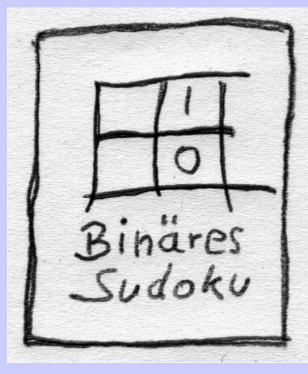
hochschule mannheim

Jörn Fischer
j.fischer@hs-mannheim.de

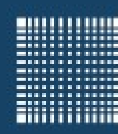
Willkommen zur Vorlesung

Programmieren 1



Vorstellung

Zu meiner Person...



Überblick

Inhalt (roter Faden)

1 - Organisatorisches
2 - Einführung
3 - Programmentwurf
4 - Erste Schritte
5 - Die Programmiersprache Java
6 - Variablen und Datentypen
7 - Arithmetische Ausdrücke
8 - Anweisungen (Schleifen, bedingte Verzweigungen)
9 - Arrays, Enumerationen
10 - Methoden
11 - Rekursion
12 - Klassen, Objekte
13 - Vererbung
14 - Referenzen
15 - Interfaces
16 - Exceptions

1 - Organisatorisches

Kapitel 1 - Organisatorisches

- Übungen
- Prüfungsvoraussetzung
- Fragen an mich

1 - Organisatorisches

Übungen

- Übungsstunden finden in 2'er oder 3'er Gruppen statt
- In den Übungen besteht Anwesenheitspflicht!
- Wenn ihr krank seid, reicht bitte eine Krankmeldung vom Arzt ein.
- Wer sich letztes Semester schon für die Klausur qualifiziert hat und mitschreiben möchte, der kann sich einfach für die Klausur anmelden.
- Es gibt alle zwei Wochen neue Übungsblätter
- Zusätzlich werde ich Life-Testate einführen

1 - Organisatorisches

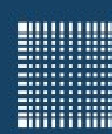
Prüfungsvoraussetzung

- Anwesenheit in den Übungsstunden
- Mindestens 80% der erreichbaren Gesamtpunktzahl der **Pflichtübungen**

1 - Organisatorisches

Fragen an mich...

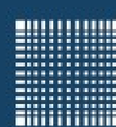
- Bester Zeitpunkt um Fragen los zu werden sind die Übungsstunden oder wenn es Fragen zum Vorlesungsstoff sind natürlich auch in der Vorlesung.
- Rückkopplung ist ausdrücklich erwünscht !!!!!
- Zwischen den Vorlesungen bin ich oft kurz angebunden und habe wenig Zeit
- Wenn ihr eine persönliche Frage habt, dann könnt ihr mir gerne eine E-Mail schreiben.



1 - Organisatorisches

Haben Sie Fragen?

2 - Einführung

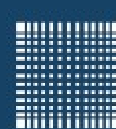


Kapitel 2 - Einführung

- Welches Buch kann ich lesen?
- Die Programmiersprache Java.
- Wie funktioniert ein Computer?
- Welche Zahlensysteme werden benutzt?

Welches Buch kann ich lesen

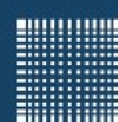
- Es gibt unglaublich viele Bücher über Java (auch online)
- Sinnvoll für dieses Semester ist es, wenn Sie sich erst einmal ein Buch aus der Bibliothek über Java-Programmierung ausleihen.



2 - Einführung

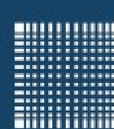
Welches Buch kann ich lesen

- Einführung in die Informatrik, Heinz-Peter Gumm, Manfred Sommer
- Java ist auch eine Insel, Galileo Computing, 1480 Seiten
- Grundkurs Programmieren in JAVA, Carl Hanser Verlag GmbH, 711 Seiten
- Java von Kopf bis Fuß, O' Reilly, 720 Seiten
- **Mein Favorit:** JAVA Eine Einführung, Martin Schrader, Lars Schmidt-Thiemen, Springer, 635 Seiten (leider nicht mehr aufgelegt, also nur in der Bibliothek)
- Taschenbuch Programmiersprachen, Hanser, 631 Seiten, 31 Seiten JAVA (Nur als Überblick)
- usw....



Die Programmiersprache Java

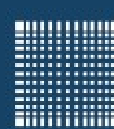
1950
Fortran
Lisp
1960
Cobol
Algol
PL/1
Basic
BCPL
Simula
1970
Pascal
Smalltalk
Prolog
C
ML
1980
Ada
Modula-2
Miranda
C++
Eiffel
1990
Oberon
Java
2000



2 - Einführung

Die Programmiersprache Java

- Entwicklung in Assembler ist sehr aufwendig (schlecht portier- und wartbar)
- Programmiersprache B, angelehnt an BCPL (Basic Combined Programming Language)
- Entwicklung von C, im Gegensatz zu B typisiert
- Java hat viele Teile der Syntax von C/ C++ übernommen, hat aber Altlasten wie die Aufteilung in Header und Source Files über Bord geworfen
- Java ist durch VM (Virtuelle Maschine) unabhängig von Hardware und Betriebssystem



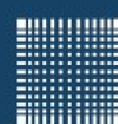
2 - Einführung

Die Programmiersprache Java - warum?

- einfach
- auf vielen Plattformen lauffähig
- schnell
- große Bibliothek und auf vielen Plattformen verwendbar
  - Ein-/ Ausgabe, Dateioperationen
  - GUI
  - Zeichenkettenverarbeitung
  - Mathematik
  - usw...

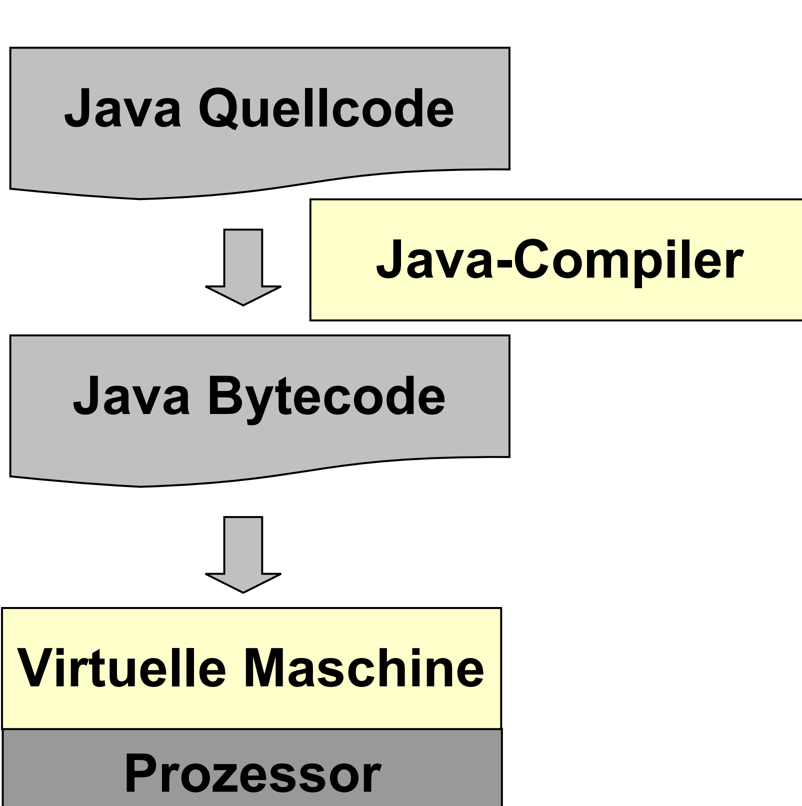
Die Programmiersprache Java (aber...)

- Java for Android hat komplett andere GUI Implementierung
- Java ist nicht ohne Jailbreak auf iPhone und iPad zu installieren
- Java kann nicht ohne große langsame Umwege auf spezielle Hardware zugreifen
- Es gibt nicht **das** Graphical User Interface (GUI) … z.B. Swing vs. AWT …
- Compilierte Java Klassen lassen sich relativ leicht wieder in Quellcode umwandeln (schränkt die Verwendung für kommerzielle Produkte ein)



2 - Einführung

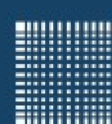
Was passiert mit einem selbst geschriebenen Programm?



- Java-Programme liegen als Quellcode vor
- Ein Programm kann aus mehreren Dateien bestehen, in denen i.Allg. jeweils eine Klassen implementiert ist.
- Eine dieser Klassen implementiert die Methode public static void main(String [] arg) , die beim Starten automatisch ausgeführt wird.
- Wir werden zunächst nur mit Programmen arbeiten, die aus einer einzelnen Datei bestehen.

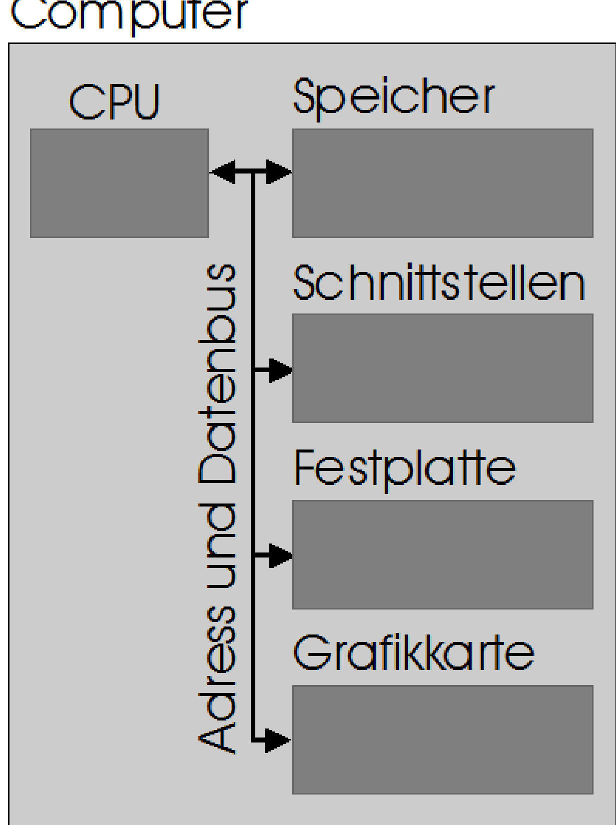
Wie funktioniert ein Computer

- Ein Computer folgt dem EVA Prinzip: Eingabe, Verarbeitung, Ausgabe
- Im Computer gibt es u.a.
  - CPU (Central Processing Unit), die Recheneinheit
  - RAM (Random Access Memory)
  - ROM (Read only Memory) bzw. Flash-Memory für das BIOS (Basic Input Output System)
  - Festplattenspeicher oder SSD (Solid State Disk)
  - Grafikkarte
  - Interfaces (Schnittstellen)
  - Bus Systeme (z.B. USB = Universal Serial Bus, PCI-Express) meistens bestehend aus Daten- und Adressbus



