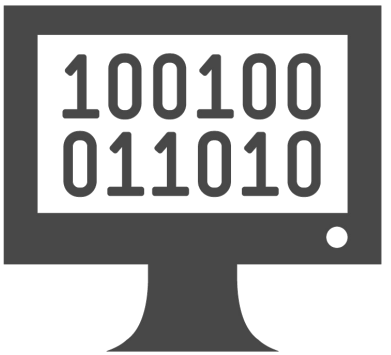
Fortgeschrittene Programmierungsmethoden



Zsolt Balo

Dr. Cătălin Rusu

Workload (in Stunden):

Vorlesung: 2
Seminar/Labor: 1 + 2

URL: http://www.cs.ubbcluj.ro/~rusu/fp
Email: rusu@cs.ubbcluj.ro

Kursinhalt

1. Überblick, grundlegende Sprachelemente

2. Objektorientierte Programmierung (OOP)
   OOP is an island

3. JAVA

4. Analyse und Design von Softwaresysteme

Kursinhalt

1. Einführung in Java
2. Klassen und Objekte Java
3. Abstrakte Klassen
4. Pakete
5. Bibliotheken
6. GUI und GRASP Schablone
7. Observer Schablon
8. Factory patterns. Dependency Injection.
9. Reflektion

Prüfungsform:

Klausur (20%)

Zwischenprüfung (20%)

Lab (30%)

Praktische Prüfung (30%)

Minimale Leistungsstandards:

K,Z,P,L >= 5

Algorithmen und Programme

- Allgemeines Verfahren zur Lösung einer Klasse von Problemen, das durch eine eindeutige Vorschrift so genau festgelegt ist, dass man es anwenden kann, ohne es verstanden zu haben.
- Eigenschaften:
  - Jeder Einzelschritt ist eindeutig festgelegt und berechenbar.
  - Das Verfahren liefert nach endlich vielen Schritten eine Lösung.
- Beispiele:
  - Modellbau: Montageanleitung
  - Informatik: Sortieralgorithmus

