Kapitel 4 - Einfache Beispielprogramme

Klassen
Die public-Klasse muss mit dem Namen der Datei übereinstimmen.
Es ist nur eine Public-Classe pro Datei erlaubt.

Symbolische Konstanten
Werden mit „final“ deklariert und werden gross geschrieben

System.in.read
Legt bei jedem Aufruf als Rückgabewert das nächste Zeichen des Eingabestroms in einer Variablen ab.

Konstruktoren
Tragen den selben Namen wie die Klasse und haben keinen Rückgabewert.

Kapitel 5 - Datentypen und Variablen

Einfache (elementare) Datentypen
boolean, numerischer Typen: Integertypen, Gleitpunkttypen

Typ	Inhalt	Wertebereich
boolean	true oder false	True oder false
char	16bit Unicode Zeichen	0 bis 65535
byte	8Bit-Ganzzahl	-128 bis 127
short	16Bit-Ganzzahl	-2E15 bis 2E15-1
int	32Bit-Ganzzahl	-2E31 bis 2E31-1
long	64Bit-Ganzzahl	-2E63 bis 2E63-1
float	Gleitpunkttyp	-3.4*10E38 bis 3.4*10E38
double	Gleitpunkttyp	-1.7*10E308 bis 1.7*10E308

Float: 1 Vorzeichenbit (Bit 31),
8 Bits Exponenten (Bit 23-30),
23 Bits Mantisse (Bit 0 bis 22)
7 Stellen nach dem Komma
Double: 1 Vorzeichenbit (Bit 63),
11 Bits Exponenten (Bit 52-62),
52 Bits Mantisse (Bit 0 bis 51)
Rund 15 Stellen nach dem Komma

Das höchste bit gibt das Vorzeichen an.
Ist es 0 so ist der Wert positiv, ansonsten negativ.

Floatwerte müssen immer mit einem “f“ enden. -> 5.3f

Referenztypen (abstrakte Datentypen)

Klassen-Typ, Array-Typ, Schnittstellentyp

Klassen-Typen

Werden mit dem Schlüsselwort „class“ definiert
Beginnen definitionsgemäss mit einem Grossbuchstaben

Methoden

Anweisungsfolge, die unter einem Namen gespeichert ist.
Muss innerhalb einer Klasse deklariert werden.
Besteht aus einem Methodenkopf und einem Methodenrumpf.
Der Methodenkopf gibt dem Compiler die Aufrufschnittstelle bekannt.
Haben zugriff auf die Datenfelder desselben Objekts.
Die Namen werden klein geschrieben.

Variablen

Variablen können einen beliebigen Namen tragen.
Einschränkungen sind:
Es sind nur „_“ und „$“ als Sonderzeichen erlaubt.
Eine Variable darf nicht mit einem Buchstaben beginnen.
Zudem darf der Name kein Java-Schlüsselwort sein.

Statische Variablen

Lebensdauer erstreckt sich über den Zeitraum der Abarbeitung der Methode bzw. des Blocks, zu dem sie gehört.

Dynamische Variablen

Gültigkeit und Lebensdauer einer dynamischen Variablen wird nicht durch die Blockgrenzen, d.h. durch die statische Struktur des Programms bestimmt. Dies sind immer Objekte.

Referenzen

Der Zugriff auf dynamische Variablen erfolgt über Referenzen.
Vergleichbar mit einer Fernbedienung am Fernseher.
Referenzen können einander zugewiesen werden.
Der Zugriff auf eine nicht definierte Referenz muss vom Compiler verhindert werden.

Objekte

Mit dem new-Operator wird ein Objekt ohne Namen auf dem Heap erzeugt. Dann wird eine Referenz auf das Objekt zurückgeben. Diese Referenz kann dann einer Referenzvariablen zugewiesen werden.

Klassen / Instanz- und Lokale Variablen

Klassenvariablen werden für die Klasse einmal angelegt.
Sie können von überall her aufgerufen werden.
Instanzvariablen gibt es für jedes Objekt, sie können nur über eine Referenz aufgerufen werden.

Lokale Variablen gibt es in Methoden. Die Gültigkeit erstreckt sich dann auf den Block, indem sie sich befindet.

Konstante Variablen
Werden mit dem Schlüsswort „final“ angelegt. Objekte können nicht final gemacht werden. Referenzen dagegen schon.

Komposition
Lebensdauer des zusammengesetzten Objektes ist gleich, wie diejenige seiner Komponenten.

Aggregation
Die Teile können länger leben, als das ganze.

Protokoll
Mögliche Wege, wie mit einer Methode gesprochen werden kann.

Speicherbereiche für Variablen

Stack
Wird auch als Stapel bezeichnet. Es kann nur immer auf die zuletzt abgelegte Variable zugegriffen werden. (LIFO)

Heap
Speicherplatz für dynamische Variablen. Die Organisation des Heaps übernimmt die virtuelle Maschine. Die Freigabe der nicht mehr benötigten Objekte erfolgt durch den Garbage Collector.

