Java

Steilkurs

Software-Praktikum im Grundstudium
WS 2004/2005

Dipl.-Inform. Michael Kirchhof
Dipl.-Inform. Bodo Kraft
Prof. Dr.-Ing. Manfred Nagl

Department of Computer Science III
Software Engineering
Ahornstr. 55
52074 Aachen, Germany

http://www-i3.informatik.rwth-aachen.de

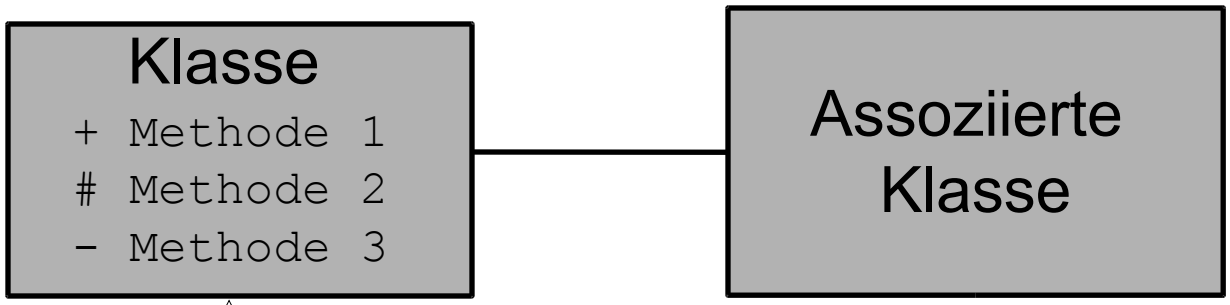
Gliederung

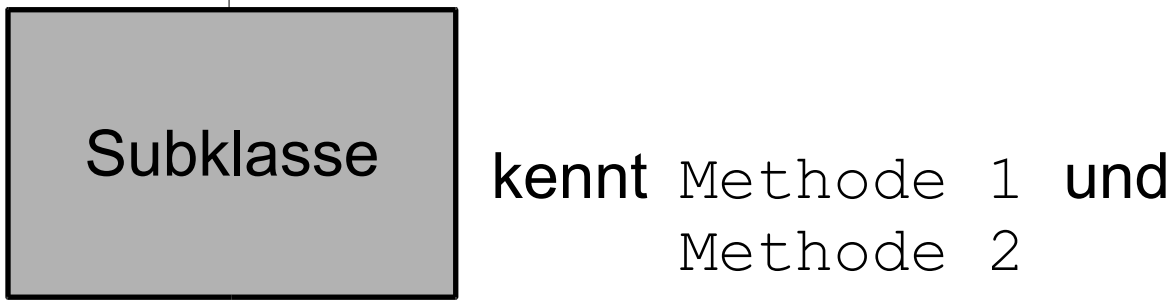
- Grundlagen der Objektorientierten Programmierung
- Einführung in die Programmiersprache Java
- Java auf UNIX-Rechnern verwenden
- Verfügbare Informationen (Literatur, URLs)

OO - Grundlagen

- **Objekte**
  - tauschen Nachrichten aus (rufen Methoden auf)
  - können Zustände haben (Attribute)
  - können Innereien verkapseln
- **Klassen**
  - gruppieren gleich strukturierte Objekte
  - definieren verfügbare Methoden und Attribute
- **Sichtbarkeiten:** `public`, `protected`, `package`, `private`
- **Implizites Verhalten**
  - Objekte werden durch Konstruktoren initialisiert
  - Vor Zerstörung wird Destruktor aufgerufen

Sichtbarkeiten

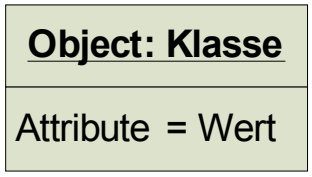




Klassen und Objekte

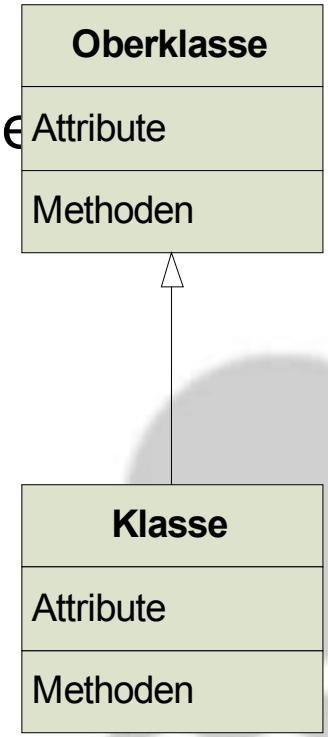
- **Klassen** stellen Schablonen für gleichartige Strukturen dar
- **Klassen** sind Konstrukt der Programmierung
- **Klassen** sind statisch
- i.A. können beliebig viele Objekte aus einer Klasse instantiiert werden.

- Ein **Objekt** beschreibt eine spezielle Ausprägung ein Klasse
- **Objekte** sind Konstrukte der Laufzeit.
- **Objekte** sind dynamisch und verändern ihren Zustand zur Laufzeit.
- Die Menge aller Objekte zu einem Zeitpunkt spiegelt den Zustand des Systems wieder.