2 - Einführung

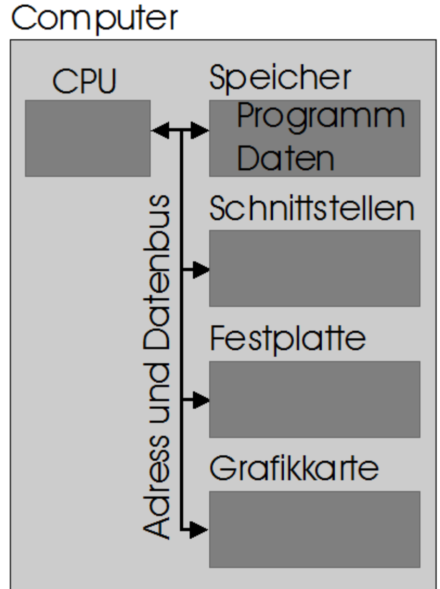
Wie funktioniert ein Computer



Wie funktioniert ein Computer

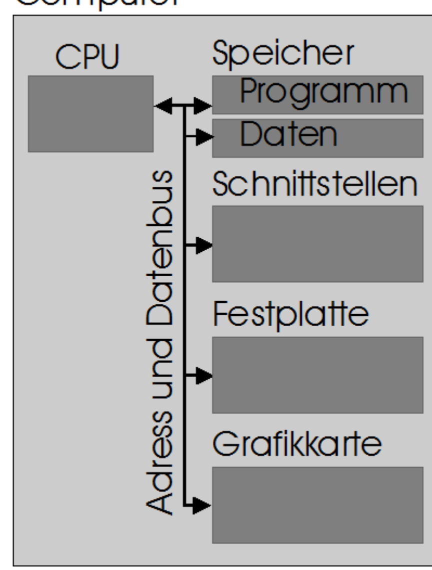
- Es gibt unzählige Computerarchitekturen. Wichtig für die Programmierung ist beispielsweise die Unterteilung in

Von Neumann Architektur

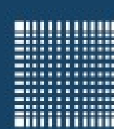


Moderner PC

Harvard Architektur



manche Digitalen Signal Prozessoren



2 - Einführung

Zahlensysteme und Umwandlung

- Zahlensysteme sind Stellenwert-Systeme.
- Jeder Stelle ist ein Vervielfachungsfaktor in Form einer Potenzzahl zugeordnet.
- Beim Dezimalsystem ist jede Stelle einer Zehnerpotenz zugeordnet (10 mögliche Ziffern).
- Beim Binärsystem ist jede Stelle einer Zweierpotenz zugeordnet (2 mögliche Ziffern pro Stelle).
- Beim Hexadezimalsystem ist jede Stelle einer Sechzehnerpotenz zugeordnet (16 mögliche Ziffern).

Zahlensysteme und Umwandlung

Wandlung von Dezimal in Hexadezimal:

```
4025 / 16 = 251 Rest 9  -> 9 (letzte Stelle ganz rechts)
251  / 16 = 15  Rest 11 -> B
15   / 16 = 0   Rest 15 -> F (erste Stelle ganz links)
```

Dez	16^5	16^4	16^3	16^2	16^1	16^0
	1048576	65536	4096	256	16	1
4025	0	0	0	F	B	9

Zahlensysteme und Umwandlung

Wandlung von Dezimal in Binär:

```
185 / 2 = 92 Rest 1 (Least Significant Bit -> LSB)
92  / 2 = 46 Rest 0
46  / 2 = 23 Rest 0
23  / 2 = 11 Rest 1
11  / 2 = 5  Rest 1
5   / 2 = 2  Rest 1
2   / 2 = 1  Rest 0
1   / 2 = 0  Rest 1 (Most Significant Bit -> MSB)
```

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1
Binär	1	0	1	1	1	0	0	1

Zahlensysteme und Umwandlung

Wandlung von Binär in Dezimal:

```
185 = 128 + 32 + 16 + 8 + 1
```

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1
Binär	1	0	1	1	1	0	0	1

- Summiert man die Zahlen, über den Einsen einer Binärzahl, so erhält man die Dezimalzahl (s.o.)

Alternativ kann auch verschachtelt vorgegangen werden:

185 = ((((((((1)*2+0)*2+1)*2+1)*2+1)*2+0)*2+0)*2+1)

Zahlensysteme und Umwandlung

Wandlung von Binär in Hexadezimal:

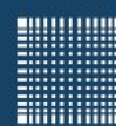
$1011_{bin} = 11_{dez} = B_{hex}$

$1001_{bin} = 9_{dez} = 9_{hex}$

$1011\ 1001_{bin} = B9_{hex}$

	2^3	2^2	2^1	2^0	2^3	2^2	2^1	2^0
	8	4	2	1	8	4	2	1
Binär	1	0	1	1	1	0	0	1

- Die Binärzahl wird von rechts nach links in 4‘er Gruppen eingeteilt und jede Hex. Ziffer einzeln umgewandelt.



2 - Einführung

Zahlensysteme und Umwandlung

Wandlung von Hexadezimal in Binär:

$B9_{hex}$ enthält Ziffern „B“ und „9“

$B_{hex} = 11_{dez} = 1011_{bin}$

$9_{hex} = 9_{dez} = 1001_{bin}$

$B9_{hex} = 1011\ 1001_{bin}$

	2^3	2^2	2^1	2^0	2^3	2^2	2^1	2^0
	8	4	2	1	8	4	2	1
Binär	1	0	1	1	1	0	0	1

- Jede Hex. Ziffer wird einzeln in Binär umgewandelt!

Zahlensysteme und Umwandlung

Wandlung von Hexadezimal in Dezimal:

$FB9_{hex}$ enthält Ziffern „F", „B" und „9"

$F_{HEX} = 15_{dez}$

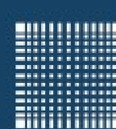
$B_{hex} = 11_{dez}$

$9_{hex} = 9_{dez}$

Dez	16^5	16^4	16^3	16^2	16^1	16^0
	1048576	65536	4096	256	16	1
4025	0	0	0	F	B	9

$FB9_{hex} = 15 * 256 + 16 * 11 + 9 * 1 = 4025_{dez}$

3 - Programmentwurf



Kapitel 3 - Programmentwurf

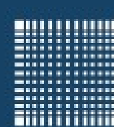
- Algorithmenbegriff
- Aktivitätsdiagramm
- Pseudocode

Algorithmenbegriff:

- Ein **Algorithmus** ist ein Verfahren mit einer
  - präzisen (d.h. in einer genau festgelegten Sprache formulierten)
  - endlichen Beschreibung unter Verwendung
    - effektiver (d.h. tatsächlich ausführbarer)
    - elementarer Verarbeitungsschritte
- Ein Algorithmus benötigt nur endlich viele Ressourcen:
  - Rechenzeit
  - Speicher

Ein Algorithmus ist unabhängig von der Programmiersprache.

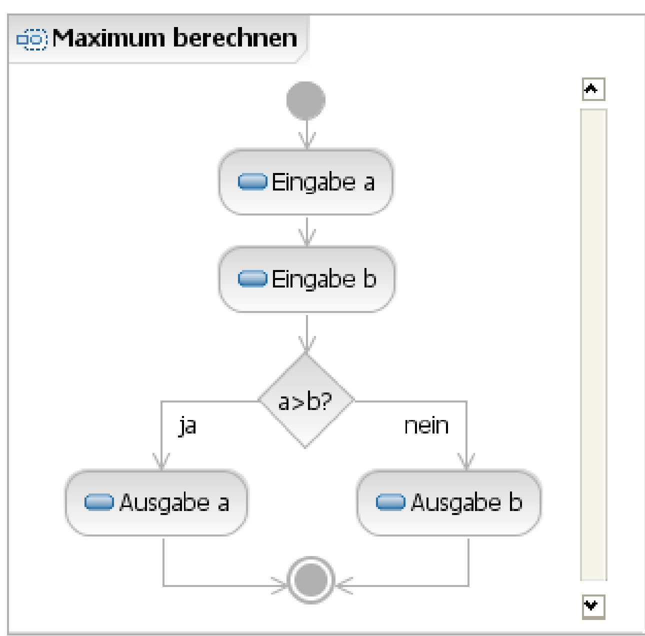
Aktivitätsdiagramm

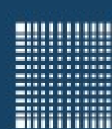


3 - Programmentwurf

Aktivitätsdiagramm

- Standardisierte Notation für Abläufe
  - Bestandteil der UML (Unified Modeling Language)
  - Basiert auf Programmablaufplan / Flussdiagramm
- Beispiel:



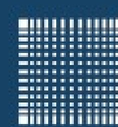


3 - Programmentwurf

Aktivitätsdiagramm

- Grafische Darstellung als geschlossener Block
  - Muss mit Startknoten ● beginnen
  - Muss mit Endknoten ◉ enden

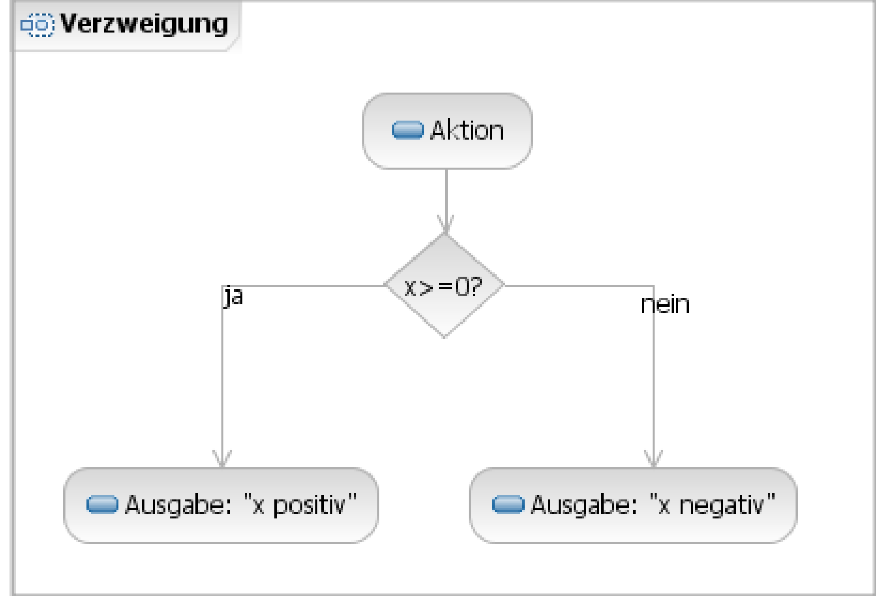
  - Aktionen: abgerundete Rechtecke
  - Fallunterscheidung: Raute mit Bedingung
  - Schleife: Fallunterscheidung mit Rücksprung
  - Programmfluss: Kanten ←