Programme

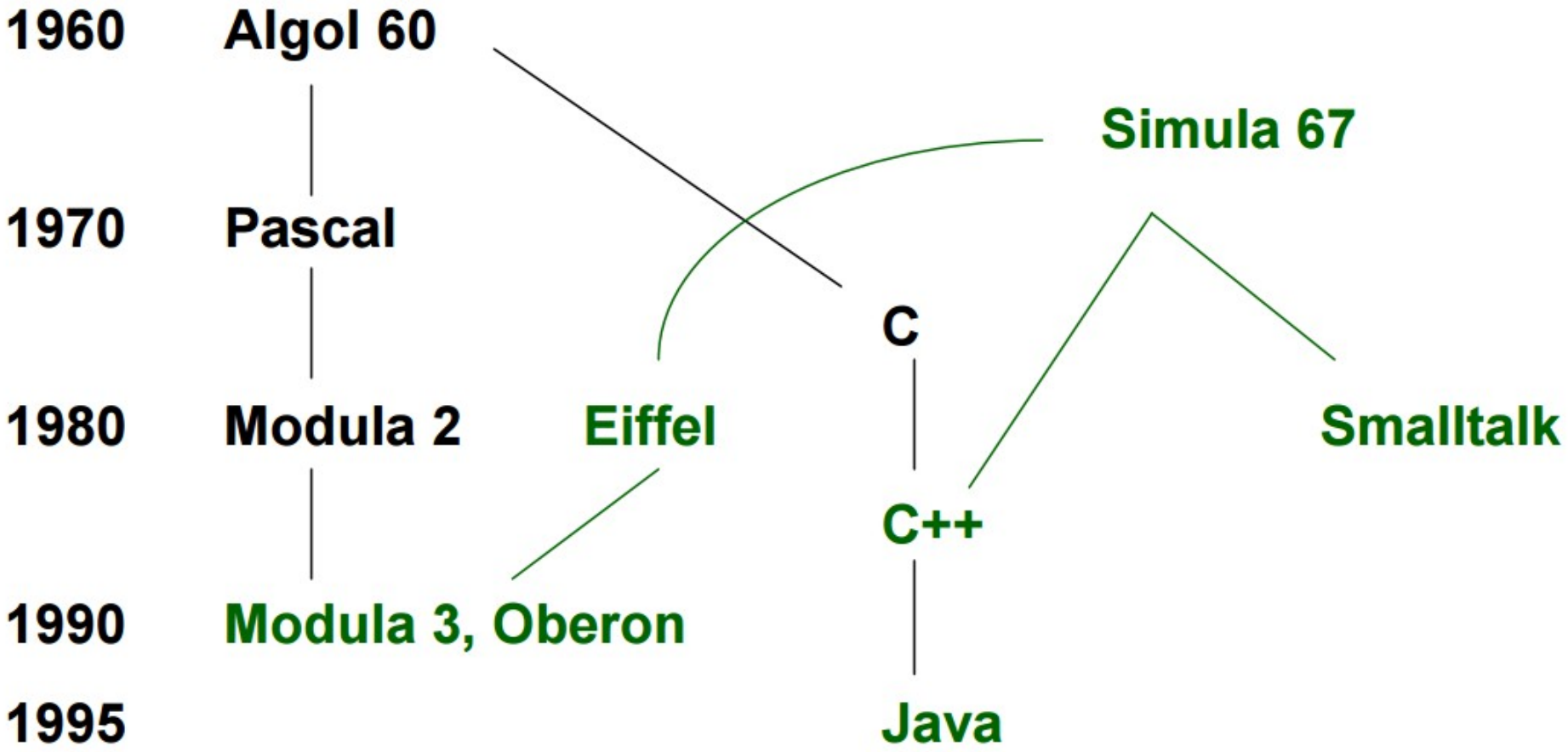
Programm

- Beschreibung von Datenstrukturen und Algorithmen in einer Programmiersprache

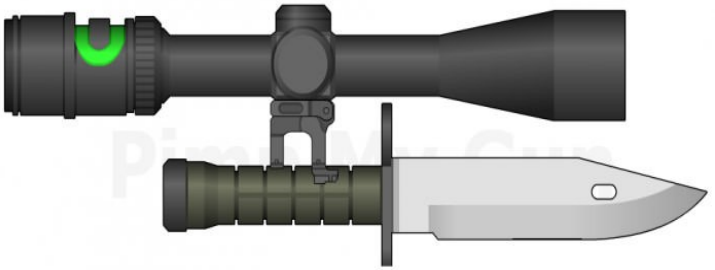
Software-Entwicklung

- Systematische Konstruktion von Programmen zur Lösung eines in der realen Welt gestellten Problems

Entwicklung objektorientierter Programmiersprachen



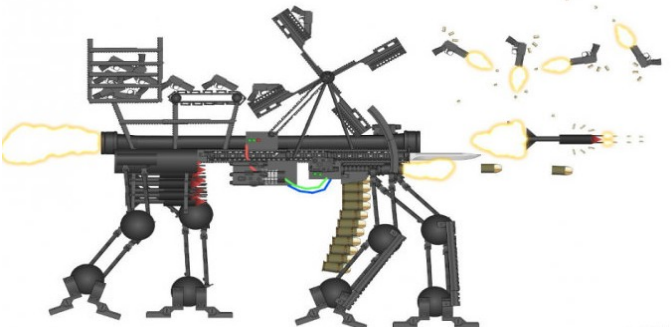
Assembly



C

C++

Python




LEARN JAVA, THEY SAID
IT'LL BE FUN, THEY SAID
quickmeme.com

Einstieg in die Programmierung mit Java

Installation und Vorbereitung

- Download von der Sun-Seite und Installation

  http://www.oracle.com/technetwork/java/index.html
- Entwicklungsumgebung für Java
  - *Notepad++*
  - *Eclipse, NetBeans, IntelliJ, Atom*

Aspekte von Java

- Objektorientiert: Klassenkonzept, strenge Typisierung
- Unabhängig von Plattform: Durch Übersetzung in Virtuelle Maschine (JVM)
- Netzwerkfähig, nebenläufig
- Sicherheitskonzept

Aspekte von Java

Nachteile:

- Laufzeithandicap durch Interpretation der JVM

Vorteile:

- Verteilte Anwendungen, Web-Anwendungen
- Rechnerunabhängigkeit von Graphikanwendungen

Grober Aufbau eines Java-Programms

- Ein Java-Programm besteht aus
  - einer Menge von Klassen und Schnittstellen
- Eine Klasse besteht aus
  - **Attributen:** Beschreiben den Zustand eines Objekts
  - **Methoden:** Beschreiben die Operationen, die ein Objekt ausführen kann
  - **Konstruktoren:** Operationen zur Erzeugung von Objekten

Beispiel

```
public class Hallo {
    public static void main(String[] args) {
    System.out.println(„Hallo!“);
    }
}
```

Methodenaufruf (allgemein):

```
    object.methodName(actual parameters);
```

Beispiel:

```
    System.out.println(„Hallo!“);
```

Übersetzung und Ausführung von Java-Programmen

Übersetzung in Bytecode

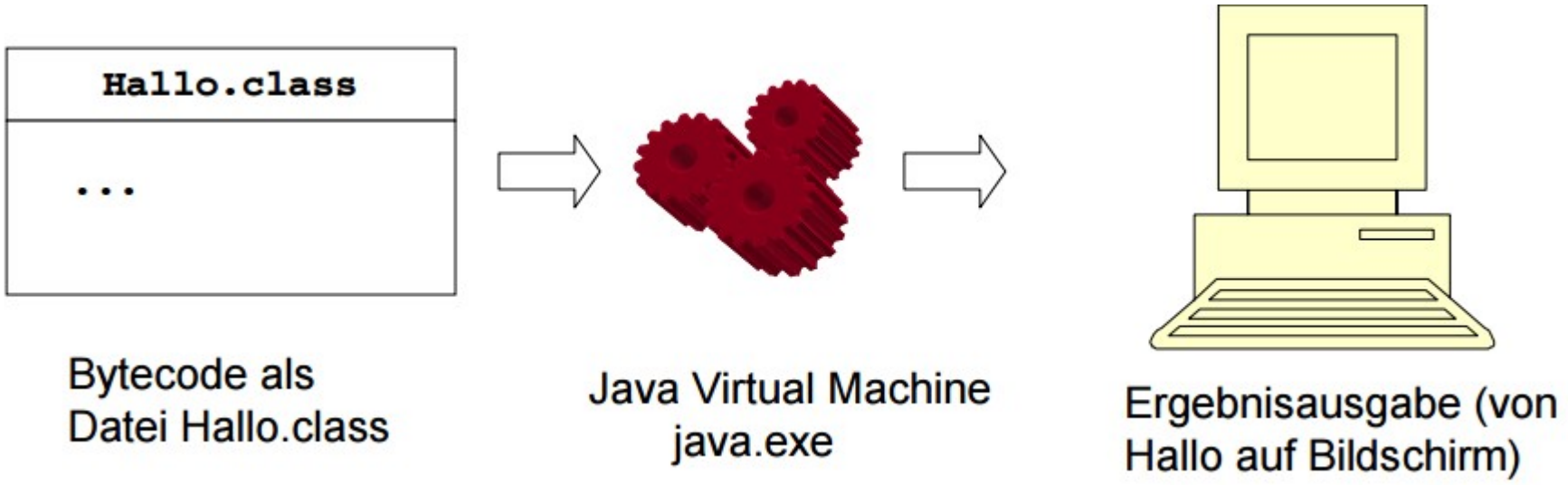
- Aus einer Textdatei mit Endung „.java“ erzeugt der Compiler javac eine

Datei mit gleichem Namen, aber Endung „.class“

- Diese enthält den Bytecode für die JVM

Übersetzung und Ausführung von Java-Programmen

Die Datei mit dem Bytecode wird der JVM übergeben und von der JVM ausgeführt (d.h. interpretiert).