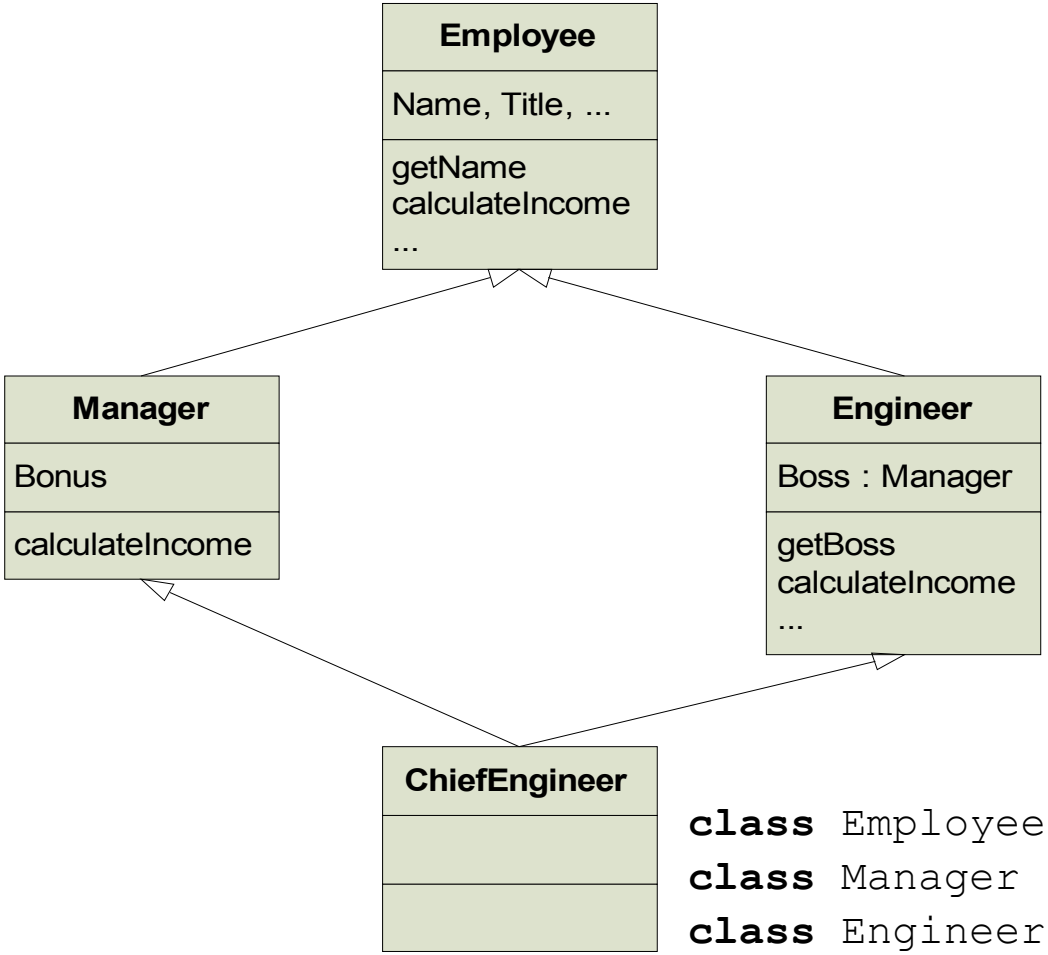


OO - Vererbung (1) - Grundkonzepte

- **Spezialisierung**
  Eine Oberklasse `A` wird verwendet und um we
  Funktionalität zu einer Klasse `B` erweitert.
- **Generalisierung**
  Gemeinsamkeiten einiger Klassen zur einer allgemeinen Oberklasse zusammenfassen.
- **Substitution**
  Wo Superklasse steht, darf Subklasse verwendet werden
- **Überschreiben**
  Methoden von Oberklassen können überschrieben werden

