line + "\n");
65         }
66
67         inputDatei.close();
68     }
69     catch( IOException ex2 ) {
70     }
71 }
72 catch( FileNotFoundException ex ) {
73     textFeld.setText("Fehler: _Datei_kann_nicht_geöffnet_werden.");
74 }
75 }
76
77 }
78
79 }
```

Übung 15

Multithreading

15.1 Aufgaben

Java unterstützt von Haus aus parallellaufende Teilprozesse (Threads) innerhalb eines Programms (Multithreading). Solche Threads können bei Betriebssystemen, die Multiprozessor-Hardware (z.B. Solaris, Linux) unterstützen, dann sogar auf mehrere Prozessoren aufgeteilt werden und wirklich parallel abgearbeitet werden. Auf Einprozessor-Systemen wird der Parallellauf simuliert.

Als Beispiel für eine Multithreading-Anwendung kopieren Sie das folgende Java-Programm und testen Sie dessen Wirkungsweise.

```
1 import java.awt.*;
2 import java.awt.event.*;
3
4 public class ThreadApplikation extends Frame {
5
6     ThreadApplikation myself;
7     TextArea tArea;
8     Button startButton;
9     Button stoppButton;
10
11     MyThread t1;
12     MyThread t2;
13     MyThread t3;
14
15     public ThreadApplikation() {
16
17         myself = this;
18
19         setLayout( new FlowLayout() );
20         tArea = new TextArea( 25,20 );
21
22         startButton = new Button("Start");
23         startButton.addActionListener( new StartListener() );
24         stoppButton = new Button("Stopp");
25         stoppButton.addActionListener( new StoppListener() );
26
27         add( tArea );
28         add( startButton );
29         add( stoppButton );
30
31         this.addWindowListener(
32             new WindowAdapter() {
33                 // Bei Programmexit-Event beende Programm
34                 public void windowClosing(WindowEvent e) {
35                     System.exit(0);
36                 }
37             }
38         );
39
40         this.setSize(600,400);
41         this.setVisible( true );
```

```
42
43
44 }
45
46 public void printText( String str ){
47     tArea.append( str );
48 }
49
50 public static void main( String[] args ) {
51     new ThreadApplikation();
52 }
53
54 class StartListener implements ActionListener {
55
56     public void actionPerformed( ActionEvent e ) {
57         t1 = new MyThread( myself, "Thread_1");
58         t2 = new MyThread( myself, "Thread_2");
59         t3 = new MyThread( myself, "Thread_3");
60         t1.start();
61         t2.start();
62         t3.start();
63     }
64 }
65
66 class StoppListener implements ActionListener {
67
68     public void actionPerformed( ActionEvent e ) {
69         try {
70             t1.interrupt();
71             t2.interrupt();
72             t3.interrupt();
73             t1 = null;
74             t2 = null;
75             t3 = null;
76         }
77         catch( SecurityException ex ){
78             System.exit(1);
79         }
80     }
81 }
82
83
84 }
85
86 class MyThread extends Thread {
87
88     private String myName;
89     private int counter;
90     private ThreadApplikation p;
91
92     public MyThread( ThreadApplikation parent, String name ) {
93         p = parent;
94         myName = name;
95         counter = 0;
96     }
97
98     public void run() {
99         while( !interrupted() ) {
100             counter++;
101             String str = myName + ":_ " + counter + "\n";
```