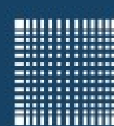


3 - Programmentwurf

Aktivitätsdiagramm

- Verzweigungen werden durch Rauten dargestellt
  - Ja-Ausgang wird genommen, wenn Bedingung zutrifft
  - Nein-Ausgang wird genommen, wenn Bedingung nicht zutrifft
  - Bei Fallunterscheidungen entsprechend weitere Ausgänge

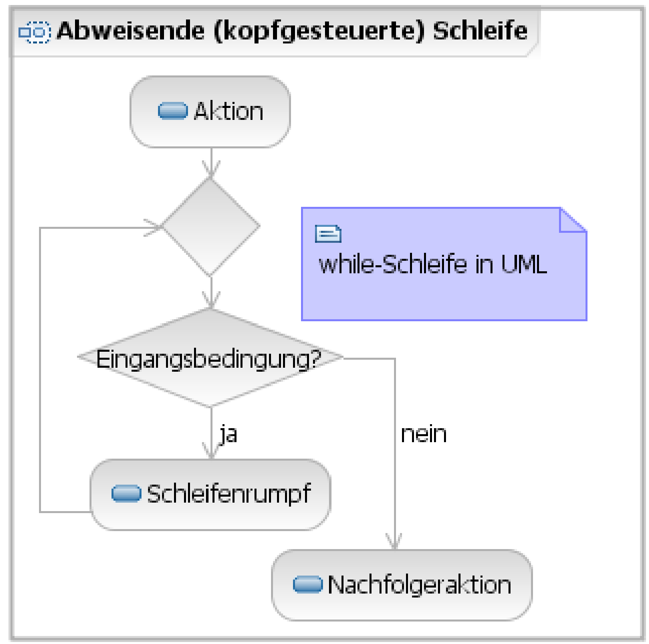


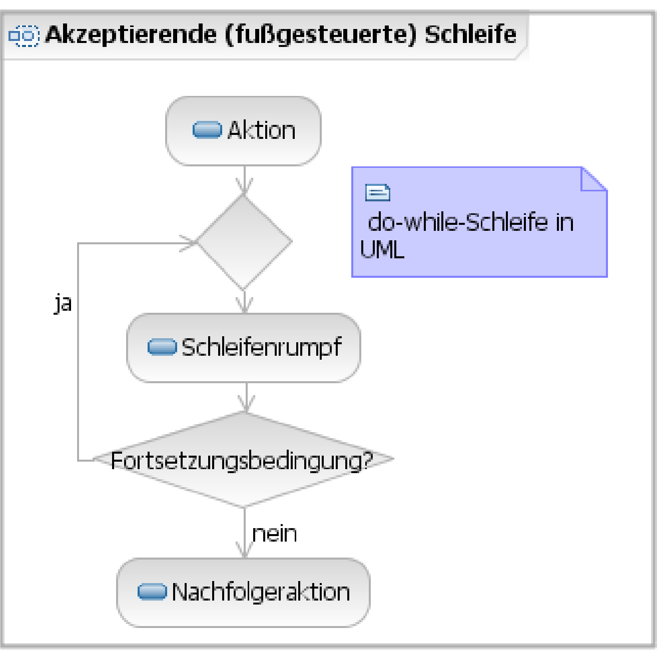


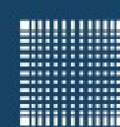
3 - Programmentwurf

Aktivitätsdiagramm

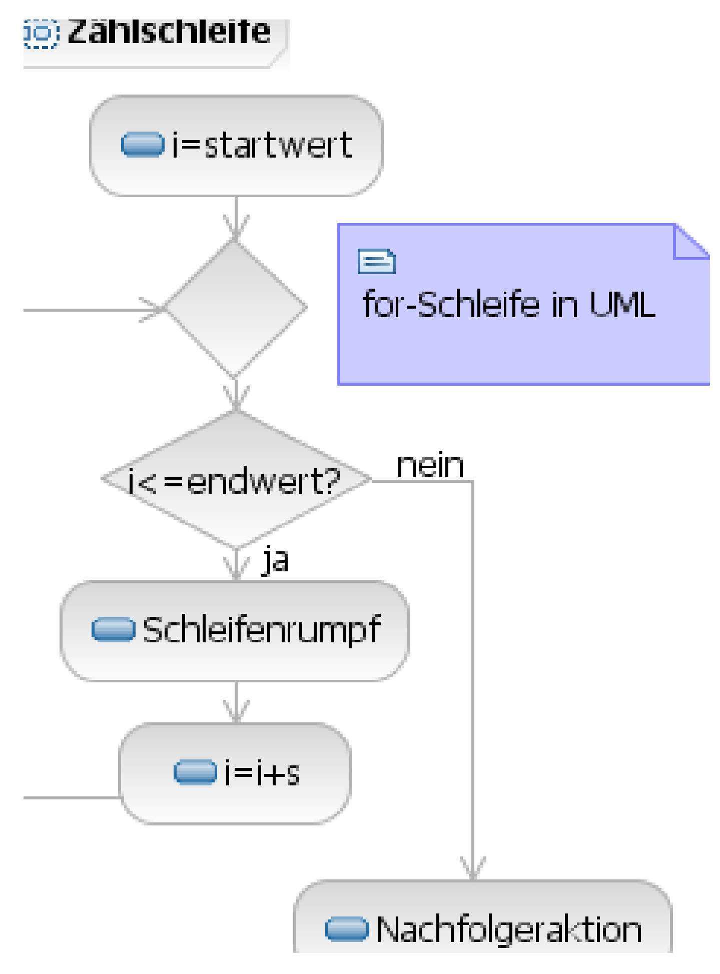
- Bedingungsabfrage durch Verzweigung
- Rücksprung an Schleifeneintritt durch Zusammenführung







Aktivitätsdiagramm



Zählschleife:

for *i* = *startwert* to *endwert* step *s*

Schleifenrumpf ausführen

Falls *s<= -1* : rückwärts zählen

Falls *s== 0* : nicht sinnvoll

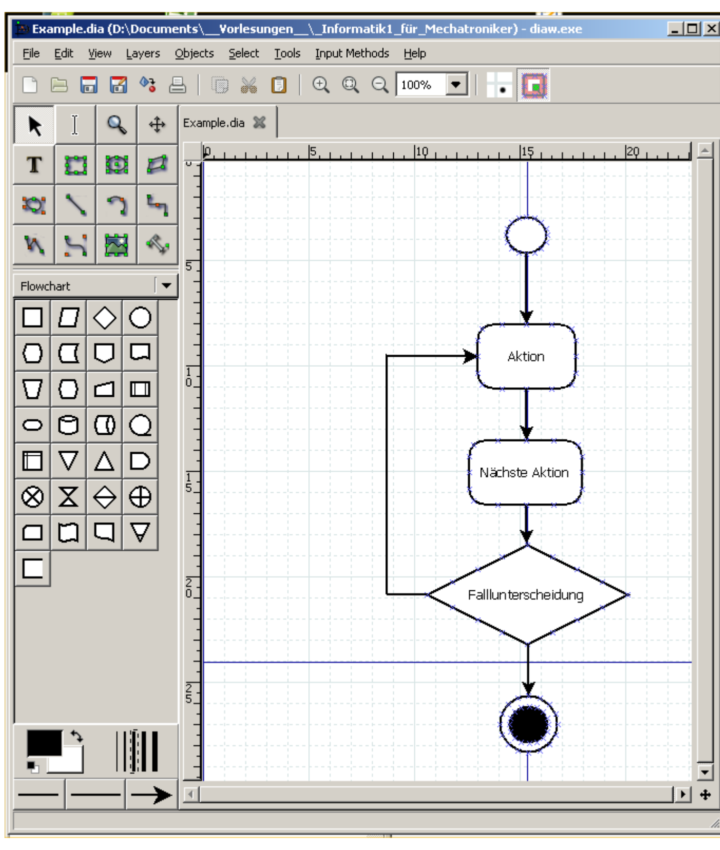
Falls *s>= 1* : vorwärts zählen

Aktivitätsdiagramm

- Aktivitäten lassen sich hierarchisch schachteln
- Eine Aktivität kann wieder als eine Menge von Detail-Aktivitäten (ggf. dargestellt in weiterem Aktivitätsdiagramm) aufgefasst werden
- **Wichtig:** Die Ein- und Ausgänge müssen aber dementsprechend übereinstimmen!
- Diese Methode spiegelt das „Divide and Conquer" wieder
- Als Symbol wird ein Aktivitätssymbol mit zwei enthaltenen Aktivitätssymbolen verwendet

Zusammengesetzte Aktivität

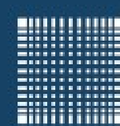
Aktivitätsdiagramm



Dia-Portable

- Auf www.portableapps.com
- Freeware
- Drag'n drop
- beherrscht auch andere UML-Darstellungen

Pseudocode



3 - Programmentwurf

Pseudocode:

- Pascal-ähnliche Notation
- Man vermeidet Programmiersprachen spezifische Ausdrücke wie z.B.:

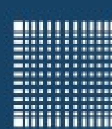
  a+=1 ; a = a>3?b:c ; a=a++ + ++a ; [](float a, float b) {...}
- Ist nicht standardisiert

```
begin BetragPruefen
Eingabe(a)
Eingabe(b)
if a > b then
    Ausgabe(a)
else
    Ausgabe(b)
end if
end BetragPruefen
```

4 - Erste Schritte

Kapitel 4 – Erste Schritte

- Eclipse + Java Development Kit (JDK)
- Kommentare
- Namenskonventionen
- Ein- und Ausgabe mit JAVA



4 - Erste Schritte

Eclipse und JDK

1. Laden Sie sich das neueste Java JDK von Oracle herunter.

   (achten Sie darauf ob ihr System ein 32 oder 64 Bit System ist)

2. Installieren Sie das JDK

3. Laden Sie sich Eclipse (oder Eclipse Portable) herunter:

   http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/marsr

4. Installieren Sie sich Eclipse

5. Starten Sie Eclipse

Ein erstes JAVA Projekt

New Project

Select a wizard

Create a Java project

Wizards:

type filter text

General
CVS
Java
Java Project
Java Project from Existing A
Maven
Examples

Java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Package Exp...

myFirstProject

New

Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W

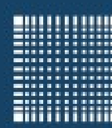
Copy Ctrl+C
Copy Qualified Name

Java Project
Project...
Package
Class
Interface
Enum

Rechte Mouse-Taste

Ein erstes JAVA Projekt

Klassennamen eintippen



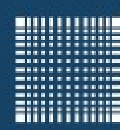
4 - Erste Schritte

Ein erstes JAVA Projekt

Das erste Programm:

```
public class test{
 public static void main(String[] argc) // start of the main program
 {
  System.out.println("Hello World! "); // output of „Hello World!“
 }
}
```

Achtung: JAVA unterscheidet Groß-/Kleinschreibung (=case sensitive)!!!