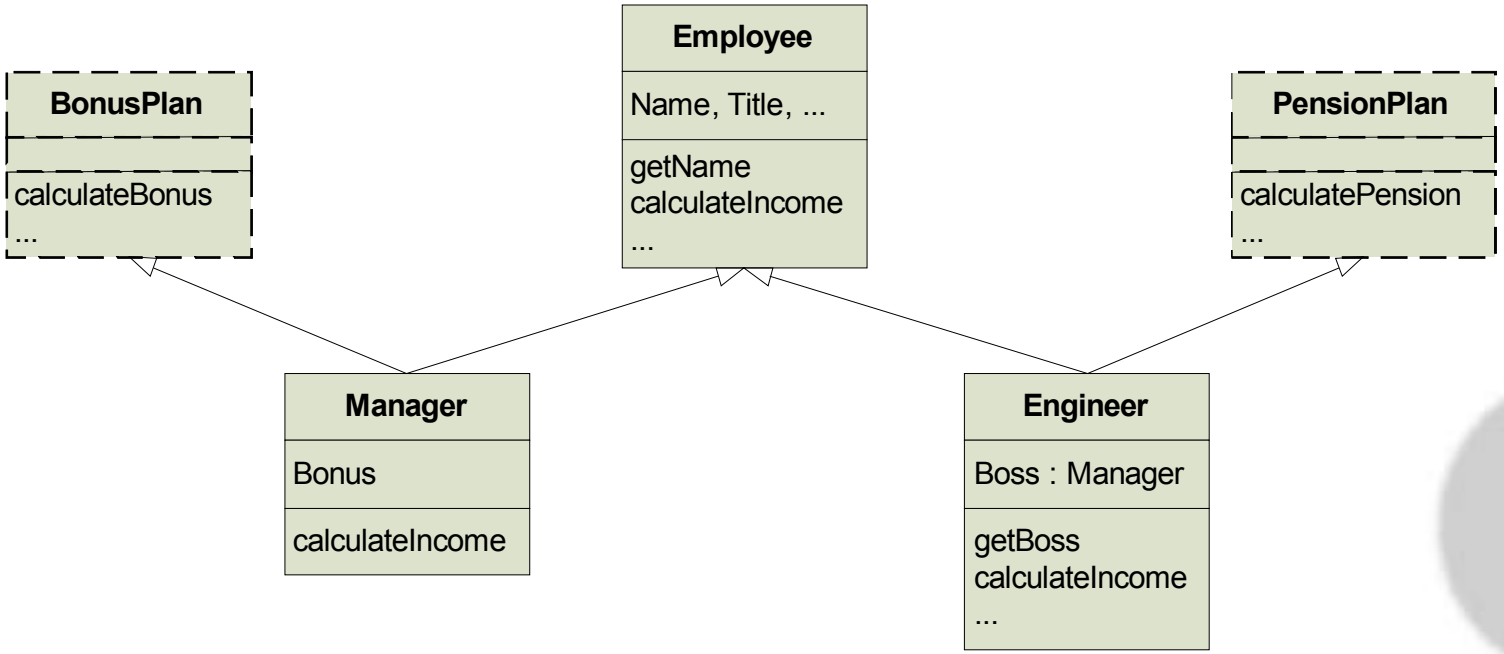


OO - Vererbung (2) - Beispiel




```
class Employee                          {...}
class Manager  extends Engineer {...}
class Engineer extends Employee {...}
class ChiefEng extends Engineer, Manager{...}
```

OO - Vererbung (3) - Schnittstellen



- Durch Seitenvererbung von Schnittstellen ist die Vereinigung der Methoden mehrerer Vorgänger'klassen' möglich.
- Java-Ersatz für Mehrfachvererbung

OO - Aggregation vs. Assoziation

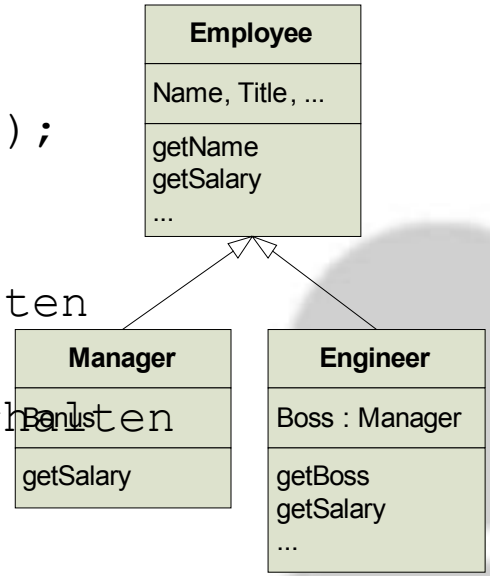
- **Assoziation** ein Objekt *kennt* andere Objekte
- **Aggregation** ein Objekt *enthält* andere Objekte
- Beispiel: Klasse `Employee` aggregiert `Title`, ist mit `Manager` assoziiert:

```
class Engineer {
    private Title title;
    private Manager boss;
    ...

    public void setTitle(Title t)    {title  = t.copy();};
    public void setBoss(Manager m)   {boss= m;        };
    ...
    public Title   getTitle()   {return title.copy();   };
    public Manager getBoss()    {return boss;         };
    ...
}
```

OO Programmierung - Polymorphie

- Auswahl einer Methode zur Laufzeit (Dynamisches Binden)
- Referenzierung von Objekten abgeleiteter Klassen durch Variablen/Referenzen einer Vorgängerklasse

```
Manager  aldritch = new Manager(...)
Engineer joe      = new Engineer(...);
Employee emp;

emp = aldritch ;     // Manager erhalten
emp.getSalary();     // Festgehalt
emp = joe;           // Ingenieure erhalten
emp.getSalary();     // Stundenlohn
```



Zweck: z.B. Liste von Angestellten, unabhängig von speziellen Merkmalen wie Stellung, Gehaltsberechnung, ...

- Problem:ist auch die Rückzuweisung `graham = wageRecipient;` möglich?