```
102     p.printText( str );
103     }
104     }
105 }
```


Übung 16

Projekt: Java und Datenbanken

16.1 Aufgabe

Schreiben Sie eine Applikation, mit der Sie eine Adressdatenbank auf einem SQL-Server verwalten können. Verwenden Sie dafür Klassen aus dem Package `java.sql`.

Die folgende Abbildung zeigt einen Vorschlag für ein User-Interface zur Verwalten der Datenbank.



The image shows a Java Swing window titled "Adressdatenbank". The window has a blue title bar with standard window controls (minimize, maximize, close). The main content area contains four text input fields, each with a label to its left: "Name:", "Vorname:", "Strasse:", and "Ort:". Below the input fields is a horizontal toolbar with six buttons: "Vor", "Zurück", "Neu", "Editieren", "Speichern", and "Löschen".

Unterstützende Materialien:

- 16.2 Hinweise - Anlegen einer leeren Access-Datenbank, Installation des ODBC-Treibers, Eine Verbindung zur Datenbank aufbauen, Eine Tabelle einrichten, Datensätze speichern, Datensätze auslesen, Die Klasse JDBCMySQLDemo

16.2 Hinweise

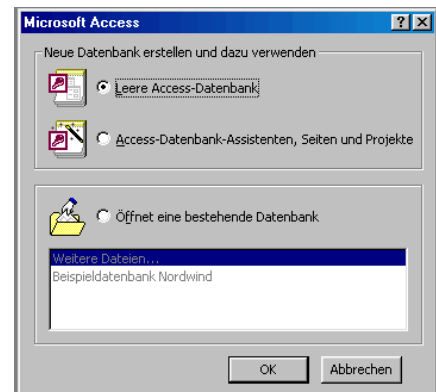
16.2.1 Anlegen einer leeren Access-Datenbank

Voraussetzung für die Nutzung einer Access-Datenbank in Java ist das Vorhandensein einer solchen Datenbank in Form einer MDB-Datei.

Die folgenden Schritte beschreiben, wie Sie in MS Access eine leere Access-Datenbank anlegen.

Start von MS Access

Starten Sie MS Access und wählen Sie aus dem Eingangsdialog die Option „Leere Access-Datenbank“. Bestätigen Sie die Auswahl mit „OK“.

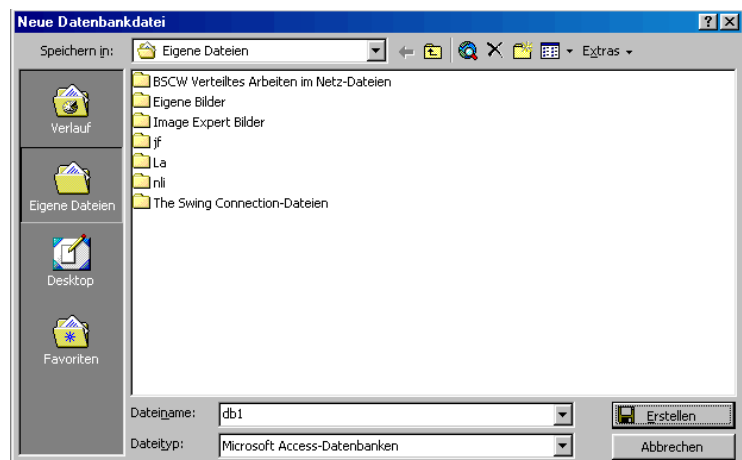


Leere Datenbank anlegen

Vergeben Sie im anschließenden Datei-Dialog einen geeigneten Dateinamen für die Access-Datenbank.

Bei Drücken des Auswahlbuttons „Erstellen“ wird eine neue Access-Datenbank im ausgewählten Verzeichnis angelegt.