4 - Erste Schritte

Kommentare

- mit // wird der Rest der Zeile auskommentiert
- mit /* Kommentar */ wird alles zwischen /* und */ auskommentiert und vom Compiler nicht beachtet
- Für die Übung bitte über jedes Programm eine kurze Beschreibung, was es macht und welche Funktionen enthalten sind und…
- …über jede Methode eine kurze Funktionsbeschreibung, was jeder Übergabeparameter bedeutet und was der Rückgabewert bedeutet

Kommentare - warum?

- Software wird meist in Teams erstellt
    - Andere müssen Ihren Code verstehen und nachvollziehen können
        - um Fehler zu beheben
        - um Ihre Programme weiter zu entwickeln
    - Sie selbst müssen auch nach längerer Zeit Ihren Code noch
        - verstehen
        - erweitern
        - verbessern
        - Debuggen
- 60 – 70% aller Entwicklungsarbeiten und –kosten entstehen durch Wartung und Weiterentwicklung!

Kommentare (wie?)

- Jede Methode sollte in Zukunft auf **Englisch** dokumentiert werden (Beispiel: javadoc kompatible Dokumentation):

```
/***************************************************************************
 @brief      Description:  generateThread
             This function generates a thread.
 @param      threadNum   : is of type integer and defines the
                           Threadnumber...
 @return                 : SUCCESS   = thread was generated
                           FAILED    = thread couldn't be generated
***************************************************************************/
char generateThread(int threadNum)
{
…
}
```

Modularisierung

- JAVA bietet Klassen und Pakete zur Modularisierung des Quelltextes an
- Mit Paketnamen lassen sich zusammengehörige Klassen zu einer Einheit zusammenfassen
- Mit unterschiedlichen Paketnamen können Klassen, die eigentlich wegen einer Namenskollision nicht kombinierbar wären, trotzdem in einem Programm genutzt werden

Modularisierung (Zugriff auf Pakete)

- Auf Klassen im selben Paket kann direkt zugegriffen werden
- Klassen in anderen Paketen müssen mit Hilfe des Punktoperators angesprochen werden:

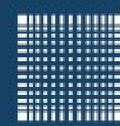
```
de.companyXY.graphics.Screen myScreen = new Screen();
```

- Mit der import-Anweisung wird diese Schreibweise vereinfachen:

```
import de.companyXY.graphics.*;

Screen myScreen = new Screen();
```

- Oder wie von SUN empfohlen, nur die eigentliche Klasse einbinden:

```
import de.companyXY.graphics.Screen;

Screen myScreen= new Screen();
```



Modularisierung (Definition von Paketen)

- Jede Klasse kann mit Hilfe einer `package`-Anweisung einem Paket zugeordnet werden
- Wenn Pakete hierarchisch geschachtelt sind, bleiben über- und untergeordnete Pakete trotzdem logisch unabhängig voneinander

`Beispiel:`

```
package de.companyXY.graphics

public class Screen{

}
```

Hier wird die Klasse Screen auf das Verzeichnis ./de/companyXY/graphics relativ zum Projektverzeichnis abgebildet.

Namenskonventionen

Sprache

- Kommentare und Variablen bitte auf Englisch

Variablen sinnvoll benennen

- selbsterklärend
- nicht *i1* oder *k19*
- nicht übermäßig lang (Tipparbeit!)
- Beispiel: *maximum* oder max statt *m*

Abweichung: Schleifenzähler + Hilfsvariablen kurz und prägnant, z.B.:

- *c* für einen *char*-Wert
- *d, e, f* für einen *double*-Wert
- *i*, *j*, *k* für *int*-Werte

Namenskonventionen

Namenskonventionen

- werden von der Programmiersprache vorgegeben
- haben sich als vorteilhaft erwiesen
- sollen die Lesbarkeit von fremdem Codes erhöhen
- werden z.B. in mitgelieferten Funktionsbibliotheken eingehalten
- sind einzuhalten in Übungen und Klausur!

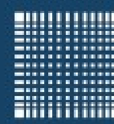
Lokale Variablennamen

- beginnen mit Kleinbuchstabe
- weiter mit Kleinbuchstaben
- bei neuem Wortstamm einen Großbuchstaben mittendrin,
  z.B. *numberProfessors*
- Keine Unterstriche (_) mehr → veraltet,
  z.B.: number*Professors statt number_professors*

Ein- und Ausgabe mit JAVA

- Die „print“ Anweisung

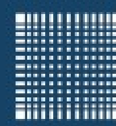
```
public class Test{
 public static void main(String[] args)
 {
  System.out.print("Ganze Zahl = " + 5);// kein Zeilenumbruch
  System.out.println("...");// mit Zeilenumbruch
 }
}
```



4 - Erste Schritte

- Eingaben können in Java mit Hilfe der Scanner Klasse gemacht werden:

```
import java.util.*;

public class MyClass {
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);

        // Input functions for different datatypes
        int i    = sc.nextInt();
        float f  = sc.nextFloat();
        double d = sc.nextDouble();
        String s = sc.nextLine();
    }
}
```



4 - Erste Schritte

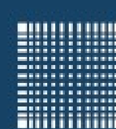
Und ein Beispielprogramm…

```
import java.util.*;

public class MyClass{
  public static void main(String[] argc){
      Scanner sc = new Scanner(System.in);
      System.out.print("Bitte Länge in Inch eingeben:")
      double length = sc.nextDouble();
      System.out.println("\n entspricht:" + length*2.5 + " cm");
  }
}
```

5 - Die Programmiersprache JAVA

Kapitel 5 – Die Programmiersprache JAVA

- Schlüsselwörter
- Ausdrücke
- Anweisungen
- Token
- Literale



5 - Die Programmiersprache JAVA

Schlüsselwörter:

abstract	continue	goto	package	this
assert	default	if	private	throw
boolean	do	implements	protected	throws
break	double	import	public	transient
byte	else	instanceof	return	try
case	extends	int	short	void
catch	final	interface	static	volatile
char	finally	long	super	while
class	float	native	switch	
const	for	new	synchronized	

Ausdrücke:

- liefern einen Wert als Ergebnis
- haben bestimmten Typ (z.B. *int*, *char*)

z.B.

- Literale (Wertkonstanten), z.B. *5.0*
- Variablen, z.B. *x*
- Arithmetische Ausdrücke, z.B.: *5*x+3*y*
    - liefern ganze Zahl oder Fließkommazahl
- Boolesche Ausdrücke, z.B.: *(x && y) || z*
    - liefern *true* (in C: *1*) oder *false* (in C: *0*)
- Vergleichsausdrücke, z.B.: *x>3*
    - liefern *true* (in C: *1*) oder *false* (in C: *0*)

Anweisungen

- definieren Programmablauf
- enthalten / nutzen Ausdrücke

z.B.

- Zuweisungen
- Folgen von Anweisungen
- Prozedur-, Funktions-, Methoden-Aufrufe
- Bedingte Abfragen
- Schleifen
- etc.

Token

- zusammenhängendes Wort gebildet aus einem Alphabet
  - Alphabet: z.B. Buchstaben, aber auch: Zahlen, _

- getrennt vom nächsten Token durch Trennzeichen
  - Whitespaces: Leerzeichen, Zeilenumbruch, Tabulator
  - Weitere mögliche Trennzeichen:
    - Klammern
    - Komma
    - Semikolon

Literale

- konkrete Angabe eines Zahl-, Zeichen- oder Zeichenkettenwertes im Quellcode z.B. “Hello World 1234“
- Wertkonstante z.B. 6.4564

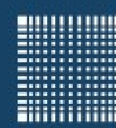
6 - Variablen und Datentypen

Kapitel 6 – Variablen und Datentypen

- Elementare Datentypen
- Variablen
- Konstanten
- Strings
- Arrays

Elementare Datentypen

Datentyp	Wertebereich	Beschreibung
boolean	true, false	zweiwertig
byte	-128..127	ganzzahlig
short	-32768 .. 32767	ganzzahlig
char	-32768 .. 32767	ganzzahlig/ Buchstaben
int	-2147483648 .. 2147483647	ganzzahlig
long	-9223372036854775808 .. 9223372036854775807	ganzzahlig
float	+-1.40239846 10^-45 .. +-3.40282347 10^38	Fließkommazahl mit etwa 7 Stellen Genauigkeit
double	+-4.9 10^-324 .. +-1.79 10^308	Fließkommazahl mit etwa 15 Stellen genauigkeit



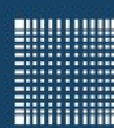
6 - Variablen und Datentypen

Variablen

- Platzhalter eines definierten Datentyps.
- Muss vor dem Gebrauch deklariert werden
- Muss vor Gebrauch initialisiert werden
- Ist nur innerhalb des Blocks { }, in der sie deklariert ist, gültig!

Beispiele:

```
float zahl1;

int   zahl2 = 5, zahl3 = 7;

double dPi = 3.1415927;

float fPi = 3.1415927f;
```



6 - Variablen und Datentypen

Konstanten

- Konstanten werden mit vorangestelltem Schlüsselwort final deklariert
- Sie müssen vor dem Gebrauch deklariert werden
- Bei der Deklaration wird i.Allg. ein Datentyp festgelegt und ein Wert übergeben.

Beispiele:

```
final int   zahl2 = 5, zahl3 = 7;

final double dPi = 3.1415927;

final float fPi = 3.1415927f;
```

Strings

- String-Literal
  - Es beginnt und endet mit "Anführungszeichen".
  - Dazwischen dürfen beliebig viele Zeichen (auch Escape-Codes) stehen.
- Beispiele für String-Literale:
  - *"Hallo zusammen!"*
  - *"Hallo Java-Programmierer!\nWie geht's?\n"*
- Anwendung z.B.:
  - *System.out.print("Hallo Java-Programmierer!\nWie geht's?\n");*

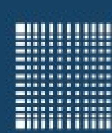
Strings

- Strings können Zeichenketten beliebiger Länge enthalten
- Zeichen sind als Unicode in jeweils 2 Byte gespeichert

z.B.