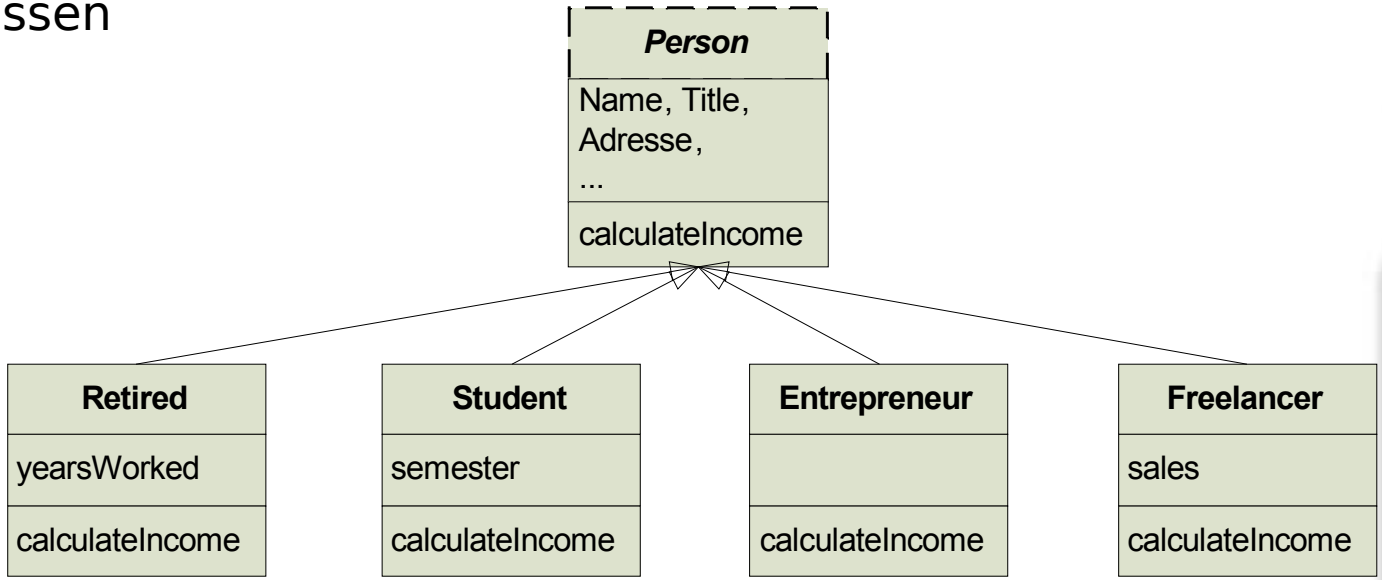
OO Programmierung - abstrakte Klassen

- **Abstrakte Klasse** unvollständige Klasse
- **Abstrakte Methode** ohne Implementierung

Eigenschaften

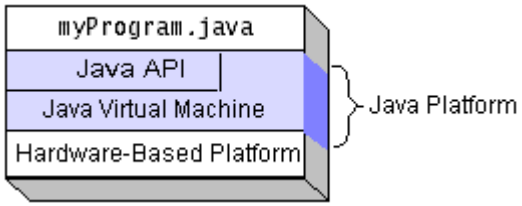
- von abstrakten Klassen können keine Objekte instantiiert werden
- abgeleitete (konkrete) Klasse müssen Implementierung realisieren

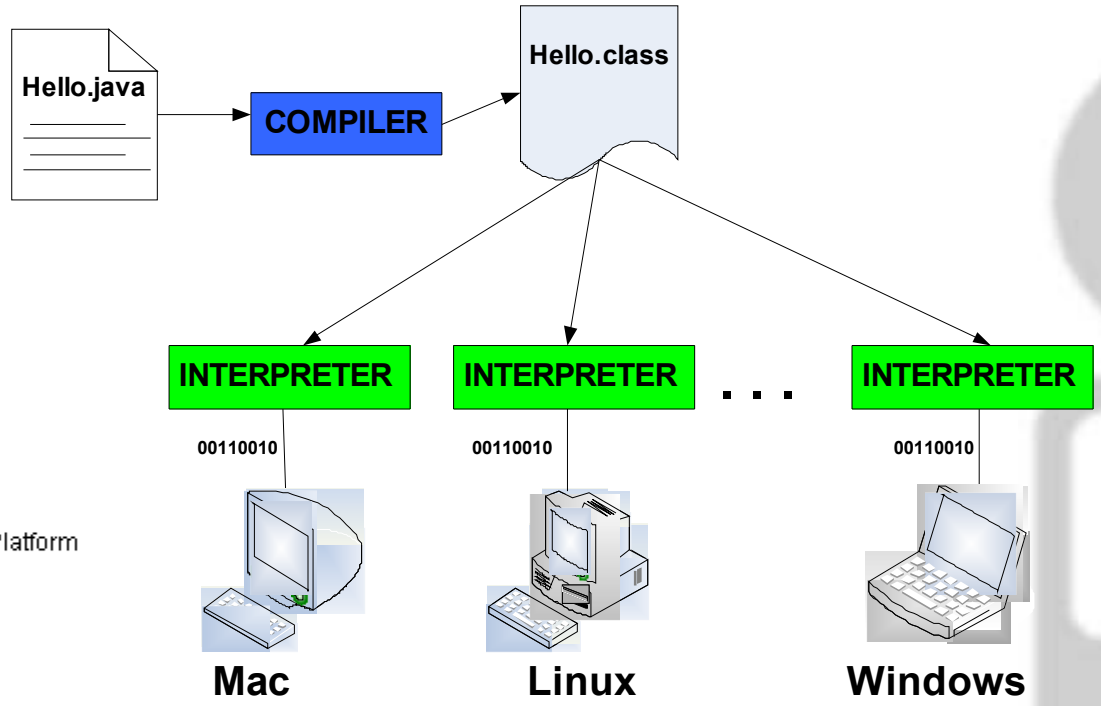
Zweck: Definition von Schnittstellen für eine Menge verschiedener Klassen