Beenden Sie anschließend das Programm MS Access.



16.2.2 Installation des ODBC-Treibers für eine Access-Datenbank

Damit Sie von Java aus auf eine Access-Datenbank zugreifen können, muß jetzt ein ODBC-Treiber für die soeben angelegte Datenbank installiert werden.

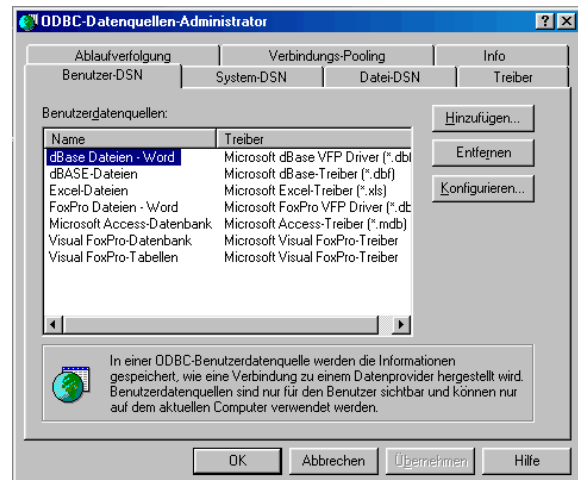
ODBC-Installationsprogramm starten

Sie finden das ODBC-Installationsprogramm unter „Start | Einstellungen | Systemsteuerung“. Durch Doppelklick auf dem Icon „ODBC-Datenquellen“ wird das Installationsprogramm gestartet.



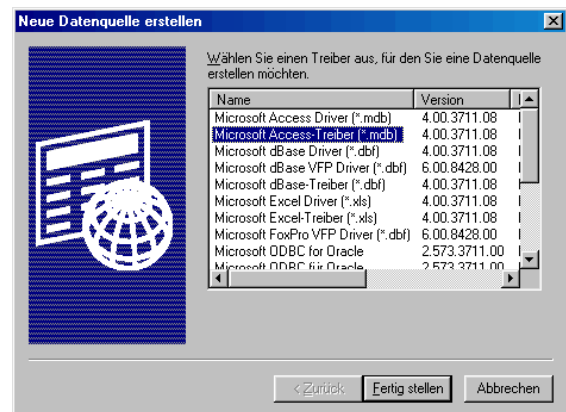
Hinzufügen einer Benutzerdatenquelle

Im ODBC-Datenquellen-Administrator wählen Sie die Option „Hinzufügen...“.



Auswahl eines ODBC-Datenbank-Treibers

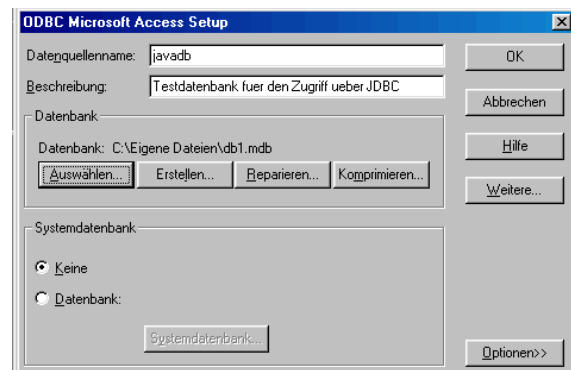
Es öffnet sich der Dialog „Neue Datenquelle erstellen“. Wählen Sie hier den „Microsoft Access-Treiber“ und drücken Sie den Button „Fertigstellen“.



Einrichten des ODBC-Treibers

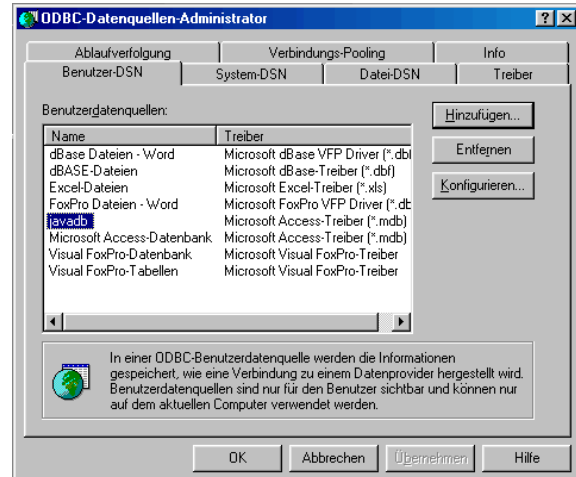
Im Dialog „ODBC Microsoft Access Setup“ nehmen Sie die Einstellungen nach dem abgebildeten Muster vor. Wichtig: Sie müssen die Datenbank über einen Datei-öffnen-Dialog mit dem ODBC-Treiber verbinden (Option „Datenbank: Auswählen...“).

Sie können auch an dieser Stelle über die Option „Datenbank: Erstellen...“ eine neue leere Datenbank einrichten.



Rückkehr in den ODBC-Datenquellen-Administrator

Bei Rückkehr in den ODBC-Datenquellen-Administrator erkennen Sie, dass nunmehr für die ausgewählte Datenquelle ein ODBC-Treiber installiert wurde.



16.2.3 Eine Verbindung zur Datenbank aufbauen

Nachfolgend ein Beispielprogramm, das demonstriert, wie eine Verbindung zur Datenbank aus einer Java-Applikation heraus hergestellt werden kann.