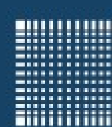
```
String vorname = "Dieter";

int laenge = vorname.length();
```

7 - Arithmetische Ausdrücke



Kapitel 7 – Arithmetische Ausdrücke

- Auswertungsreihenfolge
- Rechnen mit unterschiedlichen Datentypen
- Operatoren
- Bitweise Operatoren



7 - Arithmetische Ausdrücke

Auswertungsreihenfolge

- Ein berechnender Ausdruck wird von links nach rechts ausgewertet
- Klammern werden von innen nach außen ausgewertet
- Zuweisungen werden von rechts nach links ausgewertet.

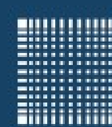
7 - Arithmetische Ausdrücke

Rechnen mit unterschiedlichen Datentypen

Datentypen werden automatisch einander angeglichen:

Beispiel, wie die Virtuelle Maschine die erste Zeile interpretiert:

```
double d = 3.0f + 4;
double d = 3.0f + 4.0f;
double d = 7.0f;
double d = 7.0d;
```



Rechnen mit unterschiedlichen Datentypen

Wenn Zieldatentyp „größer“ bzw. genauer als Datentyp des Wertes

- Automatische Datentypumwandlung

byte→char/short → int → long → float → double

Wenn Zieldatentyp „kleiner“ bzw. ungenauer als Datentyp des Wertes

- Datentypumwandlung sollte durch Type Cast erzwungen werden, da sonst Fehler entstehen können.

- Compiler merkt das **nicht immer**!
- Schreibweise: (*<Zieldatentyp>*) *<Wert>*

```
short sh;
long l = 18;
sh = (short) l;
```

Rechnen mit unterschiedlichen Datentypen

Rundungsfehler:

- double → float: nächste darstellbare Zahl
- double → long, int, short, char oder byte: Nachkommastellen abschneiden + abschneiden überzähliger vorderer Bits
- float → long, int, short, char oder byte: Nachkommastellen abschneiden + abschneiden überzähliger vorderer Bits

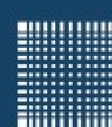
Rechenfehler

- Ursache: Abschneiden überzähliger vorderer Bits
- long → int, short, char oder byte:vordere 32, 48 oder 56 Bits abschneiden
- int → short oder char: vordere 16 oder 24 Bits abschneiden
- short → byte: vordere 8 Bits abschneiden

Beispiele:

(int)3.1415 → 3

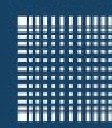
(byte)120 → 120



7 - Arithmetische Ausdrücke

Operatoren (Auswertungsreihenfolge)

Klammern	() []	13
Negation Inkrement Dekrement	- ! ~ ++ --	12
Arithmetische Operatoren	* / %	11
	+ -	10
Shift Operatoren	<< >> >>>	9
Vergleichsoperatoren	> >= < <=	8
	== !=	7
Bitweise Operatoren	&	6
	^	5
	|	4
Logische Operatoren	&&	3
	||	2
Zuweisungsoperator	= += -= *= /= %= >>= <<= &= ^= |=	1



7 - Arithmetische Ausdrücke

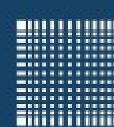
Operatoren

Es gibt die folgenden abkürzende Zuweisungsoperatoren:

	*=	/=	%=	+=	-=	<<=	>>=	>>>=	&=	^=	|=

Beispiel:

```
a = a + 1;
// kürzer geschrieben:
a += 1;
```



7 - Arithmetische Ausdrücke

Operatoren

Weitere abkürzende Schreibweisen der Zuweisung

- **Inkrement**-und **Dekrement**-Operator **++** und **--**
- **Beispiel:**
  a = a + 1; oder auch *a += 1;*
  kürzer geschrieben:
  a++*;*
  oder
  ++*a;*

Präfix-Operatoren: Stehen vor Variable, z.B: **++**a

Postfix-Operatoren: Stehen nach Variable, z.B: a**--**

Operatoren

Unterschied zwischen Präfix- und Postfix-Operatoren:

Präfix-Operator

- **erst** inkrementieren / dekrementieren
- **dann** mit neuem Wert im Ausdruck weiterrechnen

Postfix-Operator

- mit altem Wert im Ausdruck weiterrechnen
- **dann** erst dekrementieren / inkrementieren

Beispiel:

```
int a, b = 1;
a = b++;     // b == 2,  a == 1
a = ++b;     // b == 3,  a == 3
```

7 - Arithmetische Ausdrücke

Operatoren

- Prä- und Postinkrement gibt es auch in der Programmiersprache C.
- Der Postinkrement in Java verhält sich jedoch z.T. anders als in C:
- Wenn das Ergebnis eines Postinkrements der Variablen zugewiesen wird, die inkrementiert wird, so wird der Postinkrement ignoriert.
- Das ist in etwa so, als würde die Zuweisung erst in eine Hilfsvariable abgelegt werden, um sie dann der Variablen selbst zuzuweisen.

Beispiel in JAVA:

```
int a = 0;
a = a++;     // a == 0
a = ++a;     // a == 1
```

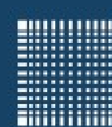
Operatoren

Sinnvolle Verwendung von Inkrement und Dekrement

- Zählschleifen
- Arrayindizierung

Inkrement und Dekrement in komplexen Ausdrücken

- erschwert Lesen von Programmen
- erschwert Fehlersuche
- ▸ vorsichtig und sparsam einsetzen!



7 - Arithmetische Ausdrücke

Bitweise Operatoren

Operator	Beschreibung
|	bitweise oder (OR)
&	bitweise und (AND)
~	bitweise nicht (NOT)
^	Ausschließendes oder (XOR)
<<	links Verschiebung (shift left)
>>	rechts Verschiebung (shift right) mit Vorzeichen Erweiterung
>>>	Rechts Verschiebung (shift right) ohne Vorzeichen Erweiterung

Bitweise Operatoren

Bitweise ODER (OR):

0	0	0
0	1	1
1	0	1
1	1	1

Beispiel:

  0 1 0 0 0 1 0 1 0 0 1 $_{bin}$ = 553_{dez}

| 1 0 0 1 0 1 0 0 1 0 0 $_{bin}$ = 1188_{dez}

= 1 1 0 1 0 1 0 1 1 0 1 $_{bin}$ = 1709_{dez}

Bitweise Operatoren

Bitweise UND (AND):

0	0	0
0	1	0
1	0	0
1	1	1

Beispiel:

$\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1_{bin} = 553_{dez}$

$\&\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 0_{bin} = 1188_{dez}$

$=\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0_{bin} = 32_{dez}$

Bitweise Operatoren

Bitweise NICHT (NOT):

0	1
1	0

Beispiel:

~ 0 1 0 0 0 1 0 1 0 0 1 $_{bin}$ = 553$_{dez}$

= 1 0 1 1 1 0 1 0 1 1 0 $_{bin}$ = 1494$_{dez}$

Bitbreite ist entscheidend für das Ergebnis

Bitweise Operatoren

Bitweise ausschließendes ODER (XOR):

0	0	0
0	1	1
1	0	1
1	1	0

Beispiel:

$$
\begin{array}{ll}
 & 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1_{bin} = 553_{dez} \\
\wedge & 1\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 0_{bin} = 1188_{dez} \\
= & 1\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1_{bin} = 1677_{dez}
\end{array}
$$

Bitweise Operatoren

Verschiebung nach links (SHIFT LEFT):

Nach links verschieben um x Stellen entspricht der Multiplikaton mit 2^x

Beispiel:

$0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1_{bin} << 1 \quad = \quad 553_{dez} * 2$

$0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 0_{bin} \quad = \quad 1106_{dez}$

$0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1_{bin} << 2 \quad = \quad 553_{dez} * 4$

$1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 0_{bin} \quad = \quad 2212_{dez}$

Bitweise Operatoren

Verschiebung nach rechts (SHIFT RIGHT):

Nach rechts verschieben um x Stellen entspricht der Integer-Division durch 2^x

Beispiel:

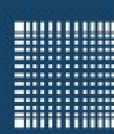
$0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1_{bin} >> 1 \quad = \quad 553_{dez}\ /\ 2$

$0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0_{bin} \quad = \quad 276_{dez}$

$0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1_{bin} >> 2 \quad = \quad 553_{dez}\ /\ 4$

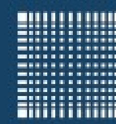
$0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0_{bin} \quad = \quad 138_{dez}$

8 - Anweisungen



Kapitel 8 – Anweisungen

- Ein-Ausgabe Anweisungen
- Schleifen
- Fallunterscheidungen
- Sprungbefehle



8 - Anweisungen

Ein-Ausgabe Anweisung (mit Hilfsfunktionen)

```
import java.io.*
import java.util.*
public class MyClass{
 public static void main(String[] arg)
 {
  Scanner sc = new Scanner(System.in);
  int zahl = sc.nextInt();
  double kommaZahl = sc.nextDouble();
  String text = sc.nextLine();

  System.out.println("integer zahl    = " + zahl);
  System.out.println("float kommaZahl = " + kommaZahl);
  System.out.println("string text     = " + text);
 }
}
```

Schleifen (for)

Möchte man einen oder mehrere Befehle wiederholen, so kann man die for-Schleife nutzen. Sie besteht aus drei durch Semikolon getrennten Bereichen:

- Die **Initialisierung** legt den Startwert des Zählers fest) z.B.: int i=0
- Die **Zählbedingung**, legt fest bis zu welchem Wert gezählt wird z.B.: i < 100
- Die **Änderung des Zählwertes** für jeden Durchlauf z.B.: i = i + 2

```
for  ( Initialisierung ; Zählbedingung ; Änderung des Zählwertes){

     // Zu wiederholender Anweisungsblock

}
```

Schleifen (for)

Wird die Zählbedingung leer gelassen, bedeutet das, dass sie immer erfüllt ist und eine Endlosschleife entsteht, die nur durch eine `break;` Anweisung im Schleifenrumpf abgebrochen werden kann.

Beispiele Endlosschleife:

```
for (;;){

}
```

Beispiele Zähler von 0 bis 99:

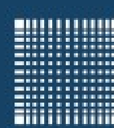
```
for (int zaehler = 0; zaehler < 100; zaehler++){
   System.out.println("zaehler hat den Wert:"+zaehler);
}
```

Schleifen (for)

Die for Schleife ermöglicht zudem mehrere Initialisierungsbefehle, die mit Komma getrennt werden. Auch der Teil der For Schleife, in der der Zähler geändert wird kann mehrere mit Komma separierte Befehle enthalten.