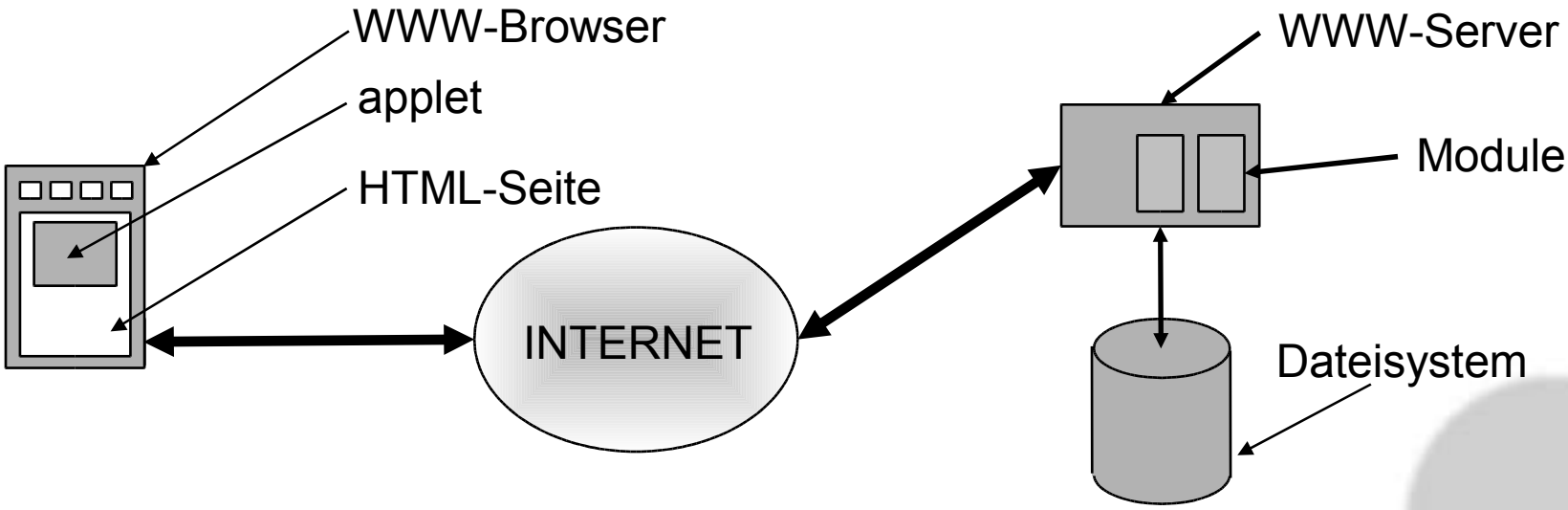
Was ist Java?

- “neue” Sprache, entwickelt von SUN seit 1990
- objektorientierte Programmiersprache, angelehnt an C++
- keine Trennung von Deklaration und Rumpf (.h, .c /.cpp)
- Plattformunabhängiger Bytecode für Internet / WWW





Internet WWW-Client und -Server



Internet-Protokolle:
ftp: file transfer protocol
http: hypertext transfer protocol
...

HTML: HyperText Markup Language
Standardformat für Textseiten im WWW
URL: Uniform Resource Locator
(eindeutige) Adresse eines Dokumentes im Internet

Versprechungen von Java (1)

- Robust
  - keine Zeigerarithmetik
  - Garbage-Collection
  - strenges Typkonzept
- Objektorientiert
  - geringer Sprachumfang
  - umfangreiche Klassenbibliothek (!)
- Einfach
  - syntaktisch ähnlich zu C++, aber einfacher
- Dynamisch
  - nur relevante Klassen laden und binden
- Plattformunabhängig (Quellcode + Bytecode)
  - keine Portierung notwendig
- lediglich Web-Browser zur Ausführung notwendig

Versprechungen von Java (2)

- Unterstützung der Verteilung im WWW
- Interaktive Anwendungen
- 'Intelligenz' für WWW-Client (fat clients)
  - im Web bisher lediglich Datenaustausch, keine Programme
  - HTML: statische Daten
- Unterstützung bewährter Konzepte anderer Sprachen
  - Ausnahmen
  - Garbage-Collection
  - Package-Konzepte
  - Concurrency
- Sicherheit?