Übersetzung und Ausführung

Übersetzung von Hallo.java:

cat@darkstar:~$ javac Hallo.java

Interpretation von Hallo.class:

cat@darkstar:~$ java Hallo

Gibt auf Bildschirm zurück:

Hallo!

Primitive Datentypen

- Wahrheitswerte

  boolean
- Zeichen, Symbole

  char
- Zahlen

  byte, short, int, long, float, double

Primitive Datentypen

Datentyp	Wertebereich	BIT
boolean	true, false	8
char	0 bis 65.535	16
byte	-128 bis 127	8
short	32.768 bis 32.767	16
int	-2.147.483.648 bis 2.147.483.647	32
long	-9.223.372.036.854.775.808 bis 9.223.372.036.854.775.807	64
float	+/- 1,4E-45 bis +/- 3,4E+38	32
double	+/- 4,9E-324 bis +/-1,7E+308	64

Variablen und Konstanten

- Deklaration von Variablen:

  <Datentyp> <Name>;

- Beispiel:

```
boolean a;
```

- Zuweisung von Werten:

  <Datentyp> <Name> = <Wert>;

- Beispiel:

```
int b;
b = 7;
boolean a = true;
char c,d,e;
```

Variablen und Konstanten

- Beschränkungen für Variablenbezeichnungen:

  Variablen beginnen mit einem Buchstaben oder Unterstrich (nicht erlaubt sind dabei Zahlen). Nach dem ersten Zeichen dürfen aber sowohl Buchstaben als auch Zahlen folgen. Operatoren und Schlüsselwörter dürfen nicht als Variablennamen verwendet werden

- Reservierte Schlüsselwörter:

  abstract, assert, boolean, break, byte, case, catch, char, class, const, continue, default, do, double, else, enum, extends, false, final, finally, float, for, future, generic, goto, if, implements, import, inner, instanceof, int, interface, long, native, new, null, operator, outer, package, private, protected, public, rest, return, short, static, strictfp, super, switch, synchronized, this, throw, throws, transient, true, try, var, void, volatile, while

Variablen und Konstanten

- Konstanten:

  Variablen, die während des Progammablaufs unverändert bleiben sollen, deklariert man als Konstanten

- Beispiel pi als Variable:

```
double pi = 3.14159;
```

- Deklaration von Konstanten:

  final <Datentyp> <NAME>;

- Beispiel PI als Konstante

```
final double pi = 3.14159;
```

Kommentare

1. `// entire line`
2. `/* multiple`
   `lines */`
3. `/** used by documentation Javadoc tool`
   `multiple`
   `lines*/`

```
 /* … /*
 */ … */   NOT OK!
```

```
/* …
   //
   //
*/   OK!!
```

Arrays und Matrizen

- Erzeugen eines int-Arrays mit k Elementen:

  <Datentyp>[] <name>;

  <name> = new <Datentyp>[k];

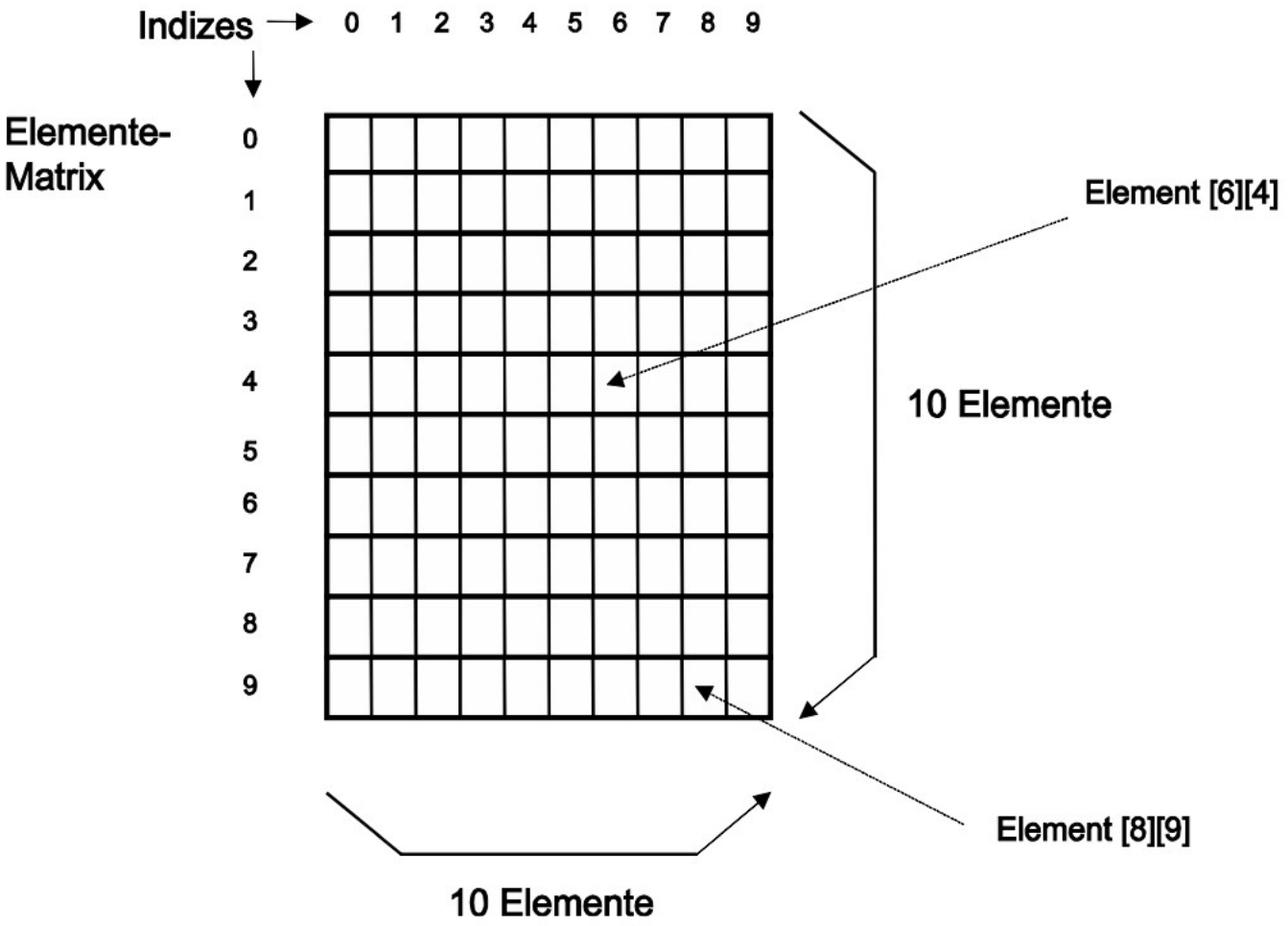
- Oder in einer Zeile:

  <Datentyp>[] <name> = new <Datentyp>[k];

- Zugriff auf die Elemente und Initialisierung der Variablen:

```
int[] a = new int[2];
a[0] = 3;
a[1] = 4;
```

Arrays und Matrizen



Arrays und Matrizen

- Wir erzeugen Matrizen, indem wir zum Array eine Dimension dazu nehmen:

```
int[][] a = new int[n][m];
a[4][1] = 27;
```

- Auf diese Weise können wir sogar noch mehr Dimensionen erzeugen:

```
int[][][][] a = new int[k][l][m][n];
```

Primitive Datentypen und Ihre Operationen

- char

  Bezeichnet ein Zeichen oder Symbol.

```
char d ='7';
char e = 'b';
```

Relationale Operatoren (Vergleichsoperatoren):

```
boolean d, e, f, g;
char a, b, c;
a = '1';
b = '1';
c = '5';
d = a == b;
e = a != b;
f = a < c;
g = c >= b;
```

Primitive Datentypen und Ihre Operationen

- Int

  Der ganzzahlige Datentyp int wird wahrscheinlich von allen am häufigsten verwendet.

```
int a, b = 0;
a = 10;
```

  Der kleinste und größte darstellbare Wert existiert als Konstante

```
int minimalerWert = Integer.MIN_VALUE;
int maximalerWert = Integer.MAX_VALUE;
```

Primitive Datentypen und Ihre Operationen

- Int

```
int a = 29, b, c;
b = a/10;
c = a%10;

int d=0, e=1;
d = d + e;
e = e - 5;
d = d + 1;
```

```
int d=0, e=1;
d += e;
e -= 5;
d += 1;
d++;
```

Primitive Datentypen und Ihre Operationen

- float, double

  Repräsentieren gebrochene Zahlen oder Gleitkommazahlen

  5. 4.3 .00000001 -2.87

  Java verwendet eine wissenschaftliche Darstellung für Gleitkommazahlen, um längere Zahlen besser lesen zu können

  1E-8

Primitive Datentypen und Ihre Operationen

- float, double

  Die möglichen Operationen sind die gleichen, wie sie bei int vorgestellt wurden, lediglich die beiden Teilungsoperatoren verhalten sich anders.

```
double a = 2;
double b = 2.2;
float c = 3;
float d = 3.3f;

float e, f;
e = d%c;
e = d/c;

f = c*(d/2.4f)+1;
f += 1.3f;
```