Beispiel zwei Zähler zählen: i rückwärts, j vorwärts

```
for (int j = 0, int i = 100; j < 100; j++, i--){
   System.out.println("i,j haben den Wert:" + i + "," + j);
}
```



8 - Anweisungen

Schleifen (For each)

Die For each Schleife ermöglicht es alle Elemente eines Arrays zu durchlaufen.

Dabei nimmt die Variable die Werte der Array-Elemente an.

```
int[] myArray = new int[]{5, 7, 3, 2};
for( int k: myArray ){
   System.out.println("k = "+k);
}
```

Schleifen (while)

„while“ bedeutet „solange“, darauf folgt die Bedingung in runden Klammern, darauf der zu wiederholende Anweisungsblock in geschweiften Klammern.

```
while( Bedingung ){    // kopfgesteuerte Schleife
      // zu wiederholender Anweisungsblock
}
```

Beispiel:

```
zaehler = 0;
while (zaehler < 100){
    System.out.println("zaehler hat den Wert:"+zaehler);
    zaehler ++;
}
```

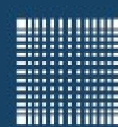
Schleifen (do-while)

Die „`do-while`“ Schleife beginnt mit dem Schlüsselwort „`do`“, wonach direkt der zu wiederholende Anweisungsblock kommt. Erst danach wird mit „while“ die Bedingung in runden Klammern angekündigt.

```
do{
   // zu wiederholender Anweisungsblock
}while( Bedingung )
```

Beispiel:

```
zaehler = 0;
do{
   System.out.println("Zaehler hat den Wert:"+zaehler);
    zaehler++;
}while(zaehler<100);
```



8 - Anweisungen

Fallunterscheidungen (if-else)

Mit der if-else Anweisung ist es möglich, eine Wenn-Dann-Beziehung auszudrücken. Nach einem „if“ kommt die Bedingung in runden Klammern. Ist sie erfüllt, so wird der darauf folgende Anweisungsblock ausgeführt, ansonsten wird ein optionaler else-Block ausgeführt.

```
int variable;
if (variable==1){
   System.out.println("variable ist gleich 1\n");
} // der nachfolgende „else“ Teil ist nicht notwendig
else{
   System.out.println("variable ist ungleich 1\n");
```

Fallunterscheidungen (switch-case)

Die switch-case Anweisung bietet eine schnelle Möglichkeit, viele Vergleiche in einer Anweisung auszuwerten:

```
switch(variable){
   case 1: /* Die Variable ist gleich 1 */ break;
   case 2: /* Die Variable ist gleich 2 */ break;
   default:/* die Variable ist ungleich 1 und ungleich 2*/
}
```

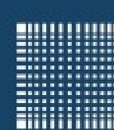
Fallunterscheidungen (? Operator)

Der Fragezeichenoperator ? ermöglicht eine sehr verkürzte Schreibweise einer Fallunterscheidung : `Bedingung ? Anweisung1 : Anweisung2`

Ist die Bedingung erfüllt, so wird Anweisung1 ausgeführt, sonst Anweisung2.

```
str = (A > B ? "A ist größer" : "B ist größer");
```

9 – Arrays, Enumerationen

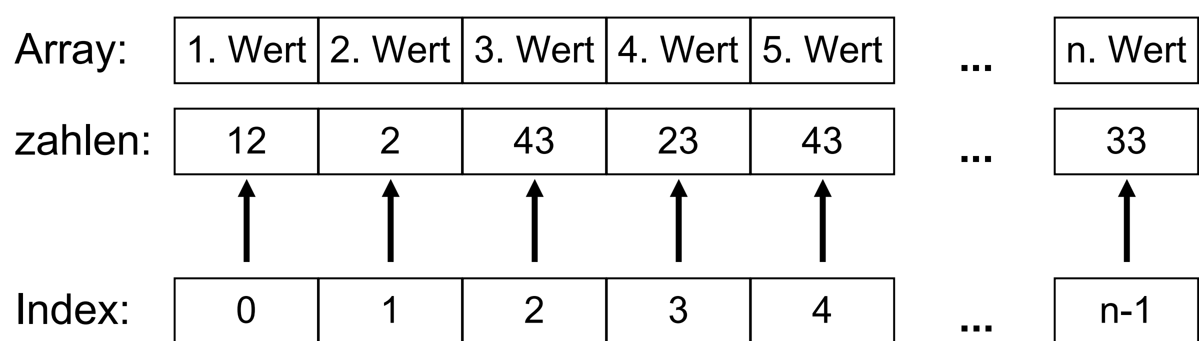


Kapitel 9 – Arrays, Enumerationen

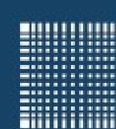
- Arrays (eindimensional)
- Arrays (mehrdimensional)
- Arrays (mit Arrays als Komponenten)
- Enumerationen

Arrays (eindimensional)

- Ein Array (oder ein Feld) hat eine Länge, die bei seiner Definition festgelegt wird
- Auf die verschiedenen Elemente eines Arrays kann mittels Indizierung in beliebiger Reihenfolge zugegriffen werden
- In der Programmiersprache JAVA beginnen die Indizes immer mit **0**



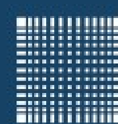
Beispiel für den Zugriff: zahlen[2] == 43
zahlen[n-1] == 33
zahlen[n] == 1 **!!! Fehler!!!!**



9 – Arrays, Enumerationen

Arrays (eindimensional)

- Arrays sind Variablen, werden also wie diese mittels einer Definition angelegt:
- Syntax: <Datentyp> <variablenname>[] = new <Datentyp>[groesse]
- alternativ: <Datentyp> []<variablenname> = new <Datentyp>[groesse]
- Beispiele:
  - *int quadrate[] = new int[30]; //* legt ein Array aus 30 Integer Werten an
  - *float noten[] = new float[40]; //* legt ein Array aus 40 Float Werten an
  - *double feld[] = {1.1, 2.2, 3.3}; //* legt ein Array aus 3 Double Werten an
- Arraylänge nachträglich nicht mehr änderbar!



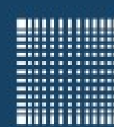
Arrays (mehrdimensional)

- Man kann in Java auch mehrdimensionale Felder anlegen.
- Streng genommen sind das Felder mit Komponenten eines Feldtyps
- `float matrix[][] = new float [3][];` legt beispielsweise ein Array an, bei dem nur die erste Komponente auf 3 festgelegt wird. Folgendes Beispiel zeigt, wie die 2. Komponente auf 5 festgesetzt wird:

```
float matrix[][] = new float [3][];

for (int i=0;i<3;i++)

    matrix[i] = new float[5];
```

- Dies ist gleichbedeutend mit

```
float matrix[][] = new float[3][5];
```



Arrays (mehrdimensional)

Es ist zwar erlaubt, die eckigen Klammern auch vor dem Variablennamen zu setzen:

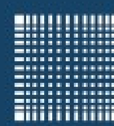
```
int[] var1, var2[], var3[][];
```

Dies ist jedoch gleichbedeutend mit:

```
int var1[], var2[][], var3[][][];
```

was zu Missverständnissen führen kann.

Also BITTE Arrays immer mit Klammern nach dem Variablennamen definieren!



9 – Arrays, Enumerationen

Enumerations

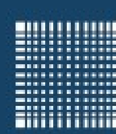
- Mit Hilfe der Enumeration können in Java fortlaufende Konstanten definiert werden:

```
public enum wochentag{MONTAG, DIENSTAG, MITTWOCH, DONNERSTAG,
FREITAG, SAMSTAG, SONNTAG};
```

Benutzt wird das Ganze z.B. folgendermaßen:

```
public static void main(String[] args)
{
 wochentag tag;
 tag = wochentag.MONTAG;
 if (tag == wochentag.MONTAG){
    System.out.println("Es ist Montag");
 }
}
```

10 – Methoden



10 – Methoden

Kapitel 10 – Methoden

- Methodendeklaration
- Methodenaufruf
- Die return-Anweisung

Methodendeklaration

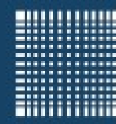
- JAVA Methoden können nur innerhalb von Klassen deklariert werden.
- Sie bestehen aus:
  - Optionalen Modifizierern, wie public, abstract, final usw., die spezielle Attribute der Methode festlegen
  - Dem Typ des Rückgabewertes
  - Einem Methodennamen
  - Einer Liste der Methodenparameter mit vorangestelltem Typ
  - Einer optionalen *throws* Klausel, die anzeigt, welche Ausnahmen durch einen Methodenaufruf ausgeworfen werden können
  - Dem Methodenrumpf, der in { } eingeschlossen die Anweisungen enthält, die beim Methodenaufruf ausgeführt werden.

Methodenaufruf

- Ein Methodenaufruf ist ein elementarer Ausdruck
- Durch Abschluss mit ; wird der Ausdruck zur Anweisung
- Die im Funktionskopf definierten Parameter, werden bei jedem Aufruf mit den übergebenen Variablen initialisiert ,bevor der Funktionsrumpf ausgeführt wird.
- Die im Funktionskopf definierten Parameter sind in der gesamten Funktion gültig und dürfen nicht durch lokale Variablen verdeckt werden.

Methoden Aufruf (Basis- und Referenz-Typ)

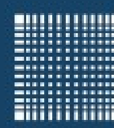
- Wird eine Variable eines Basisdatentyps übergeben, so wird innerhalb der Funktion auf einer Kopie dieser Variablen gearbeitet, die nach Rücksprung aus der Funktion keine Auswirkung auf die übergebene Variable hat.
- Wird eine Variable eines Referenztyps übergeben, so kann man dessen Wert/ Werte innerhalb der Funktion so verändern, dass er auch nach Rücksprung aus der Funktion verändert bleibt.



10 – Methoden

Beispiel für Übergabe als Basistyp:

```
void init(int var) // Variable ist eine Kopie von a auf dem Stack
{
 var = 5; // variable is only changed on the stack
          // leaving the function, the variable is lost
}
public static void main(String[] argc)
{
 int a=3;
 init(a); // contents of a is given to the function init()
 System.out.println("Der Wert von a ist " + a); // output: 3
}
```



10 – Methoden

Beispiel für Übergabe als Referenztyp:

```
public class MyInteger{
 public int value;
}
public class MyClass{
 void init(MyInteger var){ // variable is of reference type
      var.value = 5;        // variable is changed
 }
 public static void main(String[] argc){
  MyInteger a = new MyInteger();
  a.value = 3;
  init(a);
  System.out.println("Der Wert von a ist " + a.value); // output: 5
 }
}
```

Die *return* Anweisung

- Die return-Anweisung beendet den Aufruf einer Methode
- Falls der Ergebnistyp einer Methode „void“ ist, kann `return;` geschrieben werden
- Sonst wird mit `return Ausdruck;` ein Wert des Rückgabetyps zurückgegeben
- Steht das `return Ausdruck;` innerhalb einer bedingten Verzweigung, so muss bei Nichterfüllung sichergestellt werden, dass ein anderes `return Ausdruck;` innerhalb der Funktion aufgerufen wird.