Method-Area
Speicherbereich für Klassenvariablen und Programmcode.

Modifikatoren

Dies sind:
public, private, protected, static, final, transient, volatile, abstract, native, synchronized

Arrays

Objekt aus Komponenten, wobei jedes Element vom selben Typ ist.
Werden zur Laufzeit auf dem Heap angelegt. Die Länge kann nachträglich nicht verändert werden.
Es ist nicht möglich über die Grenzen eines Arrays hinaus zu schreiben.
Arrays sind von der Klasse Object abgeleitet und erben alle Methoden.

Array aus einfachen Datentyps

```
byte[ ] bArray;   // Erstellen einer Referenzvariablen auf ein Array-Objekt
bArray = new byte[4]; // Erstellen Byte-Array Länge 4 und Zuweisung
bArray [2] = 6;  // Zuweisung des Wertes 6 an die Positon 1
```
Zugriff auf das n-te Element über n-1!

Die Schritte können auch zusammengefasst werden:

```
byte[ ] bArray = new byte [ 4];     oder auch
byte[ ] bArray = {1, 2, 3, 4}
```

Array aus Referenztypen

Deklaration erfolgt gleich, wie beim Array aus einfachen Datentypen.
Der Unterschied ist, dass die Komponenten jeweils Referenzen auf den jeweiligen Klassentyp darstellen.
Bei der Zuweisung müssen also bereits initialisierte Referenzvariablen zugeordnet werden.

Mehrdimensionale Arrays

Sind Arrays aus Arrays.
Deklaration: int [ ] [ ] [ ] dreiDArray = new int [10] [20] [30];
Arrays können auch offen sein. Jedoch muss die Länge des ersten Arrays defniert sein und es darf kein definierter Array auf einen undefinierten folgen.

Konstante und Variable Zeichenketten

String steht für konstante Zeichenketten.
StringBuffer steht für variable Zeichenketten.

Strings können auf die gleiche Weise, wie Objekte erzeugt werden. Doch ist es auch möglich String name = “Anja“; zu schreiben. So erfolgt eine implizite Erzeugung eines Objektes.

Um Strings zu vergleichen, wird „“equals“ verwendet.
So überprüft z.B. “x.equals(y)“ die Gleichheit des Strings x mit dem String y und gibt einen boolean-Wert zurück.

Wrapperklassen

Stellen Methoden zur Verfügung, die die Bearbeitung von einfachen Datentypen ermöglichen.

Einfache Datentypen	Wrapper-Klassen
char	Character
boolean	Boolean
byte	Byte
short	Short
int	Integer
long	Long
double	Double
float	Float

Deklaration: Integer i1 = new Integer (1);

Verkettung von Strings und Variablen anderer Datentypen

Die Variablen werden jeweils implizit in eine Wrapper-Klasse verpackt und danach ebenfalls implizit mit der .toString() Methode umgewandelt.

Kapitel 6 - Ausdrücke und Operatoren

Ausrücke
Alles, was einen Wert zurückliefert, stellt einen Ausdruck dar.

Operatoren und Operanden
Ein einstelliger (unärer) Operator hat einen einzigen Operanden

Benötigt ein Operator zwei Operatoren zur Verknüpfung, so spricht man vom zweistelligen (binären) Operator.

Präfix und Postfix stellt die Art dar, auf die sich ein unärer Operator auf den Operanden auswirkt.

Auswertungsreihenfolge
Zuerst werden die Ausdrücke in den **Klammern** ausgewertet.
Dann folgen die **unären** Operationen von rechts nach links
Zuletzt werden die Teilausdrücke mit **mehrstelligen** Operatoren ausgewertet.

Einstellige arithmetische Operatoren
++, --
Je nach Position des Operators (Präfix, Postfix)

Zweistellige arithmetische Operatoren
+, -, *, /, %
Die vier Grundoperationen plus der Restwertoperator “%“.

Zuweisungsoperatoren
+=, -=, /=, *=, %=, &=, |=, ^=, <<=, >>=, >>>=
Jeweils ein beliebiges Operationszeichen gefolgt von einem “=“.
Die Bedeutung ist wie folgt zu verstehen: a /= b ⇔ a = a / b

Relationale Operatoren
<, >, ==, !=, <=, >=
Werden auch als Vergleichsoperatoren bezeichnet.
Benötigen zwei Operanden. Bei unterschiedlichen, aber verträglichen Datentypen, wird eine implizite Typenkonvertierung vorgenommen.
Rückgabewert ist immer Boolean.