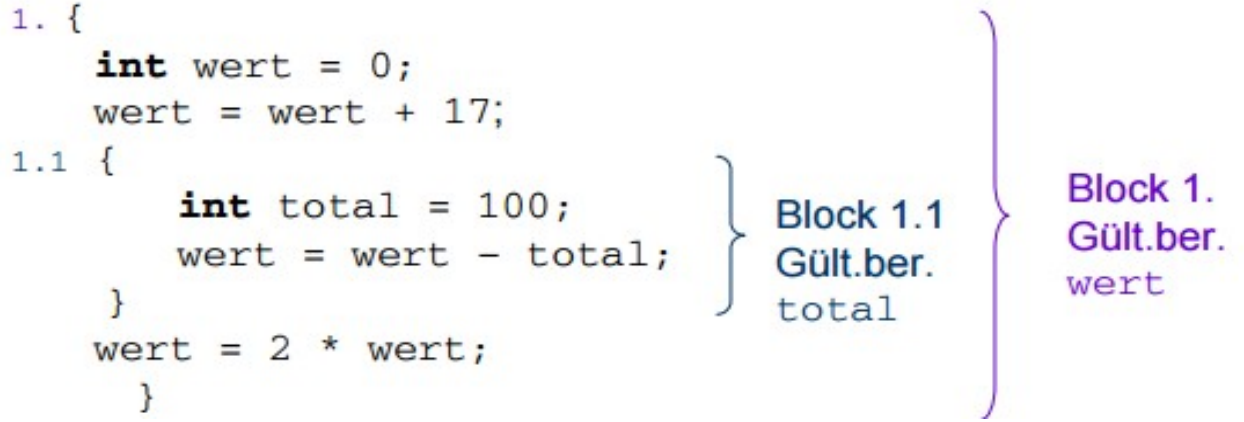
Sequentielle Anweisungen

Programm Sequentiell.java mit sequentiellen Anweisungen

```java
public class Sequentiell{
  public static void main(String[] args){

    int a=5; // Anweisung 1
    a=a*2; // Anweisung 2
    a=a+10; // Anweisung 3
    a=a-5; // Anweisung 4

    System.out.println("a hat den Wert: "+a);

  }
}
```

Gültigkeitsbereich

- Der Gültigkeitsbereich einer lokalen Variablen oder Konstante ist der die Deklaration umfassende Block
- Außerhalb dieses Blocks existiert die Variable nicht!



Verzweigungen

- Wenn eine Bedingung erfüllt ist, dann führe eine einzelne Anweisung aus:

```
if (<Bedingung>) <Anweisung>;


if (<Bedingung>) {
    <Anweisung_1>;
    <Anweisung_2>;
    …
    <Anweisung_n>;
}


if (<Bedingung>) <Anweisung_1>;
else <Anweisung_2>;
```

Verzweigungen

- Mehrfachverzweigungen lassen sich mit switch realisieren

```
switch (<Ausdruck>) {
    case <Konstante1>:
        <Anweisung1>;
    break;
    case <Konstante2>:
        <Anweisung2>;
    break;
    default:
        <Anweisung3>;
}
```

Verzweigungen

- Beispiel für switch-Verzweigung:

```
for (int i=0; i<5; i++){
  switch(i){
    case 0:
      System.out.println("0");
    Break;

    case 1:
    case 2:
      System.out.println("1 oder 2");
    break;

    case 3:
      System.out.println("3");
    break;

    default:
      System.out.println("hier landen alle anderen...");
    break;
  }
}
```

Verschiedene Schleifentypen

- Warum Schleifen?

```
System.out.println("1 zum Quadrat ist "+(1*1));
System.out.println("2 zum Quadrat ist "+(2*2));
System.out.println("3 zum Quadrat ist "+(3*3));
System.out.println("4 zum Quadrat ist "+(4*4));
System.out.println("5 zum Quadrat ist "+(5*5));
System.out.println("6 zum Quadrat ist "+(6*6));
```

for (<Startwert>; <Bedingung>; <Schrittweite>)

<Anweisung>;

```
for (int i=1; i<=1000; i=i+1){
  System.out.println(i+" zum Quadrat ist "+(i*i));
}
```

Verschiedene Schleifentypen

- While-Schleifen, wenn nicht bekannt ist, wieviele Durchläufe benötigt werden. Führe die Anweisungen solange aus, wie die Bedingung wahr ist:

  while (<Bedingung>)

    <Anweisung>;

```
int i=1;
while (i<=1000){
  System.out.println(i+" zum Quadrat ist "+(i*i));
  i=i+1;
}
```

Verschiedene Schleifentypen

- Do-While-Schleifen, wenn die Anweisungen mindestens einmal ausgeführt werden sollen

do {

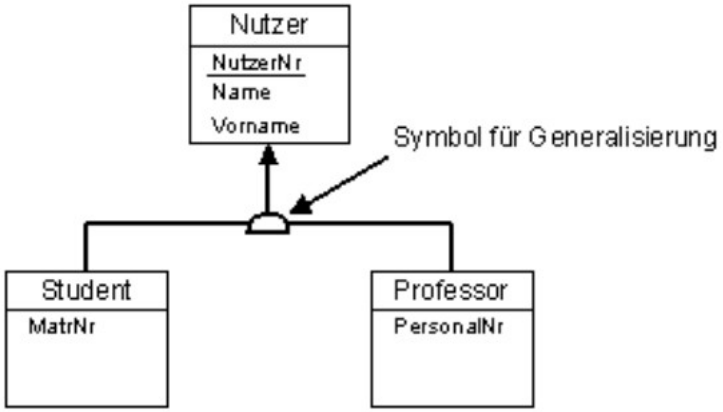
<Anweisung>;

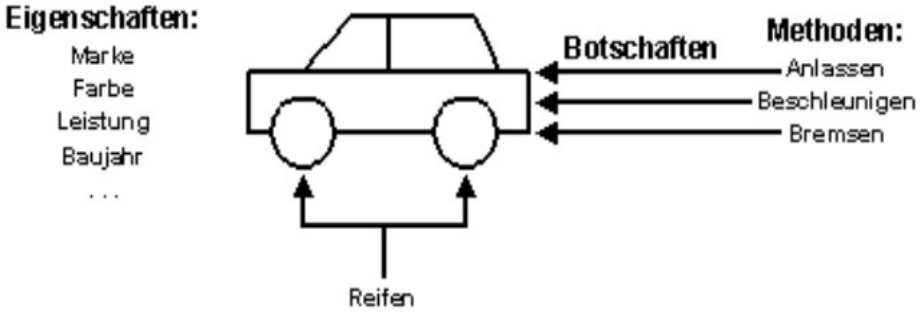
} while (<Bedingung>) ;

```
int i=0;
do{
  i++;
  System.out.println("Wert von i: "+i);
} while (i<5);
```

Objektorientierung

- Autos und Autoteile
- Professoren und Studenten
- ...