11 – Rekursion

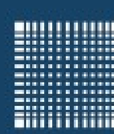
Kapitel 11 – Rekursion

- Definition
- Realisierung
- Beispiel: Iterative Suche
- Beispiel: 3D - Gebirge

11 – Rekursion

Definition

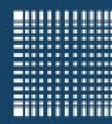
- Ein Funktionsaufruf ist rekursiv, wenn die aufgerufene Funktion innerhalb ihres Ablaufs mindestens noch einmal aufgerufen wird.



11 – Rekursion

Realisierung

- Typischerweise beinhaltet eine rekursive Funktion eine Fallunterscheidung
- Hier wird entschieden, ob mit Rekursion (Selbstaufrufen) fortgefahren wird
- ...oder, ob die Funktion in den aufrufenden Kontext zurückkehrt



11 – Rekursion

Iterative Suche

- Die folgende Methode sollte im sortierten Array number[100], die Zahl 33 suchen.

```
public int search(int number[],int min, int max){
        int index = (max-min)/2;
        if (number[index]>33){
           max = index;
           return search(number,min,max);
        }
        else if (number[index]<33){
           min = index;
           return search(number,min,max);
        }
        else{
           return index;
        }
}
// call with System.out.println(search(number[],0,100);
```

Beispiel: 3D - Gebirge

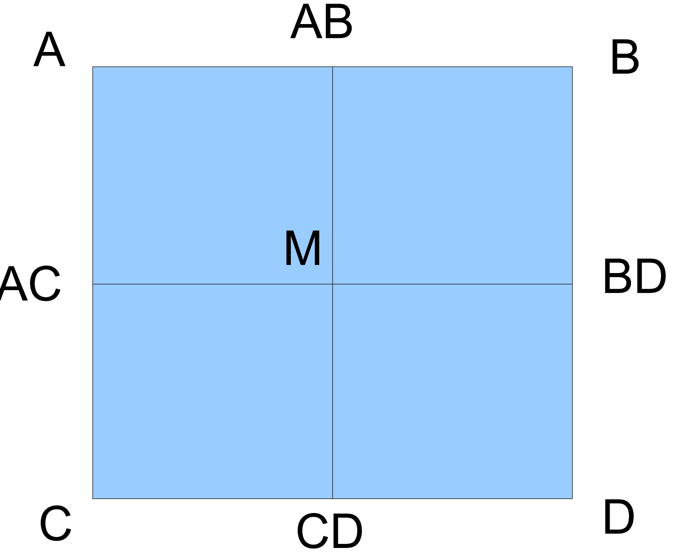
1. Gegeben sei ein Rechteck, indem jede Ecke mit einer zufälligen Höhe initialisiert ist (kann auch 0 sein)
2. Für die Unterteilung AB wird der Mittelwert aus den Höhen A und B gebildet und eine zufällige Abweichung [+/- deviation] aufaddiert
3. Genauso werden AC,BD,CD und M Höhen zugewiesen.
4. Deviation wird halbiert
5. Es werden 4 neue Rechecke bei

   A,AC,M,AB ; AC,C,CD,M ; CD,D,BD,M

   AB,M,BD,B definiert, in denen mit Punkt 2

   fortgefahren wird bis die notwendige Auflösung

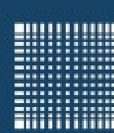
   erreicht ist
6. Man zeichne das Gebirge



Beispiel: 3D - Gebirge

Zusätzlich wird hier alle Höhen <0 auf 0 gesetzt und blau gefüllt (Wasser)

12 – Klassen und Objekte



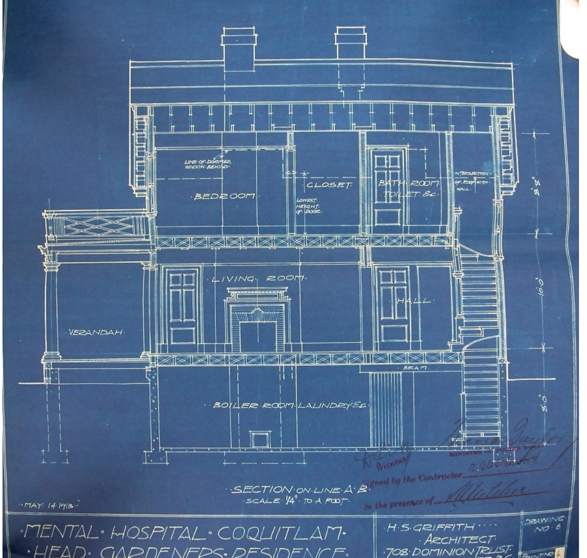
Kapitel 12 – Klassen und Objekte

- Klassendeklaration und Zugriff auf Klassenelemente
- Deklaration von Instanz- und Klassenvariablen
- Der Konstruktor
- Überladen von Methoden
- Der static Initialisierer
- Objektzerstörung

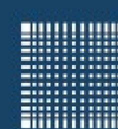
Klassen und Objekte

- Eine Klasse ist ein Bauplan während ein Objekt die mit Hilfe des Bauplans erstellte Umsetzung ist.

Klasse



Objekt



12 – Klassen und Objekte

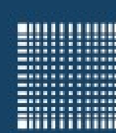
Klassendeklaration und Zugriff auf Klassenelemente

- Mit einer Klassendeklaration definiert man einen Referenztypen und legt deren Implementierung fest.

  Mit der Deklaration einer Klasse „Haus“ würde also eine Referenz (eine Art Zeiger) festgelegt, die nur auf Häuser zeigen kann. Die Klasse Haus definiert dabei, wieviele Zimmer, Türen und Fenster das Haus hat.

- Jede Klasse (außer Object) ist implizit Subklasse der Klasse Object, die Wurzel in der JAVA Vererbungshierarchie ist.

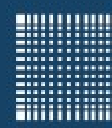
  Am Anfang war also die Klasse „Object“ und alle neu definierten Klassen enthalten die Grundfunktionalität der Klasse Object, z.B. die Funktion toString()



12 – Klassen und Objekte

Klassendeklaration und Zugriff auf Klassenelemente

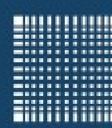
- Innerhalb der Klasse kann Folgendes deklariert werden:
  - Variablen beliebigen Typs
  - Methoden, d.h. Funktionen, die den Zustand des Objekts auslesen oder verändern
  - Konstruktoren, d.h. Methoden, die das Objekt initialisieren
  - Static Initialisierer, d.h. spezielle Anweisungsfolgen zur Initialisierung
  - Eingebettete Klassen, d.h.klassen,die nur lokal benötigt werden



Klassendeklaration und Zugriff auf Klassenelemente

Beispiel einer Klasse:

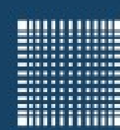
```
import java.io.*;
public class Clock {
    private int hour, minute, second; // variables
    public Clock(int hour, int minute, int second){ // constructor
        this.hour = hour;
        this.minute = minute;
        this.second = second;
        System.out.println("It is " + this.hour + ":"
                  + this.minute + ":" + this.second + " o'clock!");
    }
    public static void main(String[] arg){
        Clock personalClock = new Clock(12,13,22);
    }
}
```



Instanz und Klassenvariablen

- Die Variablen innerhalb einer Klasse sollten, wenn möglich, privat sein
- Auf Klassenvariablen sollte von aussen immer nur mit setter und getter Methoden zugegriffen werden.

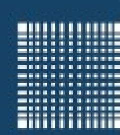
```
import java.io.*;
public class Clock {
    private int hour, minute, second; // variables
    public void setHour(int hour){ this.hour = hour; }
    public int getHour(){ return this.hour }
    public static void main(String[] arg){
        Clock personalClock = new Clock(1,2,3);
        personalClock.setHour(12);// vor allem von aussen setter
        int nowHour = personalClock.getHour();// und getter
    }
}
```



12 – Klassen und Objekte

Instanz und Klassenvariablen

- Die Variablen, die innerhalb der Klasse deklariert sind werden einmal pro Instanz/Objekt angelegt (außer static Variablen)
- Sind sie mit dem Schlüsselwort public versehen, so kann auch von aussen darauf zugegriffen werden. Das geschieht mittels des Punktoperators z.B.: `Objektname.variablenname = 112;`
- Sind Variablen mit dem Schlüsselwort private versehen, so kann nur innerhalb der Klasse direkt auf die Variablen zugegriffen werden. Von aussen sind sie nur über setter und getter Methoden direkt änderbar.



Der Konstruktor

- Der Konstruktor ist eine Methode einer Klasse, die denselben Namen wie die Klasse trägt und keinen Rückgabetypen besitzt
- Diese Methode wird bei der Instanziierung der Klasse aufgerufen.
- In der Methode werden typischerweise Variablen initialisiert und Speicher reserviert.
- Der Standard-Konstruktor enthält keine Übergabeparameter.
- Sollen bei der Instanziierung Werte oder Größen festgelegt werden, so kann der Konstruktor überladen werden, d.h. eine Konstruktormethode mit Übergabeparametern definiert werden.
- Ist der Konstruktor als private deklariert, so kann keine Instanz der Klasse von außen gebildet werden, da ein Aufruf des Konstruktors nötig wäre.

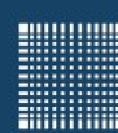
Überladen von Methoden

- Wie wir auf der letzten Folie gelesen haben, kann eine Methode überladen werden, indem die gleiche Methode mit identischem Rückgabetyp, aber anderen Übergabeparametern definiert wird
- Der Compiler entscheidet dann für jeden Aufruf der Funktion, welche der überladenen Funktionen genommen wird.

```
1) public int methode(){return 0;}

2) public int methode(int a){return a;}

3) public int methode(float a){return (int)a;}


methode();     // → Methode 1 wird aufgerufen

methode(5);    // → Methode 2 wird aufgerufen

methode(5.5);  // → Methode 3 wird aufgerufen
```

Static

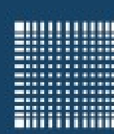
- Das Schlüsselwort static vor einer Variablen sorgt dafür, dass die Variable nur einmal angelegt wird und so nur genau einmal für alle Objekte existiert
- „static“ Variablen existieren schon bevor es eine Instanz der Klasse gibt!!!
- Statische Methoden können nur auf statische Elemente (Methoden, Variablen) zugreifen.
- Beispiel für eine statische Methode ist die main Methode.
- Ein weiteres Beispiel ist das folgende Entwurfsmuster „Lazy Creation“:
  - Instanziiere ein Objekt als eine statische Instanz
  - Mache den Konstruktor privat
  - Stelle eine statische Methode zur Verfügung, die eine Referenz auf die statische Instanz zurückgibt
  - → Das Objekt ist nur einmal instanziierbar!!!