Logische Operatoren
&&, &, ||, |, !
Können nur auf Operanden vom Typ Boolean angewandt werden.
Beim Operator “&“ wird der rechte Operand immer ausgewertet, egal ob der linke Wert bereits schon “false“ war. Dasselbe gilt auch für den “|“-Operator.

Bit-Operatoren
&, |, ^, ~, >>, <<, >>>
"^" ist das das Exklusiv-Oder auf Bitebene. ">>>" der vorzeichenlose Bit-Shift. Beim vorzeichenlosen Bit-Shift werden jeweils 0en von links her nahchgefüllt.

Der Bedingungsoperator
A ? B : C
Wenn A, return B, sonst return C
B und C müssen zuweisungskompatibel sein.

Der cast-Operator
Mit dem cast-Operator wird eine explizite Typenkonvertierung durchgeführt Möglich sind Wandlungen zwischen numerischen Datentypen und zwischen Referenztypen.
Werden Floatzahlen definiert, so müssen diese entweder gecastet, oder dann mit einem "f" deklariert werden. Ansonsten meldet der Compiler einen Fehler.

Typenkonvertierung
Implizite Typenkonvertierungen erfolgen nur von einem schmalen in einen breiteren Typen. Dagegen können explizite Wandlungen auch von breiten Typen in schmale erfolgen. Dabei können jedoch Informationsverluste entstehen.
Beim konvertieren in den Typ "char" werden negative Zahlen in positive Umgewandelt. Int -1 ist zum Beispiel char 65535

Gleitpunkt nach Integer
Die Stellen nach dem Komma werden abgeschnitten

Der Gleichheitsoperator / Ungleichheitsoperator
Kann wie bei elementaren Datentypen auch für Referenztypen angewandt werden. Zeigen beide Referenzen auf dasselbe Objekt, so wird "true" ausgegeben.

Kapitel 7 - Kontrollstrukturen

Block
Zusammengesetzte Anweisungen

Selektion
If-Else. Der else-Zweig ist optional. Fällt der else-Zweig weg, so spricht man von einer bedingten Anweisung. If-else-Anweisungen können auch geschachtelt werden.

Mehrfache Alternative - else if

```
if (Ausdruck 1)
        Anweisung_1
else if (Ausdruck 2)
        Anweisung_2
….
….
else
        Anweisung_else      // ist optional
```

Mehrfache Alternative – switch

```
switch (Ausdruck)
{
        case konstanter_Ausdruck_1:
                Anweisungen_1
                break;                  // Optional
        case konstanter_Ausdruck_2:
                Anweisungen_2
                break;                  // Optional
        default:
                Anweisung_default
}
```

Vor einer Befehlsfolge können auch mehrere “case“ Anweisungen stehen. Mit “break“ springt das Programm ans Ende der switch-Anweisung. Fehlt die break-Anweisung, wo werden die nach der nächsten case-Marke folgenden Anweisungen abgearbeitet.

Iteration

Abweisende Schleife mit while

```
while (Ausdruck)
{
        Anweisung
}
```

Abweisende Schleife mit for

```
for (int i=0; i<5; i++)
{
        Anweisung
}
```

Annehmende Schleife mit do-while

```
do
{
        Anweisung
}
while (Ausdruck);
```

Endlos-Schleife

```
for ( ; ;)
{
        ….
}

while (true)
{
        ……
}
```

break

Das Programm wird ausserhalb des momentanen Blockes fortgesetzt.

continue

Es wird ein neuer Schleifendurchgang gestartet.

Kapitel 8 - Blöcke und Methoden

Blöcke

Ein Block ist eine Folge von Anweisungen

Die leere Anweisung
Besteht nur aus einem Strichpunkt, welcher übersichtshalber auf eine seperate Linie genommen werden sollte.

Variablen im Konstruktor
Beeinflussen die Objektvariablen nicht. Wenn Sie das tun sollen, so müssen sie mit der this-Referenz definiert werden.

Lokale Variablen
Sind nur innerhalb des Blockes sichtbar. Für den umgebenden Block sind sie unsichtbar.

Lebensdauer
Während der Lebensdauer besitzt die Variable einen Speicherplatz

Gültigkeit
Bedeutet, dass eine Variable dem Compiler an einer bestimmten Programmstelle durch Vereinbarung bekannt ist.

Sichtbarkeit
Dass man von einer Programmstelle aus die Variable sieht, d.h. dass man auf ihren Namen zugreifen kann.

this-Methode
Mit der “this“-Methode kann auf eine verdeckte Variable zugegriffen werden. “this.x“ bewirkt z.B. dass nicht auf eine lokale Variable “x“ zugegriffen wird, sondern z.B. auf die Instanzvariable “x“, welche ausserhalb des Blockes definiert wurde.

Methoden

Anweisungsfolge, die unter einem Namen aufgerufen werden kann

Instanzmethoden
Für Objekte

Klassenmethoden
Für Klassen

Schlüsselwort “void“
Es ist in der Methode kein return-Wert notwendig

Parameterlose Methoden
Definiert mit einem Paar runden Klammern hinter dem Methodennamen.

