JAVA

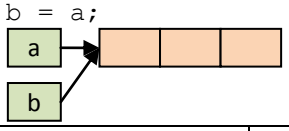
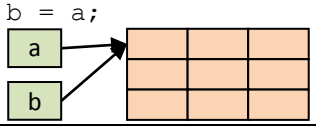
Dateihandhabung	.java						Java-File	
Kommentar	`// Kommentar` `/* Kommentar */`						einzeiliger Kommentar mehrzeiliger Kommentar	
Garbage Collection	ja						Automatische Speicherbereinigung	
Case-Sensitiv	ja						Achtung auf Gross-/Kleinschreibung	
Operanden	+	-	*				Grundoperationen (Punkt vor Strich)	
	%	/					Modulo (Rest einer Division)	Ganzzahldivision bei Ganzzahl normale Division bei Fliesszahl
	==	!=	=				Vergleich	Ungleich	Zuweisung
(! vor && vor ||)	&&	||	!				AND	OR	Negation
	<	>	<=	>=			kleiner	grösser	kleiner gleich	grösser gleich
	++	--					Inkrement / Erhöhung	Dekrement / Vertiefung
	+=	-=	*=	/=	%=		Operation mit direkter Zuweisung	
Eingabe	`a = In.read();`						über Tastatur	
Ausgabe	`System.out.print("Hallo");` `System.out.println("Hello");`						Auf Console, ohne Zeilenumbruch Auf Console, mit Zeilenumbruch	

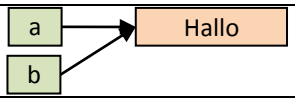
Datentypen	Ganzzahl	`byte`	8 Bit	$-128\ bis\ 127$
		`short`	16 Bit	$-32768\ bis\ 32767$
		`int` (Standard)	32 Bit	$-2\ Mio\ bis\ 2Mio$
		`long`	64 Bit	$-9 * 10^{18}\ bis\ 9 * 10^{18}$
	Zeichen	`char`	16 Bit	Unicode 16
	Wahrheitswert	`boolean`	1Bit	true, false
	Gleitkomma	`float`	32 Bit	$-2\ Mio\ bis\ 2Mio$
		`double` (Standard)	64 Bit	$-9 * 10^{18}\ bis\ 9 * 10^{18}$
	Unverwendet	`null`	-	-

Umwandlungen / Casts	`b = (byte) i;`	int in byte
	`string = i + "";`	int to String
	`i = Integer.parseInt(s);`	String to int

Variablen / Arrays	**Variablen**	**1D Array**	**2D Array**
Überschreitungen sind nicht möglich	a 1 b 2	a[0] 2 a a[1] 3 a[2] 5	a[][0] a[][1] a[][2] a a[0] 2 3 5 a[1] 4 8 1
Grösse		`a.length;` //3	`a.length;` //2 `a[0].length;` //3

Deklaration	`int a, b;`
	`int a = 1;`
		`int[] a;`
		`int[] a = new int[3];`
		`int[] a ={2,3,5};`
			`int[][] a;`
			`int[][] a = new int[2][3];`
			`int[][] a = {{2,3,5}{4,8,1}};`

Zuweisung	`b = 2;`	`a[1] = 3;`	`a[1][2] = 1;`
Abfrage	`b;` //2	`a[2];` //5	`a[0][2];` //5
Zuweisung	`a = b;`	`b = a;`	`b = a;`
bedeutet	a 2		

char	`b1 = 'a';`		Deklarieren	
	`b2 = (char)(b1 + 1);` //b `(int) A;` //65 `(char) 97;` //a		Konvertieren	
	`ch[] a = {'B','3','ü'};`		Array	
String unveränderbar immer neu angelegt	`String a, c;`	`String b="\"1";`	Deklaration	Deklaration mit Zuweisung
	`b=a;`			Achtung: Strings werden referenziert!
	`c=a+b;`	`a.equals("a");`	Zusammenfügen	Vergleich
Stringbuffer veränderbar	`Stringbuffer sb = new StringBuffer(„Hallo Welt");`			Deklaration
	`StringBuffer sb = new StringBuffer(s);`			String to Stringbuffer
	`String s = sb.toString();`			StringBuffer to String

if-then-else	`if (a == 2) {`
	`  //Anweisung wenn a=2`
	`}else if(a == 3){`
	`  //Anweisung wenn a=3`
	`}else{`
	`  //Anweisung sonst`
	`}`
		If Anweisung, muss vom Typ bool sein
		Durchläuft eine der 3 Anweisungen
		'else if' ist optional
		