```

1 import java.sql.*;
2
3 public class JdbcAccessDemo {
4
5     Connection conn;
6     final String driver = "sun.jdbc.odbc.JdbcOdbcDriver";
7     final String dbConnect = "jdbc:odbc:";
8
9     public JdbcAccessDemo() {
10         try {
11             Class.forName( driver );
12         }
13         catch( ClassNotFoundException e ){
14             System.err.println("Kann_JDBC-Treiber_nicht_finden" );
15         }
16     }
17
18     public boolean makeConnection(String dbName) {
19         boolean status = false;
20         try{
21             conn = DriverManager.getConnection( dbConnect+dbName, "", "" );
22             System.out.println("Verbindung_zu_' +dbName + "'_hergestellt.");
23             status = true;
24         }
25         catch( SQLException e ) {
26             System.err.println("Verbindungsaufbau_gescheitert.");
27         }
28
29         return(status);
30     }
31
32     public void closeConnection() {
33
34         if( conn != null ) {
35             try {
36                 conn.close();

```



```

37         System.out.println( "DB-Verbindung_wurde_geschlossen." );
38     }
39     catch( Exception e ){
40         System.err.println("Fehler_beim_Schliessen_der_DB-Verbindung." );
41     }
42 }
43 }
44
45
46 public static void main( String[] args ) {
47     JdbcAccessDemo demo = new JdbcAccessDemo();
48
49     demo.makeConnection("javadb");
50     //...
51     demo.closeConnection();
52 }
53 }

```

Erläuterungen:

Zeile 1 enthält die Importanweisung für Klassen aus dem Package „java.sql“. In diesem Paket sind alle Klassen von Java-JDBC zusammengefaßt. Das Paket gehört seit Java 1.2 zur Standardbibliothek.

In Zeile 5 bis 7 werden globale Datenfelder deklariert, die später Verwendung finden.

Die Zeilen 46 bis 53 enthalten die Methode `main()`. Die Main-Methode wird beim Programmstart aufgerufen. In ihr wird zunächst aus dem Konstruktor ein Objekt vom Typ `JdbcAccessDemo` erzeugt, danach eine Verbindung zur Datenbank hergestellt und abschließend diese Verbindung wieder gelöst.

Die Zeilen 9 bis 16 enthalten den Konstruktor der Klasse `JdbcAccessDemo`. Bemerkenswert ist insbesondere Zeile 11. Hier wird mit der Methode `Class.forName()` die Klasse für den JDBC-ODBC-Datenbanktreiber dynamisch geladen. Der Klassenname wird der globalen Konstante `driver` entnommen. Für jede Datenbank-Engine muß ein spezifischer JDBC-Treiber geladen werden. Treiber werden i. d. R. vom Hersteller bereitgestellt.

Zeile 49 der Main-Methode enthält den Aufruf der Methode `makeConnection()`. Diese Methode (Zeile 18 bis 30) stellt unter Benutzung der von `DriverManager.getConnection()` (statische Methode aus der Klasse `DriverManager`) die Verbindung zur Datenbank her. Übergeben werden der Methode `getConnection()` der Bezeichner der Datenbank (ggfs. mit Netzwerkadresse), der Loginname und das Passwort. Da eine MS-Access-Datenbank standardmäßig keinen Loginnamen und kein Passwort verlangt, werden in diesem Fall leere Strings übergeben. Die Methode `getConnection()` erzeugt bei erfolgreicher Verbindungsaufnahme ein Objekt vom Typ `Connection`, in dem die Eigenschaften der Verbindung gekapselt werden. Eine Referenz auf das `Connection`-Objekt wird im Datenfeld `conn` abgelegt.

Die Zeilen 32 bis 43 enthalten die Methode `closeConnection()`. In ihr wird die Datenbank-Verbindung durch Aufruf der Methode `close()` aus der Klasse `Connection` wieder gelöst.

16.2.4 Eine Tabelle einrichten

Es wird unterstellt, dass die Access-Datenbank zunächst leer ist. Es muß also mindestens eine Tabelle eingerichtet werden, um mit der Datenbank arbeiten zu können.

Hier ein Beispiel für eine Methode, mit der eine primitive Adresstabelle errichtet wird:

```

1 public void createAddressTable(){
2
3     final String createT =

```

```

4     "CREATE_TABLE_address_(id_INTEGER, _name_VARCHAR(50), _vorname_VARCHAR(50), _" +
5     "strasse_VARCHAR(50), _ort_VARCHAR(50), _plz_VARCHAR(5));";
6
7     try {
8         Statement s = conn.createStatement();
9         int num = s.executeUpdate( createT );
10        System.out.println( "Create-Table-Rueckgabe:_" + num );
11        s.close();
12    }
13    catch( Exception e ) {
14        System.err.println( "Fehler:_" + e.getMessage() );
15    }
16 }

```

Erläuterungen:

In den Zeilen 3 bis 5 wird die SQL-Anweisung, die für das Anlegen der Tabelle genutzt werden soll, in einem konstanten Datenfeld gespeichert.

In Zeile 8 wird durch Aufruf der Methode `getStatement()` aus der Klasse `Connection` (wir benutzen hier mit `conn` das `Connection`-Objekt der Klasse `JdbcAccessDemo` aus dem vorhergehenden Kapitel) ein Objekt vom Typ `Statement` deklariert und dem Referenzfeld `s` zugewiesen. Das `Statement`-Objekt verfügt über Methoden, mit denen Anfragen an Datenbanken gerichtet werden können.

In Zeile 9 wird die Methode `executeUpdate()` des `Statement`-Objekts benutzt, um die Anfrage (`CREATE TABLE`) an die Datenbank zu senden.

In Zeile 11 wird das `Statement`-Objekt geschlossen.

Hinweis: `Statement`-Objekte sind wiederverwendbar. Es spricht also nichts dagegen, ein `Statement`-Objekt mehrfach zu benutzen. Hier erfolgt zu Demonstrationszwecken nur eine einmalige Verwendung.

16.2.5 Datensätze speichern

Ohne Inhalt ist die Datenbank ziemlich wertlos. Darum hier ein Beispiel, wie Datensätze in die Datenbank eingefügt werden können.

```

1     public void insertData() {
2
3         final String[] data = {
4             "'Schmidt', _'Helmut', _'Soesteweg_2', _'Cloppenburg', _'46771'",
5             "'Albers', _'Gesina', _'Nordstr._4', _'Hamburg', _'20000'",
6             "'Lodders', _'Alwine', _'Pappelweg_57', _'Muenchen', _'80000'"
7         };
8
9         final String insert = "INSERT_INTO_address_VALUES_";
10        int num = 0;
11
12        try {
13            Statement s = conn.createStatement();
14            for( int i=0; i < data.length; i++ )
15            {
16                num += s.executeUpdate( insert + "(" + i + ",_" + data[i] + ");" );
17            }
18
19            System.out.println( "Insert-Rueckgabe:_" + num );
20            s.close();
21        }

```

```
22     catch( Exception e ) {
23         System.err.println("Fehler:_" + e.getMessage() );
24     }
```

Erläuterungen:

In den Zeilen 3 bis 7 werden die Datensätze in den drei Feldern eines `String`-Arrays gespeichert.

Zeile 9 enthält die SQL-Anweisung, die für das Einfügen der Datensätze benötigt wird.

In Zeile 13 wird ein `Statement`-Objekt erzeugt.

Die Zeilen 14 bis 17 beinhalten den Code, der das Einfügen der Datensätze erledigt. Benutzt wird die Methode `executeUpdate()` des `Statement`-Objekts.

16.2.6 Datensätze auslesen

Nachdem nunmehr die Datenbank mit Inhalt gefüllt ist, kann Sie benutzt werden. Nachfolgend ein Beispiel, wie die Datensätze aus einer Tabelle ausgelesen werden können.