Methoden mit Parametern
Definiert mit einem Paar runden Klammern mit den jeweiligen Übergabeparamtern. Heisst ein Übergabeparamter gleich, wie eine Objektvariable, so wird die Objektvariable überdeckt und kann nur noch mit "this." sichtbar gemacht werden.

Signatur
Name, Paramter (Anzahl, Typ, Reihenfolge) = Signatur

Einlesen eines Textfiles

```
public String einlesen() throws java.io.IOException{
  StringBuffer eingabe = new StringBuffer("");
  String str="", line="";
  int zeilen=0;
  try{
    BufferedReader br = new BufferedReader(new FileReader("Eingabe.txt"));
    do{
      line=br.readLine();
      zeilen++;
      eingabe.append(line);
    } while (line!=null);
    str=new String(eingabe);
    return str;
  } catch(IOException ioe) {
    ioe.printStackTrace();
    System.out.println("Anzahl Zeilen vor Abbruch: "+zeilen);
    return "";
```

Memoryabfragen
Runtime.getRuntime().freeMemory()
Runtime.getRuntime().totalMemory()

Objekt freigeben
objectRef = null;
Setzt aber voraus, dass auch ansonsten keine Referenz mehr auf das Objekt zeigt!

GarbageCollector aufrufen (muss aber nicht sofort passieren)
Runtime.getRuntime().gc();

Methoden mit Parametern
Eine Methode mit Parametern erkennt man am **formalen Parameter** innerhalb der runden Klammern. Ein Aufruf muss dann mit **aktuellen Parametern** erfolgen.
Der formale Parameter legt fest, **wie viele** Übergabeparamter existieren, von welchem **Typ** diese sind und welche **Reihenfolge** sie haben.
Als Paramter können auch Referenzen definiert werden.

Call by value
Wenn einem formalen Parameter ein entsprechender aktueller Parameter zugewiesen wird.

Call by reference
Es kann auch eine Referenz auf ein Objekt übergeben werden und dieses dann wie gehabt in der Methode verwendet werden.

Überladen von Methoden
Erfolgt durch die Definition verschiedener Methoden mit gleichem Methodennamen, aber verschiedenen Parameterlisten.
Es ist jedoch nicht möglich überladene Methoden mit dem selben Namen, aber mit verschiedenen Rückgabetypen zu vereinbaren.

Polymorphie von Operationen
Jedes Objekt trägt die Typinformation, von welcher Klasses es ist, immer bei sich. Deshalb ist es möglich in verschiedenen Klassen gleichnamige Methoden zu vereinbaren.

Übergabe von Parametern beim Programmaufruf
Übergebene Parameter werden im String-Array “args“ gespeichert.
Man beachte, dass der erste Parameter nach args[0] gespeichert wird.

Rekursion
Ein Algorithmus heisst rekursiv, wenn er Abschnitte enthält, die sich selbst direkt oder indirekt aufrufen. Indirekte Rekursion liegt vor, wenn sich zwei Methoden wechselseitig aufrufen.

Namen einer Klasse ausgeben
```
Object refObj = new Object();
Class refClass = refObj.getClass();
// Den Namen der Klasse ausgeben
System.out.println ("Name: " + refClass.getName());
```

Kapitel 9 - Klassen und Objekte

Eine Klasse besteht aus dem Namen der Klasse
Objekt und Klassenbezogenen Datenfeldern
Objekt und Klassenbezogenen Methoden

Information Hiding
Es sollte keinen direkten Zugriff auf die Daten eines Objektes aus anderen Klassen geben.
Dies wird durch das Schlüsselwort “private“ realisiert.

Klassenvariablen / Klassenmethoden
Werden mit dem Schlüsselwort “static“ definiert.
Hauptsächlich für die Speicherung von globalen Daten zu verwenden.
Können direkt über den Klassennamen aufgerufen werden.
Eine Klassenmethode kann auch auf Instanzvariablen arbeiten, wenn ihr explizit eine Referenz auf das entsprechende Objekt übergeben wird.
Dies sollte jedoch vermieden werden.

Die this-Referenz

- Wenn der Programmierer explizit darauf aufmerksam machen möchte, dass er auf eine Instanzvariable, bzw. eine Instanzmethode des eigenen Objekts zugreifen möchte
- Wenn ein Datenfeld denselben Namen trägt, wie eine lokale Variable
- Wenn eine Referenz auf das eigene Objekt als Rückgabewert zurückgegeben werden soll
- Wenn eine Referenz auf das aktuelle Objekt als Parameter an eine Methode übergeben werden soll

Klassenmethoden besitzen keine this-Referenz!