ein abstraktes Modell bzw. einen Bauplan für eine Reihe von ähnlichen Objekten

Beispiel

- Welche Eigenschaften hat jeder Würfel?
  - Farbe
  - Kantenlänge
- Welche Methoden hat jeder Würfel?
  - Drehen
  - Einfärben
  - Zeichnen

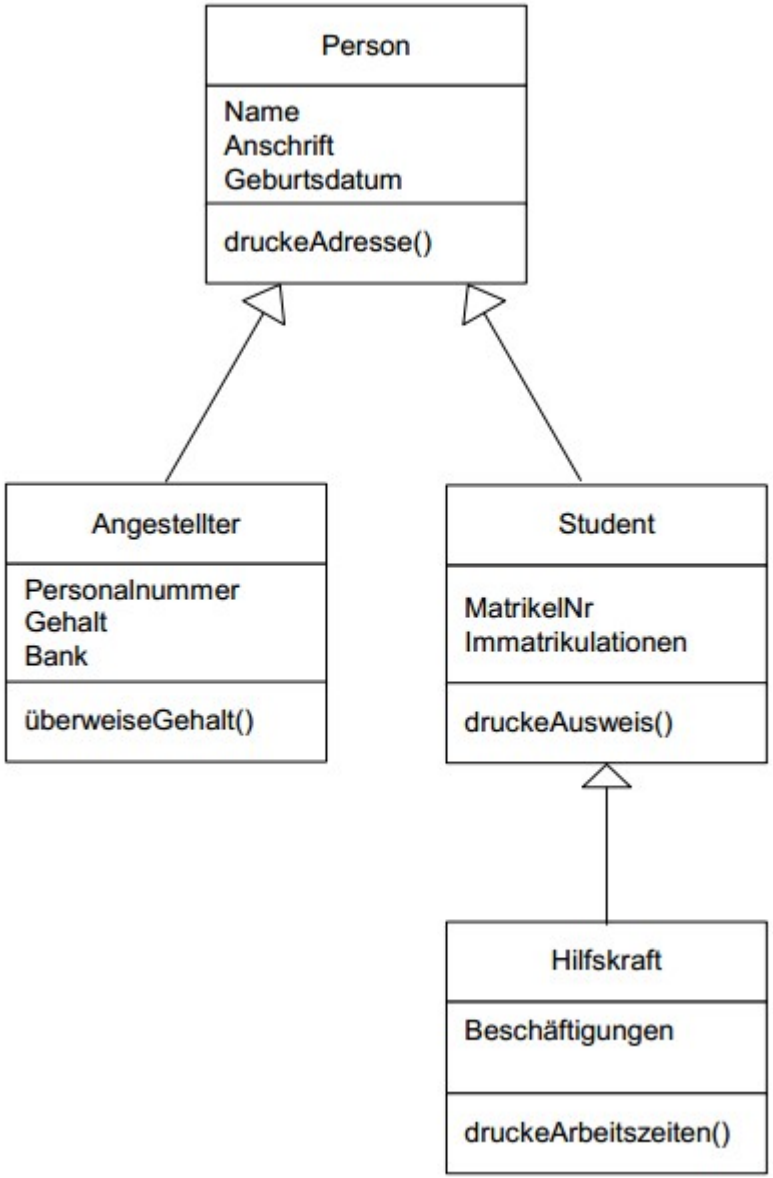
Generalisierung – Vererbung – Die Ist-Beziehung

- beschreibt eine Beziehung zwischen einer allgemeinen Klasse (Basisklasse) und einer speziellen Klasse
- die spezialisierte Klasse ist vollständig konsistent mit der Basisklasse, enthält aber zusätzliche Informationen (Attribute, Methoden, Assoziationen)
- ein Objekt der spezialisierten Klasse kann überall dort verwendet werden, wo ein Objekt der Basisklasse erlaubt ist

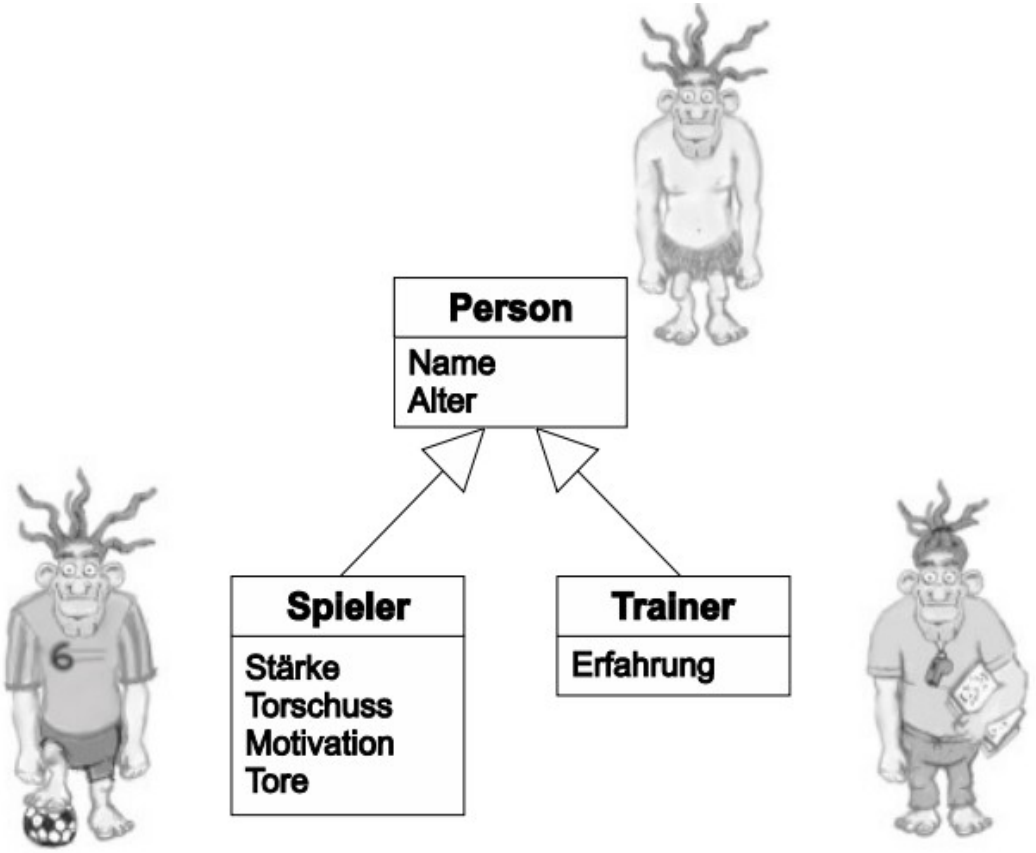
Generalisierung – Vererbung – Die Ist-Beziehung