```
1  public void selectAllData() {
2
3      final String select = "SELECT_*_FROM_address;";
4
5      try {
6          Statement s = conn.createStatement();
7
8          ResultSet rs = s.executeQuery( select );
9          ResultSetMetaData md = rs.getMetaData();
10
11         int numcols = md.getColumnCount();
12
13         for(int i=0; i < numcols; i++)
14             System.out.print( md.getColumnLabel(i+1) + ",_" );
15         System.out.print("\n");
16
17         while( rs.next() ) {
18             for(int i=0; i < numcols; i++)
19                 System.out.print( rs.getString(i+1) + ",_" );
20             System.out.print("\n");
21         }
22         s.close();
23     }
24     catch( Exception e ) {
25         System.err.println("Fehler:_" + e.getMessage() );
26     }
27 }
```

Erläuterungen:

Zeile 3 enthält die SQL-Anweisung für das Auslesen aller Datensätze aus einer bestimmten Tabelle.

In Zeile 6 wird ein `Statement`-Objekt erzeugt.

In Zeile 8 wird die Methode `executeQuery()` des `Statement`-Objekts benutzt, um die Anfrage an die Datenbank zu senden. Die Methode `executeQuery()` liefert bei Erfolg ein Objekt vom Typ `ResultSet` zurück, in dem die gefundenen Datensätze gekapselt sind.

In Zeile 9 wird das `ResultSet`-Objekt genutzt, um mit der Methode `getMetaData()` weitere Informationen über die gefundenen Datensätze in einem implizit erzeugten Objekt vom Typ `ResultSetMetaData` abzulegen.

In Zeile 11 wird über das `ResultSetMetaData`-Objekt die Anzahl der Spalten in den Datensätzen erfragt.

In Zeile 13 bis 15 werden die Spaltenbezeichnungen ausgegeben. Es wird die Methode `getColumnLabel()` des `ResultSetMetaData`-Objekts benutzt.

Zeile 17 bis 21 demonstriert, wie die einzelnen Datensätze mit der Methode `next()` des `ResultSet`-Objekts nacheinander eingelesen und auf dem Bildschirm ausgegeben werden können.

16.2.7 Die Klasse `JdbcMySQLDemo`

Abschließend noch ein Beispiel, wie auf eine MySQL-Datenbank über ein lokales Netzwerk oder das Internet zugegriffen werden kann. MySQL ist die Datenbank-Engine, die von den meisten Host-Providern (in Deutschland z.B. Strato, Puretec usw.) für Kunden zur Verfügung gestellt wird (Bezugsquelle: <http://www.mysql.org>)

Der Quelltext unterscheidet sich im Grundsatz nicht von dem der Klasse `JdbcAccessDemo`. Geändert wurden der Aufruf für die Methode `makeConnection()` und die spezifischen Daten für den JDBC-Treiber und die Adressierung der Datenbank. Außerdem wurde das `Statement`-Objekt global für die gesamte Klasse gesetzt.

Damit die Applikation funktioniert, muss zum Ausführungszeitpunkt der Datenbanktreiber im Classpath der JVM liegen.

Die verwendeten Adressdaten:

Treiber: `org.gjt.mm.mysql.Driver` (Quelle: www.worldserver.com/mm.mysql)
Adressdaten
Datenbank: `nli01`
Server: `ins.bbs-loeningen.de`
Login: `nli01`
Passwort: `nli01`

```
1 import java.sql.*;
2
3 public class JdbcMysqlDemo {
4
5     Connection conn;
6     Statement s;
7     final String driver = "org.gjt.mm.mysql.Driver";
8     final String dbConnect = "jdbc:mysql:";
9
10    public JdbcMysqlDemo() {
11        try {
12            Class.forName( driver );
13        }
14        catch( ClassNotFoundException e ){
15            System.err.println("Kann_JDBC-Treiber_nicht_finden" );
16        }
17    }
18
19    public boolean makeConnection(String dbName, String login, String pwd) {
20        boolean status = false;
```

```
21     try{
22         conn = DriverManager.getConnection( dbConnect+dbName, login, pwd );
23         s = conn.createStatement();
24
25         System.out.println("Verbindung_zu_" + dbName + "'_hergestellt.");
26         status = true;
27     }
28     catch( SQLException e ) {
29         System.err.println("Verbindungsaufbau_gescheitert.");
30     }
31
32     return(status);
33 }
34
35 public void closeConnection() {
36
37     if( conn != null ) {
38         try {
39             if( s != null )
40                 s.close();
41             conn.close();
42             System.out.println( "DB-Verbindung_wurde_geschlossen." );
43         }
44         catch( Exception e ){
45             System.err.println("Fehler_beim_Schliessen_der_DB-Verbindung.");
46         }
47     }
48 }
49
50 public void createAddressTable(){
51
52     final String createT =
53     "CREATE_TABLE_address_(id_INTEGER, _name_VARCHAR(50), _vorname_VARCHAR(50), _" +
54     "strasse_VARCHAR(50), _ort_VARCHAR(50), _plz_VARCHAR(5));";
55
56     try {
57         int num = s.executeUpdate( createT );
58         System.out.println( "Create-Table-Rueckgabe:_ " + num );
59     }
60     catch( Exception e ) {
61         System.err.println("Fehler:_ " + e.getMessage() );
62     }
63 }
64
65 public void insertData() {
66
67     final String[] data = {
68         "'Schmidt', '_Helmut', '_Soesteweg_2', '_Cloppenburg', '_46771'",
69         "'Albers', '_Gesina', '_Nordstr._4', '_Hamburg', '_20000'",
70         "'Lodders', '_Alwine', '_Pappelweg_57', '_Muenchen', '_80000'"
71     };
72
73     final String insert = "INSERT_INTO_address_VALUES_";
74     int num = 0;
75
76     try {
77         for( int i=0; i < data.length; i++ )
78             {
79                 num += s.executeUpdate( insert + "(" + i + ",_" + data[i] + ");" );
80             }

```