Initialisierung von Datenfeldern
Klassen und Instanzvariablen werden per Default initialisiert. Klassenvariablen beim Laden der Klasse, Instanzvariablen beim Anlegen eines Objektes.
Lokale Variablen werden nicht per Default initialisiert!

Initialisierung mit Hilfe eines Initialisierungsblockes
Ein Initialisierungsblock wird mit “static“ gefolgt von einem Block definiert. Dieser Block im Rahmen der Initialisierung von Klassenvariablen ausgeführt. Dasselbe gibt es natürlich auch noch für Instanzvariablen. Dort kann einfach das Schlüsselwort “static“ weggelassen werden. Dieser Block, wird jedes Mal ausgeführt, wenn ein Objekt dieser Klasse angelegt wurde. Jedoch jeweils nur einmal!

Konstruktoren zur Initialisierung
Instanzvariablen werden am besten im Konstruktor initalisiert
Anzahl der erzeugten Objekte geschieht auch am besten aus dem Konstruktor, da dieser bei jeder Schaffung eines Objektes durchlaufen wird.
Wird nicht an eine abgeleitete Klasse weitervererbt

Ein Konstruktor kann in seiner ersten Anweisung einen anderen Konstruktor derselben Klasse aufrufen. Dies geschieht mit “this (Parameter)“
So können z.B. Default-Werte übergeben werden.

Verhindern der Instantiierung einer Klasse
Mit Hilfe des Singletons-Prinzipes kann man die Anzahl der lebenden Objekte einer bestimmten Klasse regulieren.

```
class Singleton
{
	private static Singleton instance;
	private Singleton()
	{
		System.out.println(„Bin im Konstruktor“);
	}
	static Singleton getSingleton()
	{
		if (instance == null)
		{
			instance = new Singleton();
		}
	}
}

class Test
{
	public static void main (String[] args)
	{
		Singleton s2 = Singleton.getSingleton(); // funktioniert
		Singleton s3 = Singleton.getSingleton();// funktioniert nicht mehr
	}
```

Freigabe von Speicher
In Java wird Speicherplatz nicht automatisch freigegeben, wenn man dies wünscht. Der GarbageCollector entscheidet selbst, wann ein nicht mehr benötigtes Objekt vom Heap gelöscht wird.
Ein Objekt wird freigegeben, indem die entsprechenden Referenzen auf “null“ gesetzt werden.
Der GarbageCollector kann über “System.gc“ aufgerufen werden. Es ist jedoch nicht festgelegt, wann die Speicherbereinigung dann ausgeführt wird.
finalize() wird vom Compiler aufgerufen, bevor das Objekt gelöscht wird. Diese Methode kann auch von Hand aufgerufen werden.

Die Klasse Object
Jede Klasse und jedes Array wird implizit von der Klasse Object abgeleitet und erbt somit deren Methoden. So z.B. toString(), equals(Object obj), clone(), etc.

Die Klasse Class
Jeder Typ in Java wird in der Virtuellen Maschine durch ein Objekt der Klasse “Class“ repräsentiert.

Kapitel 10 - Vererbung und Polymorphie

Vererbung
Alle Datenfelder und Methoden werden von einer Superklasse übernommen. Die Superklasse merkt davon jedoch nichts, weshalb auch der Pfeil im Diagramm von Sohn zum Vater zeigt und nicht umgekehrt.
Sohnklassen können weitere Methoden oder Datenfelder enthalten.
Vererbt wird mit dem Schlüsselwort **"extends".**

Spezialisierung / Generalisierung
Eine abgeleitete Klasse stellt eine Spezialisierung einer Basisklasse dar.
Eine Basisklasse stellt die Generalisierung Ihrer abgeleiteten Klasse dar.
Klassen, welche allgemeine Datenfelder enthalten, sollten möglichst oben in der Hierarchie gehalten werden, um Wiederholungen zu vermeiden.

Polymorphie von Operationen
Ein und der selbe Methodenaufruf kann in verschiedenen Klassen verschiedene Anweisungsfolgen besitzen.

Polymorphie von Objekten
Ein Sohnobjekt kann auch an Stelle eines Vaterobjekts eintreten, da es auch vom Typ Vater ist. (Liskov Substitution Principle)
Es findet jeweils ein Cast statt, bei welchem die spezifischen Sohnmethoden und Datenfelder verdeckt werden.