Objektorientierte Programmierung mit Java

- Klassen
- Methoden, Attribute
- Verkapselung
- Vererbung
- Polymorphie (statische vs. dynamische)
- abstrakte Klassen und Methoden
- Objekte

- Aggregation vs. Assoziation
- Überladen

Java – Sprachumfang (1)

- Bezeichner `employee, Hello`
- Kommentare `//Zeile` `/* Bereich */`
- Basisdatentypen `int, float`
- Variablen `int zahl;`
- Konstanten `final float mwst = 16.0;`
- Arrays `int[] lotto = new int[6];`
- Operatoren `+, -`
- Kontrollstrukturen
  - `if`
  - `switch`
  - `for`
  - `while`

Java – Sprachumfang (2)

- Klassen - Vererbung
- Objekterzeugung u. -zugriff
- Methoden
- Konstruktoren
- Überladung von Methoden
- Importierung von Klassen
- Exceptions
- Schnittstellentypen
- Pakete
- Applets
- Applikationen

```
extends

Date bithdate = new Date();

setDate(Date geb);

public Date () {...}

setDate (Date geb, String info )

import java.util.*

exception

interface
```

Sprachumfang - Datentypen

- Bezeichner:

  identifier ::= *letter*{*letter*|*number*} (aber keine Schlüsselwörter)

- Kommentare

  `/* ... */` oder

  `// ...` (bis Zeilenende)

- Basisdatentypen

  - **`boolean`**
  - **`byte`** (1 Byte)
  - **`short`** (2 Byte)
  - **`int`** (4 Byte)
  - **`long`** (8 Byte)
  - **`char`** (2 Byte)
  - **`float`** (4 Byte)
  - **`double`** (8 Byte)

  Klasse `java.lang.String`

Sprachumfang - Variablendeklarationen

- Variablendeklarationen:

  variableDecl ::= [**final**] *Type variable* {**,** *variable*}**;**

  variable ::= *identifier*{[]}[=*initialization*]

- Attributdeklarationen:

  attrDecl ::= *qualifier variableDecl*

  qualifier ::= **static** |**public** |**protected** |**private**
  |**transient** |**volatile**

- Beispiele:

```
static int     nr;
       boolean pass=true;
final  float   PI=3.14
```

Konventionen

- Klassen und Schnittstellen: großgeschriebene Substantive (`Person, Buch, Iterator`)
- Methoden: kleingeschriebene Verben (`calculateIncome`), Spezialfällen:
  - `get`... und `set`... für attributähnlichen Zugriff
  - `is`... für Abfragen
  - `to`... für Konvertierungen
  - `length` für Längen
- Attribute: kleingeschriebene Substantive (`address`)
- Lokale Variablen: oft Abkürzungen, aber erkennbar! (`persIdx`)
- Konstanten: ganz groß geschrieben (`PERS_MAX`)

Sprachumfang - Felder

- **Grundlegendes**

```
int vector[];           // Nur Deklaration, Wert null
vector = new int[4];    // Mit Platz für 4 integer
int vectorB[] = { 1, 2, 3, 4};  // gleich mit Werten
```

- **Felder von Objekten**

```
Person friends[];           // Deklaration
friends = new friends[10]; // Platz für 10 Person-Objekten
for (int fIdx = 0; fIdx < friends.length; ++fIdx) {
    friends[fIdx] = new Person(); // Make friends
}
```

- mehrdimensionale Felder: `int matrix[4][3];`
- Feld-Elemente: `array.length` und `array.clone()`

Sprachumfang - Operatoren

- Binäre arithmetische Operatoren
  - `+` op1+op2
  - `-` op1-op2
  - `*` op1*op2
  - `/` op1/op2
  - `%` op1%op2 (Rest bei Division)
- Abkürzungen `++, --, +=, ..`
  - `op++` (Wert vor Inkrementierung auswerten)
  - `++op` (.. nach ..)
  - `op--, --op`
  - `i *= j` entspricht `i = i * j;`

Vergleichsoperatoren

`>, <` kleiner, größer

`>=, <=`

`==, !=` gleich, ungleich

- Logische Operatoren
  - `&&` op1 && op2 (Und)
  - `||` op1 || op2 (Oder)
  - `!` !op1 (Negation)
- Bit-Operatoren
  - `& | ~ << >> >>>`
- String-Operatoren
  - `+, +=`

Priorität der Operatoren

postfix Operatoren	`[] . (Parameter)`
einstl. Operatoren	`++expr , --expr, +expr, -expr, !, ~`
Erzeugung, Casting	`new, (type)expr`
Multiplikation	`* / %`
Addition	`+ -`
Shift	`>> << >>>`
Vergleich	`< > <= >= instanceof`
Gleichheit	`== !=`
Bitweises Und / Oder	`& |`
logisches Und / Oder	`&& ||`
Wenn a dann i sonst j	`a?i:j`
Zuweisungen	`= += -= /= %= = &= .`

Sprachumfang - if, switch

if-else

```
if (boolExpression) {...
} else if(boolExpression){...
} else {...
}
```

switch

```
switch (integralExpression) {
   case value: ... break;
   case value: ... break;
   default: ...
}
```