```
81
82     System.out.println( "Insert-Rueckgabe:_" + num );
83 }
84 catch( Exception e ) {
85     System.err.println("Fehler:_" + e.getMessage() );
86 }
87 }
88
89 public void selectAllData() {
90
91     final String select = "SELECT_*_FROM_address;";
92
93     try {
94
95         ResultSet rs = s.executeQuery( select );
96         ResultSetMetaData md = rs.getMetaData();
97
98         int numcols = md.getColumnCount();
99
100        for(int i=0; i < numcols; i++)
101            System.out.print( md.getColumnLabel(i+1) + ",_" );
102        System.out.print("\n");
103
104        while( rs.next() ) {
105            for(int i=0; i < numcols; i++)
106                System.out.print( rs.getString(i+1) + ",_" );
107            System.out.print("\n");
108        }
109    }
110    catch( Exception e ) {
111        System.err.println("Fehler:_" + e.getMessage() );
112    }
113 }
114
115 public static void main( String[] args ) {
116     JdbcMysqlDemo demo = new JdbcMysqlDemo();
117
118     // Datenbankadresse, User-Login, Passwort
119     demo.makeConnection("//ins.bbs-loeningen.de/gast1",
120         "gast1", "gast1");
121
122     demo.createAddressTable();
123     demo.insertData();
124     demo.selectAllData();
125
126     demo.closeConnection();
127 }
128 }
```

Anhang A

Ausgewählte Java-Ressourcen im Internet

A.1 Allgemeines

http://java.sun.com http://www.javasoft.com	Zentrale Java-Site im Netz mit den neuesten Entwicklungen und JDKs von Sun Microsystems
http://www.ibm.com/developer/java	Umfangreiche Hilfestellung zum Thema Java von Big Blue
http://www.faqs.org/faqs/de/comp-lang-java/faq/	Häufig gestellte Fragen aus der deutschen Java-Newsgroup (de.comp.lang.java)
http:// java.seite.net	Kaffee & Kuchen - Deutsche Java-Site mit Linksammlung

A.2 Bücher/Tutorials

http://www.javabuch.de	Online-Java-Handbuch (ex Goto Java 2) (dt.)
http://www.informit.de/books/java2_komp/data/start.htm	Online-Buch: Java 2 Kompendium (dt.)
http://www.mindview.net/Books	Online-Buch: Thinking in Java (engl.)
http://java.sun.com/docs/books/tutorial/	Java-Einführung vom Java-Entwickler-Team (engl.)
http://www.boku.ac.at/javaeinf/	Kursunterlage von Java-Guru Hubert Partl (dt.)
http://java-tutor.com/javabuch/	„Java ist auch eine Insel“ - Umfangreicher Javakurs (dt.)

A.3 Online-Kurse

http://java.rrzn.uni-hannover.de	Einige Online-Kurse und Materialien
---	-------------------------------------

A.4 Entwicklungsumgebungen (IDEs)

http://www.nekje.de	Eigene Produktion und Freeware - Schwerpunkt Ausbildung
http:// www.bluej.org	BlueJ - Monash University Australien, frei für Bildungsinstitute, deutsche Version verfügbar
http://www.fantastic-bits.de	JOE - Freeware, deutschsprachig
http://www-pinot.informatik.uni-kl.de/~hillenbr/jedi	Freeware, deutschsprachig
http://jedit.sourceforge.net/	Jedit - Open Source Programm

A.5 Ressourcen-Sites

http://www.jars.com	Tausende von Java-Entwicklungen mit Bewertungen
http://www.java-software.de	Datenbank mit Java-Software

A.6 Newsgroups/Diskussionsforen

de.comp.lang.java	deutschsprachige Newsgroup zu Java
http://www.spotlight2.de/foren/jav/forum_jav.htm	Gut frequentiertes Webforum zu Java

A.7 Didaktik und Methodik

http://www-is.informatik.uni-oldenburg.de/~dibo/hamster/index.html	Der Java-Hamster - Modell, mit dessen Hilfe Grundkonzepte der Programmierung am Beispiel von Java auf spielerische Art und Weise erlernt werden können.
---	---