Konstruktoren bei abgeleiteten Klassen
Wird ein Konstruktor in einer abgeleiteten Klasse aufgerufen, so wird automatisch zuerst der Konstruktor in der Basisklasse durchlaufen!
Dies führt dazu dass automatisch ein Defaultkonstruktor für die Basisklasse geschrieben werden muss, wenn dort ein Konstruktor mit formalen Paramtern definiert wurde.
In einem Konstruktor kann entweder ein anderer überladener Konstruktor aufgerufen werden oder ein Konstruktor der Vaterklasse

Typkonvertierung von Referenzen
Implizite Typenkonvertierung wird dann durchgeführt, wenn eine Referenz auf ein Sohnobjekt einem Vaterobjekt zugewiesen wurde. Dadurch wird jedoch die Sichtweise eingeschränkt und die Referenz vom Typ Vater kann nur noch die Vateranteile des Sohnobjektes erkennen. Dies wird auch **"Up-Cast"** genannt.
Explizite Typenkonvertierung ist dann nötig, wenn eine Umwandlung einer Referenz auf einen Vater zu einer Referenz auf einen Sohn erfolgen soll.
Dies nennt man auch **"Down-Cast"** und funktioniert nur dann, wenn die Referenz des Vaters auf einen Sohn zeigt. Beim Down-Cast wird das Protokoll des Typs auf welchen gecastet wird sichtbar.

Verdecken und Überschreiben
Verwendet eine Sohnklasse genau dieselben Methoden und Datenfelder wie die Vaterklasse, so werden diese überdeckt. -> **Vater wird überschrieben**

Dies geschieht auch bei gleichen Variablennamen von unterschiedlichen Variablentypen. Auf die verdeckten Daten kann mithilfe des Schlüsselwortes „“super“ oder über einen **Cast der this-Referenz** in das Vaterobjekt
super.x oder **((Vater) this).x**
Wird ein Datenfeld x mithilfe von super gesucht, so wird ausgehend von der aktuellen Klasse die gesamte Klassenhierarchie aufwärts der Reihe nach solange durchsucht, bis zum ersten Mail ein Datenfeld x gefunden wird. An dieser Stelle wird die Suche abgebrochen.

Überschreiben von Methoden
Eine Methode der Sohnklasse überschreibt eine Methode der Vaterklasse. Auf die Methode der Vaterklasse kann jedoch weiterhin mit dem Schlüsselwort “super“ zugegriffen werden. Der Code wird so “robuster“.
Wenn die Methode der Vaterklasse “final“ ist, so kann sie in der Sohnklasse nicht überschrieben werden.

Finale Klassen
Können nur benutzt, jedoch nicht abgeleitet werden.
Ist die Klasse als “final“ deklariert, so sind alle Methoden und Datenfelder automatische ebenfalls “final“.

Konstantenklassen
Konstruktor wird auf “private“ gesetzt, womit sich die Klasse nicht mehr instantiieren lässt. Es werden danach “static“-Konstanten definiert, welche dann von anderen Klassen verwendet werden können.

Statische und dynamische Bindung von Methoden
“final“, „“static, und “private“-Methoden werden statisch, d.h. beim Compilieren gebunden.
Bei allen anderen Methoden kann der Compiler nicht wissen, in welcher Klasse er eine Methode aufrufen muss. Deshalb geschieht dort die Bindung meisten spät, d.h. bei der Laufzeit.

Array von Referenzen
Ein Array von Referenzen wird gebildet, indem als Arraytyp jeweils der Klassenname angegeben wird. Z.B. person[ ] perArr = new person[3]
Die Felder müssen dann jeweils noch mit Referenzen gefüllt werden.

Der “instanceof“-Operator
Mit dem “instanceof“-Operator kann getestet werden, ob eine Referenz auf ein Objekt eines bestimmten Typs zeigt.
“**refA instanceof Grossvater**“ gibt **true** zurück, wenn refA auf Grossvater, oder ein Subobjekt zeigt. Wenn dies nicht der Fall ist, wird “false“ ausgegeben

Abstrakte Basisklassen
Wird in einer Basisklasse nur die Schnittstelle von Methoden festgelegt, so liegt eine abstrakte Basisklasse vor. Diese werden mit dem Schlüsselwort “**abstract**“ deklariert. Ist auch nur eine einzige Methode in einer Klasse“abstract“, so ist die Klasse zwangsläufig “abstract“ und kann nicht instantiiert werden.

Kapitel 11 - Pakete

Ein Paket stellt eine Klassenbibliothek dar, die einen Namen trägt.

Vorteile

- Pakete bilden eigene Bereiche für den Zugriffsschutz.
- Pakete bilden einen eigenen Namensraum, damit können Namenskonflikte vermieden werden.
- Grösste Einheiten für die Strukturierung von Programmen

Erstellung von Paketen
Alle Dateien des Paketes werden mit der Deklaration des Paketnamens versehen. Dies erfolgt mit Hilfe des Schlüsselwortes “package“.
Die Deklaration muss immer als erstes erfolgen.
Paketnamen werden klein geschrieben.

Pakete können wiederum Unterpakete enthalten. Diese werden folgendermassen angesprochen: “package paket.unterpaket;

Benutzung von Paketen
Methoden werden über den Punktoperator angesprochen.
Der jeweilige Paketname geht dem Methodennamen voraus.