		'else' ist optional

while	`while (i<100) {`
	`  //Anweisung solange i<100`
	`{`
		Durchlauf solange die Bedingung wahr ist

do-while	`do {`
	`  //Anweisung min. 1 mal`
	`} while (i<100);`
		Anweisung wird min. einmal durchgeführt
		Wiederholung solange wahr

switch	`switch (m) {`
	`case 1: case 2:`
	`  //Anweisung`
	`  break;`
	`case 3:`
	`  //Anweisung`
	`  break;`
	`default:`
	`  Out.print("error");`
	`}`
		Unterscheidung eines Wertes, String nicht erlaubt
		Fall wenn m=1 oder m=2
		break beendet case
		
		Fall wenn m=3
		
		default wird genommen, falls nichts anderes zutrifft

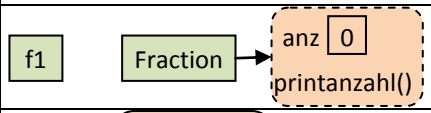
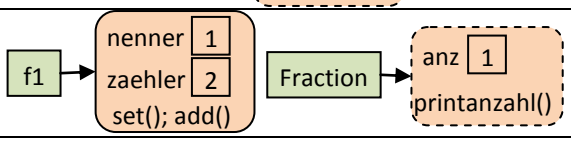
for	`for (int i=1;i<=10;i++){`
	`  //Anweisung solange i<=10`
	`}`
		Zählschleife, Deklariert die lokale Variable i
		Durchläuft solange die Bedingung wahr ist
		und erhöht dann 'i' um 1

Klassen	`class Rectangle{` `}`	Für Objekte Vererbung

Methode ohne Parameter	`void printHeader(){`
	`  //Anweisung`
	`}`
		kleingeschrieben, void = kein Rückgabewert
		Aufruf: `printHeader();`

Methode mit Parameter	`void sub(int x, int y) {`
	`  //Anweisung`
	`}`
		x,y = Übergabewerte
		Aufruf: `sub(100,200);`

Funktionen mit Rückgabewerten	`int max(int x, int y) {`
	`  return x+y;`
	`}`
		int = Rückgabetyp
		return gibt Wert zurück und bricht Funktion ab
		Anwendung: `z = max(a,b);`

Schlüsselwörter	`this` `this.code = code;`	Referenz auf eigene Objekt, bei gleichen Variablennamen
	`this(code, ID);`	Aufruf eines anderen Konstruktors der gleichen Klasse
	`static` `static int count;`	Klassenvariablen, unabhängig von Objekten (z.B. zählen)
	`static e(int a, int b)`	Funktionssammmlung (zur Speichererspanis)
	`static int PI = 3;`	Konstanten (zur Speichererspanis)
Sichtbarkeit für Methoden und Variablen	`private`	nur in der eigenen Klasse sichtbar (lokal)
	`public`	überall sichtbar (global)
Klassenaufruf mit new	`Fraction f1;`	
	`f1 = new Fraction (1,2);`	

Konstruktor	`Fraction(int n, int z){`
	`  //Anweisung`
	`}`
		werden beim Anleigen mit new … durchlaufen
		Zweck: Anfangswerte setzen
		Bedingung: Name gleich wie Klassenname

Typenabfrage	`f1 instanceof Fraction`	Ist f1 ein Fraction

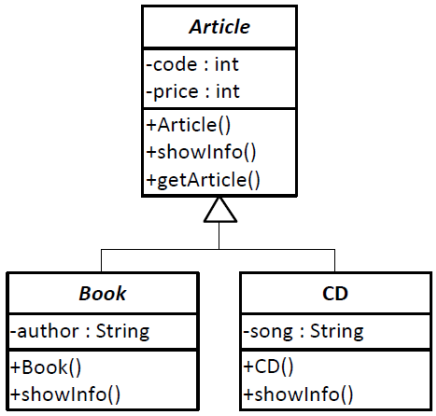
Klassifikation (Vererbung)

Idee
Gemeinsamkeiten zu teilen
Alle Felder und Methoden werden geerbt
Erweiterbarkeit mit neuen Unterklassen

Umsetzung
Unterklasse extends Oberklasse

Methoden überschreiben
showInfo wird überschrieben
getArticle wird unverändert übernommen

super-Aufrufe
Aufruf einer Methode der Oberklasse
Aufruf des Konstruktors der Oberklasse




```
public class Article {
   int code, price;
   public Aritcle(int code, int price){...}
   public String showInfo() {
      System.out.print(code + „, „ + price;
   }
   public void getArticle(){...}
}

public class Book extends Article{
   String autor, title;
   public Book(){
      this.code = 315;
   }
   public void showInfo() {
      super.showInfo();
   }
}

public class CD extends Article{
   String song;
   public CD(int code, int price, ...){
      super(code, price);
      this.song = song;
   }
   public void showInfo(){...}
}
```