Beispiel:

```
...
if (i == 1) {
  System.out.println(i+"=1");
}
if (i > 10){
System.out.println(i+">10");
 j = i;
} else { System.out.println(i+"<=10");
 j = i +10;
}
```

Beispiel:

```
 ...
switch (i) {
  case 1 : System.out.println("1"); //!
  case 2 : System.out.println("2"); break;
  case 3 : System.out.println("3"); break;
  default:System.out.println("sonst");
}
```

Sprachumfang - for, while

while-Schleifen

```
while (Ausdruck){
   Anweisungen
}
```

Überprüfung vor Schleifendurchlauf

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

Überprüfung vor Wiederholung
⇒ mind. 1 Durchlauf

break abbrechen und hinter der Schleife fortfahren

continue abbrechen, nächsten Schleifendurchlauf starten

for-Schleife

```
for( Initialisierung
          ; Ausdruck ; Schritt){
   Anweisungen
}
```

bei Beginn: Initialisierung

in jedem Durchlauf:

1. Ausdruck überprüfen
2. Anweisungen durchführen
3. Schritt durchführen

```
...
int fak = 1;
for(int i = 1;i <= num;++i)
     {fak = fak * i;}
```

Sprachumfang - Klasse, Vererbung

Klasse

```
[public]
 [[abstract]|[final]]
class Name
[extends SuperKlassenName]
[implements SchnittstellenNamen]
{ Klassenkörper }
```

im Klassenkörper:

- Variablendeklarationen
- Methodendeklarationen
- Konstruktordeklarationen
- Initialisierung

in beliebiger Anzahl und Reihenfolge

public Klasse exportieren

final keine Ableitung möglich

abstract Methoden nicht vollständig implementiert, keine Instanzbildung möglich

extends erbt von ...

implements implementiert Methoden von Interface

Beispiel:

```
public class Beispiel {
   private int anzahl = 0;
   public void setAnzahl (int x)
   {} }
```

Sprachumfang - Objekt erzeugen

- **Wichtig: Jede importierbare Klasse wird in einer Datei mit gleichem Namen gespeichert.**

  (in obigem Beispiel: Beispiel.java)

- **Objekterzeugung und -zugriff**

  `new`: erzeugt ein neues Objekt einer Klasse,

  ruft Konstruktor auf

  Bsp: `String x = new String ("Zeichenkette");`

  oder `String x = "Zeichenkette2";`

- **Warum gibt es kein *delete*?**
  - Garbage Collector: gibt nicht mehr genutzte Speicherbereiche automatisch frei
- Zugriff über Referenzen (keine Pointer)

Objektinitialisierung

Objektinitialisierung:

static-Blöcke in der Klassendefinition werden bei Erzeugung ausgeführt

```
class TestInit {
    static { System.out.println("vorher");}

    public static void main (String args []){
        System.out.println("main");
    }

    static {
        System.out.println("nachher");
    }
}
```

Sprachumfang - Methodendeklaration

Methodendeklaration

[public|protected|private]

[static] [abstract|final]

[synchronized]

Resultattyp Bezeichner **{[]}**

([*Parameterliste***]) {[]}**

[*Ausnahmeerzeugung***]**

Methodenkörper

public: öffentliche Methode

protected: nur für abgeleitete Klassen zugreifbar

private: lokale Verwendung

static: definiert Klassenmethode statt Objektmethode, kann nicht redefiniert werden.

abstract: müssen in Ableitung definiert werden.

final: können in abgeleiteten Klassen nicht überschrieben werden.

synchronized: für Threads (s.u.)

Resultattyp: *Datentyp* | **void**

Parameterliste:

Typ bez **{[]}** [, *Typ bez*

Sprachumfang - Konstruktoren (1)

- spezielle Methoden zur Objekterzeugung
- Methodenname = Klassenname, kein Rückgabewert
- Verkettung via `this()` möglich
- Gegebenenfalls wird default-Konstruktor angelegt

```
public class Complex {
  public Complex (double real, double imag)
   {
    this.real = real; this.imag = imag;
   }
  public Complex () {
    this(0.0, 0.0);
   }
  private double real, imag;
 }
```

```
Complex K1=new Complex();
Complex K2=new Complex(1.0,2.0);
```

Sprachumfang - Konstruktoren (2)

- **Konstruktor chaining:**
  - Ein Konstruktor der Superklasse wird immer aufgerufen
  - Nur als erstes Statement auch explizit (`super()`) möglich

```
class Taxpayer {
  Taxpayer (String name) { this.name = name; }
  private String name;
}
class Employee extends Taxpayer {
  Employee (String name, int wage) {
    super (name);  this.wage = wage;
  }
  //Employee (int wage) { this.wage = wage; }  // ERROR
  private int wage;
}
```