Die import-Vereinbarung
Die import-Vereinbarung macht es möglich, dass auf eine Klasse, oder eine Schnittstelle in einem anderen Paket, direkt über ihren Namen zugegriffen werden kann, ohne den Namen der Paketstruktur angeben zu müssen.
Die import-Vereinbarung erfolgt hinter der package-Deklaration, jedoch vor dem Rest des Programmes. Beispiel: “import paket.*;“
Fallen bei einer import-Vereinbarung zwei Namen zusammen, so muss immer der qualifizierte Name verwendet werden.

Paketnamen
In der Regel wird die Paketstruktur eins zu eins in die Verzeichnisstruktur übernommen.

CLASSPATH
Ist eine Umgebungsvariable, die dem Compiler und dem Interpreter sagt, wo sie nach den Daten suchen müssen.
Dies ist nötig, wenn Pakete aus unterschiedlichen Ebenen aufgerufen werden.
Wenn nur Pakete unterhalb des aktuellen Verzeichnisses aufgerufen werden, so ist der Classpath nicht zwingend nötig.
Es ist auch möglich mehrere CLASSPATH’s anzugeben.

Aufruf
Der Aufruf des Interpreters geschieht mit: java paket1.Klasse1
Der Aufruf des Compilers geschieht mit: javac paket1\Klasse1.java

Gültigkeitsbereich von Klassennamen
Wird eine Klasse nicht mit einem Zugriffsmodifikator versehen, so ist deren Gültigkeit über die ganze Klasse erstreckt. Nicht aber in Unterklassen.

Zugriffsmodifikatoren

public
Alle haben vollen Zugriff

default (friendly)
Public innerhalb desselben Paketes

protected
Alle aus demselben Paket und alle Subklassen in anderen Paketen.

private
Nur die eigene Klasse

Zugriffsschutz für Konstruktoren
Je nachdem welcher Zugriffsmodifikator auf den Konstruktor angewendet wurde, werden Einschränkungen auf die Bildung von Objekten der Klasse gesetzt.
Für das Singleton-Pattern wird z.B. ein private-Konstruktor eingesetzt.
Wird kein eigener Konstruktor geschrieben, so nimmt der default-Konstruktor den Zugriffsschutz der Klasse an.

Zugriffsschutz beim Überschreiben von Methoden
Man darf die Zugriffsmodifikatoren einer überschriebenen Methode nicht einschränken, sondern nur erweitern.

Kapitel 12 – Ausnahmebehandlung / Exception Handling

Eine Exception kann z.B. ein arithmetischer Überlauf, ein Mangel an Speicherplatz, eine Verletzung der Array-Grenzen, die Verletzung einer Zusicherung, etc. darstellen. Dies sind immer **Laufzeit-Ereignisse**.
Mit einem Exception-Handler kann der Programmierer auftretende **Fehler auffangen** und somit das Programm fehlerfrei weiterarbeiten lassen.

Ziel des Exception-Handlings
Ziel des Exception Handlings ist es, normalen und fehlerbehandelnden Code übersichtlich **zu trennen** und Ausnahmesituationen sicher zu behandeln.

Checked Exception
Dies ist eine zu berücksichtigende Exception.
Hier erzwingt Java die Fehlerbehandlung.

Exceptions in Bibliotheken
Exceptions ermöglichen es einer Bibliothek, Ausnahmezustände in einfacher Weise an das aufrufende Programm zu melden.

Implementierung von Exception-Handlern in Java
Das Exception-Handling wird in Java durch eine **try-Anweisung** realisiert.
Eine try-Anweisung muss einen try-Block und kann ein oder mehrere catch-Konstrukte und ein finally-Konstrukt enthalten. Ist mindestens ein catch-Konstrukt da, so kann das finally-Konstrukt entfallen. Ist kein catch-Konstukt vorhanden, so ist das finally-Konstrukt erforderlich.
Die try-Anweisung enthält einen Block mit normalen Anweisungen, bei denen jeweils Exceptions auftreten können.
Eventuel auftretende Exceptions können mit der catch-Anweisung gefangen werden. Der finally-Block dient zum Aufräumen.
Tritt im try-Block keine Exception auf, so werden die Handler übersprungen.

Ablauf bei einer Exception
Tritt ein Programmfehler auf, wird eine Instanz der entsprechenden Exception-Klasse mit throw geworfen und der gerade ausgeführte Block verlassen.
Nach der Abarbeitung des entsprechenden Catch-Blockes, wird das Programm nach den Handlern fortgesetzt.
Code zwischen den Handlern ist dabei nicht erlaubt!

Propaganda
Exceptions, die nicht behandelt wurden, werden an den jeweiligen Aufrufer der Methode weitergereicht. Man sagt, Exceptions werden **propagiert**.
Werden Exceptions weitergegeben, so muss dies in der Schnittstellenbeschreibung der Methode durch das Schlüsselwort “**throws**“ angegeben werden, ansonsten resultiert ein Compilerfehler.