Kompatibilität zwischen Ober- und Unterklassen

Zuweisungen
- statischer Typ -> Article
- dynamischer Typ -> Book, CD

```
Article a = new Book();
Article a = new CD();
a.title ="..";        //Fehler, weil a: Article
((Book)a).title="x"; //kein Fehler, a: Buch
```

Prüfung auf Referenz (Ist-Beziehung)

```
if (a instanceof Book){...}
```

Typenumwandlung

```
Book b = (Book) a;   //sofern instanceof gemacht
```

Dynamische Bindung

Aufruf immer der dynamischen Bindung

```
a.showInfo(); //-> a von Book
```

Abstrakte Klassen (vorgegebene Methoden)

- Unimplementiert in der Oberklasse
- Methode müssen in den Unterklassen geschrieben werden
- Abstrakte Klassen können nicht instanziert werden
- Abstakte Methode -> Abstrakte Klasse

```
abstract class Animal {
   abstract void speak();          Abstrakte Klasse
}
class Bird extends Animal{
   void speak(){...};              Konkrete Klasse
}
```

Interfaces (Schnittstellen)

Klassen aus Methodenköpfen
spezielle abstrakte Klassen
Mehrfachverwendung möglich
Gleichbehandlung von Klassen, die nicht in Beziehung zueinander stehen

```
interface Writer {
   void open();
   void write(char c) }
class Textbox implements Writer, Reader{...}
```

Wrapper-Klassen und Boxing

Grund	Basistypen sind keine Klassen								
Lösung	Wrapper-Typen (Basistypen kompatibel zu Object machen)								
Verwendung	`List.add(new Integer(5));`						**Autoboxing (seit Java 5)** `List.add(5);`		
	`int value = ((Integer)list.get(0)).intValue();`						**Autounboxing** `int value = list.get(0);`		
Typen	**Datentyp**	boolean	char	byte	short	int	long	float	double
	Klasse	Boolean	Character	Byte	Short	Integer	Long	Float	Double

Generizität

Prinzip: Typenplatzhalter T

T wird durch noch unbekannten Datentyp ersetzt

```
class List<T> {
   T[] data;
   void add(T x){…}
   T remove();
}
```

```
class SortedListe<T extends Comparable<T>>{
   T[] data = (T[]) new Comparable[100];
   void add(T elem) {… elem.compareTo() …}
   T remove();
}
```

Generische Methode

```
public static <T extends Comparable<T>> T max(T[] a){ ... }
```

Mehrerer Parameter

```
class List<T, V>{…}
```

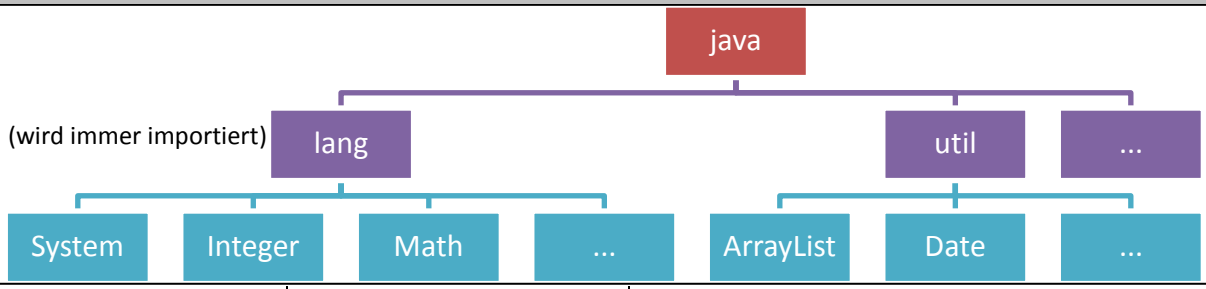
Arraydeklaration

```
T[] data = new T[100];                 //Fehler
T[] data = (T[]) new Object[100];      //kein Fehler
```

Verwendung
Typsicherheit
Vermeiden von Laufzeitfehler
Weniger Typenumwandlungen

```
List<String> list = new List<String>();
list.add(„abc“);
```

Packete



Packete	Import	Dokumentation
package ch.ntb.* Ziel: Ordnung schaffen	**import** ch.ntb.*	• hinkt immer etwas hinterher • /** … **/ • Doku automatisch erstellbar