In der Biologie wird in Form von Taxonomien mit Generalisierungsstrukturen gearbeitet

beispielsweise gliedert sich das Reich der Tiere in die beiden Unterreiche der Einzeller und Vielzeller

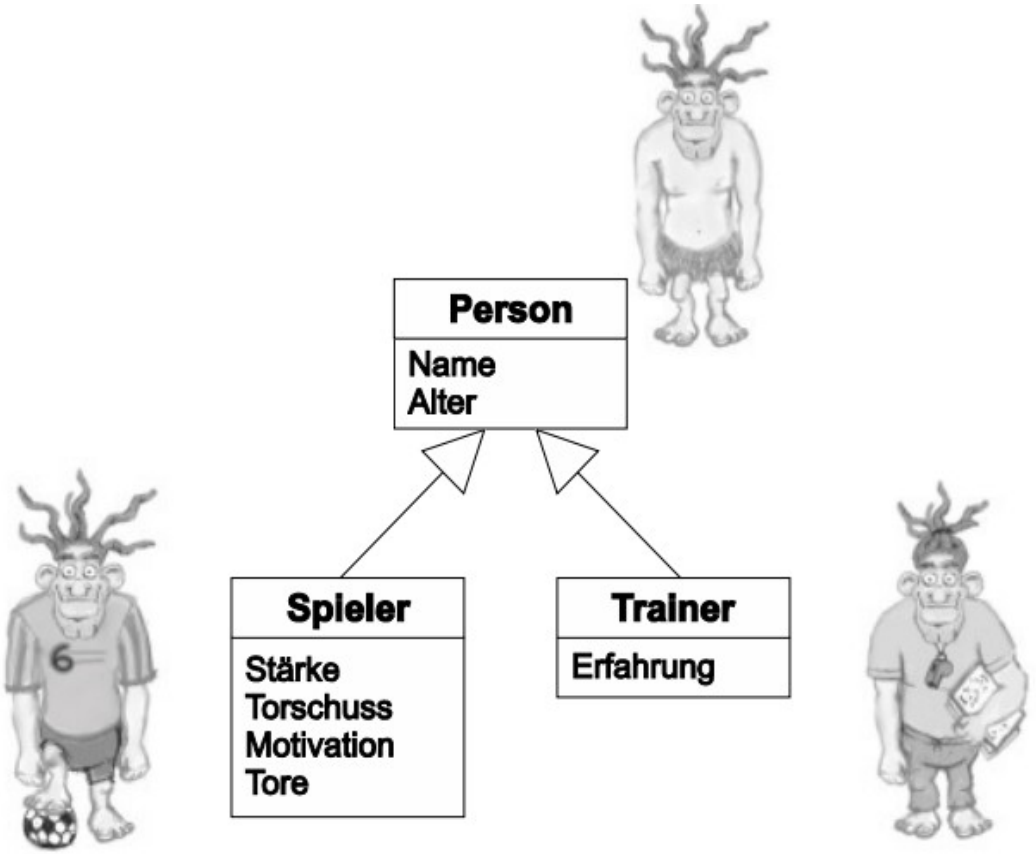


Generalisierung – Vererbung – Die Ist-Beziehung




```java
public class Person{
  // Eigenschaften einer Person:
  public String name;
  public int alter;
}
```

Generalisierung – Vererbung – Die Ist-Beziehung




```java
public class Spieler extends Person{
  // Zusätzliche Eigenschaften eines Spielers:
  public int staerke; // von 1 (schlecht) bis 10 (super)
  public int torschuss; // von 1 (schlecht) bis 10 (super)
  public int motivation; // von 1 (schlecht) bis 10 (super)
  public int tore;
}
```

Methoden

- Objekte interagieren miteinander durch das gegenseitige Aufrufen von Methoden (=Nachrichtenaustausch).
- Eine Methode (Nachricht) besteht aus zwei Teilen:
  - der Signatur (bzw. Methodendeklaration), die Angaben über Sichtbarkeit, Rückgabetyp, Name der Methode und Parameter macht.
  - dem Methodenrumpf, in dem die Deklarationen der lokalen Variablen und die eigentlichen Anweisungen stehen.

Methoden

```
class Point {
  public double x, y;
  //Methodendeklaration:
  public double distance(Point pkt) {
    double xdiff = x - pkt.x;
    double ydiff = y - pkt.y;
    return Math.sqrt(xdiff*xdiff +ydiff*ydiff);
}
...
//Methodenaufruf:
Class PointTester {
  public static void main(String[] args) {
  Point lowerLeft = new Point(); lowerLeft.x = 0.1; …
  Point upperRigth = new Point(); upperRight.x = 0.1; …
  double d = lowerLeft.distance(upperRight);
}
```

Standardvariable „this“

- this ist eine Referenz zum aktuellen Objekt, oder anders ausgedrückt this ist eine Referenz auf die aktuelle Instanz der Klasse in der man sich gerade befindet.
- Über this kann auf alle Variablen und Methoden der Instanz zugegriffen werden.