Ausnahmen vereinbaren und auswerfen
Ausnahmen können mit Hilfe der Anweisung "**throw**" an beliebiger Stelle in einem try-Block ausgeworfen werden. „"throw" akzeptiert jede Referenz auf, die auf ein Objekt vom Typ "throwable" zeigt.
Mit der "throw"-Anweisung wird der try-Block verlassen, egal was danach noch folgt!

Die Exception-Hierarchie

Die Klasse "Error"
Diese Errors müssen abgesehen von ein paar wenigen Ausnahmen nicht vom Programmierer behandelt werden.

Die Klasse "Exception"
Diese Fehler können vom Programmierer zur Laufzeit behandelt werden.

Checked und Unchecked Exceptions

Man spricht von **Unchecked Exceptions** falls eine Exception vom Programmierer nicht abgefangen werden muss (können aber abgefangen werden).
Dazu gehören alle Errors und RunTimeExceptions.

Man spricht von **Checked Exceptions** falls eine Exception vom Programmierer behandelt werden muss, und dies auch vom Compiler überprüft (checked) wird.

Reihenfolge der Handler
Die Suche nach dem passenden Handler erfolgt von oben nach unten.
Eine Exception, die im Klassenbaum der Exceptions am weitesten oben steht, sollte an letzter Stelle stehen.
Zuerst folgen also die Handler für die meist spezialisierten Klassen, danach in zunehmender Generalisierung die allgemeineren Handler.

Exceptions weiterreichen
Exceptions können innerhalb des catch-Blockes erneut geworfen werden.
Innerhalb von Handlern geworfene Exceptions werden nach aussen an die nächste umschliessende try-Anweisung weitergereicht.

Kapitel 13 - Schnittstellen

Schnittstellen oder auch Interfaces genannt, bieten in Java die Möglichkeit, die spezifizierende Schicht (Entwurf) zu unterstützen.

Das Schlüsselwort “interface“
Interfaces werden mit dem Schlüsselwort “interface“ definiert.
In einem Interface können beliebige Methoden beschrieben, oder auch Konstanten definiert werden.

Das Schlüsselwort “implements“
Jede Klasse, die eine Schnittstelle implementieren möchte, muss dies bei der Klassendefinition mit dem Schlüsselwort “implements“ gefolgt vom Interfacenamen deklarieren. Sobald ein Interface implementiert wurde, müssen alle Methoden der Schnittstelle implementiert werden, ansonsten wird die Klasse automatisch “abstract“.

Parameterübergabe
Wird als formaler Übergabeparameter ein Schnittstellentyp angegeben, so kann als aktueller Parameter auch eine Referenz auf ein Objekt übergeben werden, welches das Interface implementiert.

Aufbau einer Schnittstelle
Jede Schnittstelle beginnt mit der Schnittstellendeklaration, gefolgt vom Schnittstellenkörper.
Wenn eine Schnittstelle von einer anderen abgeleitet werden soll, so wird auch hier mit dem Schlüsselwort “extends“ gearbeitet.
Der Schnittstellenkörper enthält Konstantendefinitionen und Methodendeklarationen. Alle in der Schnittstelle aufgeführten Methoden sind automatisch “public“ und “abstract“. Alle aufgeführten Datenfelder sind immer implizit “public static final“. Die Datenfelder müssen alle initialisiert werden. Der Initialisierungsausdruck muss dabei nicht konstant sein.

Verwenden von Schnittstellen
Es ist auch möglich Arrays von Schnittstellentypen anzulegen und diese Arrays mit Objekten zu füllen, deren Klasse die Schnittstelle implementieren.

Typsicherheit von Schnittstellen
Schnittstellen sind ein elegantes Mittel zur Überprüfung, ob der Anwender den richtigen Typ übergeben hat. Deshalb sollte immer ein möglichst spezifischer, formaler Übergabetyp für Methoden angegeben werden.

Implementieren von mehreren Schnittstellen
Eine Klasse kann beliebig viele Schnittstellen implementieren.
Dabei werden die Schnittstellennamen jeweils getrennt durch ein Komma angegeben.

Dabei können natürlich Probleme entstehen, wie z.B. wenn exakt dieselbe Methode von zwei Schnittstellen verlangt wird, oder Konstanten mit genau demselben Namen.

Vererbung von Schnittstellen
Schnittstellen können mit dem Schlüsselwort “extends“ vererbt werden.
Dabei ist es auch möglich mehrere Schnittstellen zu vererben.
Dies ist jedoch in Java nicht von allzugrosser Bedeutung.
Um Mehrfachvererbungen zu realisieren, müssen die Schnittstellennamen wiederum getrennt durch ein Komma aufgelistet werden.