Exceptions (Ausnahmen)

Fehlercodes
Jede Funktion liefert einen Resultatwert
`f() {…}`

Ausnahmebehandlung
- Ausnahmebehandlung vom Normalablauf trennen
- Führt nicht zu Systemabbruch

Systemausnahmen (vordefiniert)
- von jvm: java virtual machine

Benutzerausnahmen (vordefiniert)
- vom Benutzer mit **throw** geworfen,

Eigene Exception

Exception Handler (fangen mit catch)
Reagieren auf Ausnahmen
Der erste passende Handler wird gewählt
> deshalb: Reihenfolge wichtig

finally (wird immer ausgeführt)
z.B. schliessen von Dateien und Verbindungen

Weiterwerfen
Ausnahmen behandeln oder mit **throws** weiterwerfen

```
try{
   f();
   if(i==n) throw new OverflowException(i);
   g() throws OverflowException{
   }
}catch(IndexOutofBoundsException ex){
   Out.println(„Indexfehler“);
   ex.printStackTrace();
}catch (NullPointerexception ex)
   Out.print(„…“);
}
finally{…}

class OverflowException extends Exception {
   int info;
   OverflowException(int info) {
      this.info = info;
   }
}
```

Runtime-Exceptions

```
ArithmeticException                 //Division durch 0
NullPointerException                //Zeiger hat Wert null
ArrayIndexOutOfBoundsException      //Arraygrösse überschritten
```

Threads („Parallele Prozesse“)

- Threads teilen sich den Speicher
- Programme laufen quasi-parallel

Erzeugen von Threads
Von **Thread abgeleitet**
run implementieren
mit **start**() starten
mit **sleep**(int delay), Zeit abwarten [ms]

```
public class Prozess_1 extends Thread{
   public static void main(String[] args) {
      new Prozess_1().start(); new Prozess_2().start();
   }
   public void run(){
      int n = 0, delay = 500;
      while(n<10){
         Out.print(n+" ");
         try{ sleep(delay); n++;
         }catch(InterruptedException e){};
} } }
public class Prozess_2 extends Thread{
   public void run(){...}
}
```

Synchronisation
Threads können sich in die Quere kommen
Grund: Unterbruch im kritischen Bereich
Lösung: Sperrverfahren

```
a) Objekt als Schlüssel / Schloss (lock)
   int balance; Object lock = new Object();
   void deposit(int x){
      synchronized(lock){ balance=balance+x; }
   } //dasselbe beim withdraw
b) synchronized-Methode (wenn ganze Methode kritisch)
   int balance;
   synchronized void deposit(int x){ balance=balance+x;}
   //dasselbe beim withdraw
```

wait & notify
... lösen Blockaden auf

```
notify() //weckt irgendein Thread auf
wait()   //Legt Thread in Warteschlange, gibt lock frei
notifyAll() //weckt alle Threads auf
```

Softwaretechnik

Idee: gut lesbare, sinnvolle, verständliche Programme/Klassen

Vorgehen
Statik:

1.	Kandidaten für Klassen suchen	Nach Substantiven durchsuchen
2.	Klassen in UML Diagramm zeichnen	Nur Klassennamen schreiben
3.	Einteilen in Klassen und Attribute	einfach=Attribut, komplex=Klasse wer kennt wen/wer besteht aus wem
4.	Beziehungen zwischen Klassen	Beziehungen in UML aufzeichnen

Dynamik:

5.	Objektdiagramm	Situation zu einem gewissen Zeitpunkt
6.	Sequenzdiagramm	Zeitlicher Ablauf

Streams

Ein I/O Stream repräsentiert eine Quelle oder ein Ziel
Streams unterstützen viele Datentypen
Typen

	Bytes (8-bit)	Zeichen (characters)
Eingang	InputStream	Reader
Ausgang	OutputStream	Writer

Klasse System
3 Streams werden automatisch erzeugt:
System.in; System.out; Systen.err;

```
import java.io.*;
public class FileTest {
   public static void main() {
      try {
         FileWriter out = new FileWriter("log.txt");
         BufferedWriter b = new BufferedWriter(out);
         PrintWriter p = new PrintWriter(b);
         p.println("This is the first sentence");
         p.close();
      } catch (Exception e) { }
   }
}
```

Pipes
Kommunikation zwischen verschiedenen Programmen oder Threads

```
PipedOutputStream pos = new PipedOutputStream();
PipedInputStream pis = new PipedInputStream();
pos.connect( pis ); // oder pis.connect( pos );
```

Link – Spechen sie Java