Statische Elemente

- Variablen und Methoden, die nicht zu einer bestimmten Instanz sonder zur Klasse gehören.
- Statische Variablen/Methoden sind auch dann verfügbar, wenn noch keine Instanz der Klasse erzeugt wurde.
- Statische Variablen/Methoden können über den Klassennamen aufgerufen werden.
- Deklaration durch das Schlüsselwort: static

```
class Point {
  double x, y;
  static int count;
}
```

Konstruktoren

- Jede Klasse benötigt einen oder mehrere Konstruktoren.
- reservieren den Speicherplatz für eine neu zu erzeugende Instanz,
- weisen den Instanzvariablen initiale Werte zu,
- haben denselben Namen wie die Klasse,
- werden wie Methoden deklariert, aber ohne Rückgabewert

Konstruktoren

- Wenn kein Konstruktor erstellt wurde, wird von Java default-mäßig ein Konstruktor (ohne Parameter) zur Verfügung gestellt.

```
class Student {
  private String matrNr;
  private String name;
  private int semester;

  Student(String name, String matrNr) {
    this.name = name;
    this.matrNr = matrNr;
  }
}
```

Spezialisierte Konstruktoren

```java
class Student {
  private String name, matrNr;
  private int semester;

  Student(String studName, String studMatrNr) {
    //Konstruktor 1
    name = studName;
    matrNr = studMatrNr;
  }

  Student(String name, String matrNr, int semester) {
    // Konstruktor 2
    this(name, matrNr); //Aufruf Konstruktor 1
    this.semester=semester;
  }
}
```

Zugriffskontrolle

- In JAVA gibt es drei Attribute und eine Standardeinstellung, die den Gültigkeitsbereich von Klassen, Variablen und Methoden regeln, d.h. festlegen, ob bzw. welche anderen Objekte auf eine Klasse, Variable oder Methode der programmierten Klasse zugreifen können:
  - private
  - protected
  - public

JUnit

- Motivation
- Extreme Programming
- Test-First
- Das Framework
  - Grundlagen
  - Assert
  - TestCase

JUnit

- Motivation
- Extreme Programming
- Test-First
- Das Framework
  - Grundlagen
  - Assert
  - TestCase

Motivation

- JUnit: Open source Test-Framework
  - Schreiben und Ausführen automatischer Unit Tests unter Java
  - Aktuell: Version 4 (http://www.junit.org/)
  - In jeder gängigen Java-IDE verwendbar
- Autoren:
  - Kent Beck (Extreme Programming)
  - Erich Gamma
- Entsprechende Frameworks für gängige Programmiersprachen erhältlich

Extreme Programming (XP)

- Unit-Tests vor XP wenig in Praxis angewandt - in XP: zentrale Tätigkeit
- XP: agiler Softwareentwicklungsprozess
  - Schlüsselaktivität: Programmieren
  - für kleine bis mittelgroße Teams
  - außergewöhnlich stark durch den Kunden gesteuert
  - während Entwicklung Reaktion auf vage und rasch ändernde Anforderungen möglich
  - ermöglicht Erstellung langlebiger Software

Extreme Programming (XP)

- **Pair Programming + Gemeinsame Verantwortlichkeit**
- Programmierstandards festlegen und einhalten
- möglichst einfaches Design + häufiges Refactoring
- Ständige isolierte, automatisierte Unit-Tests
  - Unit-Test müssen zu 100% laufen
  - i.d.R. verbunden mit Test-First-Ansatz
- Kunde gehört zum Team
- Kurze Iterationszyklen

TestFirst-Ansatz

- Ziel:
  - Qualitätssicherung: Testbarkeit, Einfachheit
- Vorgehen:
  - Vor eigentlicher Codierung Test schreiben
  - Erst wenn Test fehlerfrei, dann ist Code fertig
  - Nur soviel Produktionscode wie Test verlangt
  - Kleine Schritte (abwechselnd Test- und Produktionscode)
  - Vor Integration in Gesamtsystem muss Unit Test erfolgreich sein

TestFirst-Ansatz

- Vorteile:
  - gesamter Code ist getestet (Zerstörung funktionierenden Codes sofort entdeckt)
  - Tests dokumentieren Code
  - Schnelles Feedback durch kurze Wechsel Erzeugung von Test- und Produktionscode
  - Einfaches Design, da durch Test bestimmtes Design
- Vorsicht:
  - triviale Testfälle

JUnit Beispiel

- public double sum(double a, double b)
- public double diff(double a, double b)
- public double mult(double a, double b)
- public double div(double a, double b)
- public void setMem(double a)
- public double getMem()
- public void clearMem()

JUnit Beispiel

```java
import static org.junit.Assert.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;


public class CalculatorTest {
  private Calculator calculator;

  @Before public void setUp() {
    calculator = new Calculator();
  }

  @Test public void testSum() {
  double a = 5.6, b = 6.5;
    assertEquals(12.1, calculator.sum(a, b), 0.1);
  }

  @Test(expected = IllegalArgumentException.class)
  public void testDiv() {
    double a = 3.0;
    double b = 0.0;
    calculator.div(a, b);
  }
}
```

References

1. Bruce Eckel, Thinking in Java

2. Java Tutorials from Oracle

   http://docs.oracle.com/javase/tutorial/