Objektzerstörung

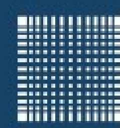
- Der JAVA-Garbage-Collector gibt den reservierten Speicherplatz automatisch frei
- Der Garbage-Collector ist ein Thread niedrigster Priorität, der periodisch überprüft, ob die Instanzen/Objekte noch referenziert werden
- Existiert keine Referenz auf das Objekt, so wird es zerstört bzw. der Speicher freigegeben
- Es gibt keine Möglichkeit außer durch Beenden des Programms den Garbage-Collector gezielt aufzurufen.
- Mit der Methode protected void finalize() throws Throwable{...} kann jedoch in jeder Klasse eine Methode ergänzt werden, die bei Zerstörung des Objekts aufgerufen wird.

13 - Vererbung



Kapitel 13 – Vererbung

- Vererbung
- Verdeckte Variablen und überschriebene Methoden
- Aufrufreihenfolge der Konstruktoren
- Abstrakte Klassen



Vererbung

- Um JAVA Code ohne Copy/Paste sinnvoll wiederverwenden zu können, kann eine Klasse vererbt werden.
- Das geschieht mit dem Schlüsselwort extends (erweitert)
- Alle Methoden der Superklasse (von der geerbt wird) stehen dann automatisch in der abgeleiteten Klasse zur Verfügung.

```
public class StopWatch extends Clock
{
  //  ohne eine Methode hinzuzufügen, kann man alle
  //  in der Klasse Clock definierten Methoden nutzen
}
```

Verdeckte Variablen und überschriebene Methoden

- Wenn Variablen gleichen Namens in der Super- und in der abgeleiteten Klasse deklariert werden, werden die Variablen der Super-Klasse überdeckt. Auf sie kann nur mittels `super:variablenname` zugegriffen werden.
- Auch Methoden gleichen Namens und gleicher Signatur können in der abgeleiteten Klasse Überschrieben werden.

Aufrufreihenfolge der Konstruktoren

- Beim Instanziieren der abgeleiteten Klasse, wird zunächst der Konstruktor der super-Klasse (eventuell davor, der der super-super Klasse etc.) aufgerufen.
- Dadurch werden bei der Initialisierung erst die super-Klassen Variablen initialisiert, dann die der abgeleiteten.

Abstrakte Klassen

- Durch Vorstellen des Schlüsselwortes abstract (z.B. `abstract class Clock`) kann eine Klasse definiert werden, die **nicht instanziiert** werden kann, sondern nur eine Schablone für abgeleitete Klassen bietet
- Auch die Methoden innerhalb der abstrakten Klasse können als abstract definiert werden, ohne Inhalt.
- Abstrakte Methoden müssen in allen abgeleiteten Klassen, die nicht abstract sind, implementiert werden
- Hat eine abstrakte Klasse weder Variablen, noch Methoden, die nicht als abstract ohne Methodenrümpfe definiert sind, so wird die Klasse **„rein abstrakte Klasse“** genannt.

14 - Referenzen

Kapitel 14 – Referenzen

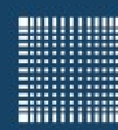
- Referenzen und primitive Datentypen
- Autoboxing und Unboxing
- Kopierkonstruktor

Referenzen und primitive Datentypen

- *Referenzen* sind „Zeiger“ auf Objekte, beinhalten also eine Speicheradresse
- Elementare Datentypen oder *primitive Datentypen* beinhalten den Wert, mit dem gearbeitet wird.
- Wird ein *Referenztyp* einem anderen Referenztyp zugewiesen, so zeigen Beide auf das gleiche Objekt
- Wird ein *primitiver Typ* einem anderen zugewiesen, so ändert sich der Wert.
- Vergleicht man zwei Referenzen mit dem == Operator, so wird deren Adresse verglichen und nur, wenn sie auf dasselbe (nicht das gleiche) Objekt zeigen, ergibt das den Wert true.
- Vergleicht man zwei Variablen eines primitiven Datentyps, so werden ihre Werte verglichen.

Referenzen und primitive Datentypen

- Wird einer Methode eine Referenz übergeben, so wird ihr nur der Zeiger auf das Objekt übergeben und das Objekt kann direkt in der Methode verändert werden.
- Wird einer Methode ein primitiver Datentyp übergeben, so wird eine Kopie des Wertes erzeugt. Eine Änderung der Kopie wirkt sich aber auf das Original nicht aus.



14 - Referenzen

Autoboxing und Unboxing

- Autoboxing: ist die automatische Umwandlung eines *primitiven Datentyps* in das zugehörige Objekt der Wrapper-Klasse (z.B. int -> Integer).

```
Integer weight = new Integer(3);
```

- Unboxing: ist der umgekehrte Vorgang. Die Umwandlung eines Objekts der Wrapper-Klasse in einen *primitiven Datentyp*.

```
int w = weight.intValue();
```

Kopierkonstruktor

- Der Kopierkonstruktor wird aufgerufen, wenn bei der Instanziierung eines Objekts ein anderes Objekt der gleichen Klasse übergeben wird.

```
MyClass myObject = new MyClass(otherObject);
```

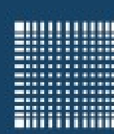
- Wird kein Kopierkonstruktor implementiert, so werden alle Variableninhalte einfach kopiert. Bei primitiven Datentypen ist das kein Problem. Bei Referenzdatentypen zeigt dann aber die Kopie auf das gleiche Objekt, was zu unerwünschten Nebeneffekten führt, denn Objekte wie z.B. myObject und otherObject (siehe obiges Beispiel) würden auf ein gemeinsames Object referenzieren und wären damit nicht mehr voneinander unabhängig.

Kopierkonstruktor

- Ein Kopierkonstruktor muss implementiert werden, wenn in der Klasse Referenztypen benutzt werden, dessen Variablen ebenfalls kopiert werden sollen.

```
public class Car{
    int x;
    CarDisplay displ = new CarDisplay();
   Car(Car obj){
        x = obj.x;
        displ.tacho = obj.displ.tacho;
        displ.clock = obj.displ.clock;
   }
 }
```

15 – Interfaces



15 – Interfaces

15 - Interfaces

- Interfaces
- Interfaceelemente

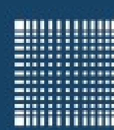
Interfaces

- Interfaces sind ähnlich wie rein abstrakte Klassen
- Sie haben jedoch eine unabhängige Vererbungshierarchie
- Es kann public und abstract vorangestellt werden, gefolgt vom Schlüsselwort interface, dann der Name des Interfaces und eventuell extends und eine Liste von Superinterfaces.

```
public interface Konto{
       double abheben(double betrag);
       double einzahlen(double betrag);
}
public class Girokonto implements Konto{
       ....
}
```

Interfaces Elemente

- Variablen, die in einem Interface deklariert werden sind implizit public static und final. Es sind klassenspezifische symbolische Konstanten
- Neben den Methoden und Variablen, die im Interface deklariert sind hat das Interface die aus direkten Superinterfaces geerbten Methoden und Variablen.
- Alle Methoden eines Interfaces sind implizit abstract, d.h. bei ihrer Implementierung werden nur Ergebnistyp und Signatur einer Methode festgelegt. Ein Methodenrumpf entfällt



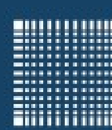
15 – Interfaces

Interfaces als Ersatz für Mehrfachvererbung

- Warum also Interfaces, wenn man auch abstrakte Klassen nehmen könnte?
- Java erlaubt keine Mehrfachvererbung, es ist also in Java nicht möglich von mehreren Klassen zu erben.
- Ein Beispiel für Mehrfachvererbung ist, wenn man ein Amphibienfahrzeug von der Klasse Fahrzeug und Boot ableitet, um sowohl die Methoden eines Fahrzeugs, als auch die eines Bootes benutzen zu können.
- Als Ersatz erlaubten die Entwickler der Sprache Java die Möglichkeit mehrere Interfaces in einer Klasse implementieren zu können

```
class AmphibienFahrzeug implements Fahrzeug implements Boot
```

16 – Exceptions



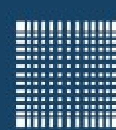
15 – Exceptions

16 - Exceptions

- Geprüfte-, ungeprüfte Exceptions und Fehler
- Behandlung einer Exception (Try-Catch-Finally)
- Beispiel

Geprüfte-, ungeprüfte Exceptions und Fehler

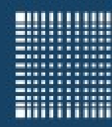
- Geprüfte Exceptions (Ausnahmebehandlung) werden zur Compilierzeit vom Compiler geprüft
- Ungeprüfte Exceptions treten während der Laufzeit auf und werden vom Compiler nicht geprüft. (Beispiel Division durch 0)
- Fehler können normalerweise nicht rückgängig gemacht werden.



16 – Exceptions

Behandlung einer Exception

- Falls von einer Methode eine Exception ausgelöst werden kann, die Behandlung aber irgendwo in der Aufrufhierarchie erfolgen soll, so muss in der Methodendeklaration die throws-Klausel stehen.
- Der Programmteil, der eine Exception auslösen kann, wird in einen try-Block eingeschlossen.
- Der Programmteil, der die Exception behandelt folgt, in einem catch-Block.
- Dabei sollten zuerst die spezielleren Ausnahmebehandlungen mit einem catch Block abgefangen werden, gefolgt von allgemeineren catch Blöcken.
- Am Ende einer in einem try-catch Block behandelten Ausnahme, wird ein optionaler finally Block ausgeführt. Dies geschieht unabhängig davon, ob eine Ausnahmebehandlung stattgefunden hat, oder nicht.
- Eine Exception Behandlung kann kontrolliert mit dem Schlüsselwort throw ausgelöst werden (Beispiel: throw new EOFException();)



16 – Exceptions

Beispiel

```
public class Main {
    public static void main(String[] args) {
        try {
            int test[] = new int[100];
            test[200] = 3;
        } catch (ArrayIndexOutOfBoundsException aobEx) {
            System.out.println("Array index out of bounds...");
            aobEx.printStackTrace();
        } catch (Exception e){//all other exceptions
        }
    }
}
```