Sprachumfang - Überladen

- **Überladen von Methoden**
  - Methoden mit gleichem Namen, aber unterschiedlichen Parameterlisten
  - häufig bei Konstruktoren

```
static class Math
{
  static void show (String title, int []   vektor);
  static void show (String title, int [][] matrix);
  static void show (              int [][] matrix);
}
public void printStuff(int[][] matrix, int[] vector) {
  Math.show ("The matrix:", matrix);
  Math.show (vector);
}
```

Sprachumfang - Exceptions

```
public class Employee { ...
  public setWage (int newWage) throws Exception {
    if (newWage <= 0)  {
      throw (new Exception ("Won't work for free!"));
    } else { this.wage = newWage; }
  }
}
class EmployeeTest {
  public static void main (String[] args) {
    int      zero = 0;
    Employee joe = new Employee(...);
    try {
      joe.setWage (zero);
    } catch (Exception e) {
      System.out.println ("Exception: " + e.getMessage() );}
    }
  }
}
```

Sprachumfang - Exceptions

- Reguläre Klassen unter `java.lang.Throwable`
- Konstruktor mit `String`-Parameter üblich
- `throw`/`try`/`catch` und `finally`

```
try {
    ...
} catch (Exception e) {
    System.err.println ("Error: "+ e.getMessage());
    ... // andere eigene Anweisungen im Fehlerfall
}
```

- Konvention: spezifische Exceptions vs. InternalError.
  - » Spezifisch: Aufrufer ist Schuld, wird am Anfang geworfen
  - » InternalError: Aufgerufener ist Schuld, wird irgendwann geworfen

Sprachumfang - Interfaces

- **Schnittstellentypen - Interfaces**

» eine Schnittstelle legt eine Menge von Methoden fest, liefert keine Implementierung

» Klasse kann mehrere Schnittstellen implementieren

» Definition:

```
public interface TakingOrders {
    void takeOrder (Task task);
}
```

» Implementierung:

```
public class Employee extends Taxpayer
                      implements TakingOrders {
    void takeOrder(...) { ... }
}
```

» Verwendung:

```
public void keepBusy (TakingOrders to) {
  to.takeOrder(new Task(GIANT));
}
```

Sprachumfang - Packages

- Klassen befinden sich immer in Paketen
- Package = logisch zusammengehörige Menge von Klassen
- Hierarchischer Aufbau durch Schachtelung von Packages
- Deklaration:

```
package Name;    // vor der Klassendefinition
```

`Name` muß der Verzeichnisbenennung entsprechen

Beispiel: `package money.receivers;` aktuelle Übersetzungseinheit im Verzeichnis: `money/receivers`

- Packages können komplett importiert werden:

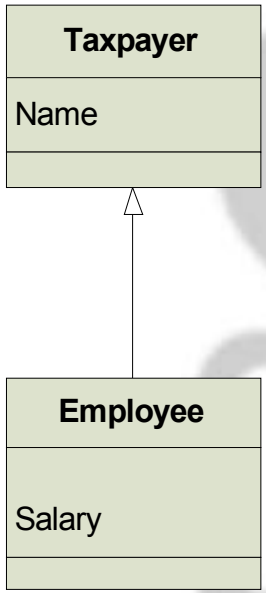
```
import money.receivers.*;
```

oder man importiert einzelne Klassen:

```
import money.receivers.Employee;
```

Sprachumfang - Substitution etc.

- Kompatibilität zwischen Klassen
  - supervariable = subinstanz : immer ok (substitution)
    Bsp: `Taxpayer tim = new Employee ("Tim", 1000);`
  - `subvariable = superinstanz` : nur mit downcast
    Bsp: `Employee tim = new Taxpayer ("Tim"); // ERROR`
  - `ClassCastException` bei Fehlern



- Typüberprüfung mit **`instanceof`**

Bsp: `if (tim instance of Employee) { … }`

Sprachumfang -Substitution

```
class Father {
   int x;
   Father() {x = 0;}
   public int getx() {
      return x;
   }
}
class SonA extends Father {
   int z;
   SonA() {z = 1;}
   public int getz() {
      return z;
   }
}
class SonB extends Father {
   int y;
   SonB() {y = 1;}
   public int gety() {
      return y;
   }
}
```

```
public class Test{
public static void main (String
[] args) {
Father f;
SonA sA, sA_new;
SonB sB, sB_new;
sA = new SonA();
sB = new SonB();

f = sA;
if (f instanceof SonA)
 sA_new = (SonA) f;

f = sB;
if (f instanceof SonB)
 sB_new = (SonB) f;

// Exception:
sA_new = (SonA) f;
}
```

Sprachumfang - abstrakte Klassen

- abstrakte Klassen und Methoden
- Nie abstrakt: Konstruktoren, **`static`**, **`final`**, **`private`**
- hat eine Klasse eine abstrakte Methode ist sie selbst auch abstrakt
- Beispiel:

```
abstract class Taxpayer {
   abstract String show();
   void print() {
      System.out.println(">"
      + show() + "<");
    }
}

class Employee
          extends Taxpayer {
   String show() {
      return "Hallo";
    }
}
```

```
class TestAbstract {
 public static void main
         (String args []){
  Employee tom= new Employee();
  Employee.print();

  Taxpayer tim = new Employee();
    System.out.println(
      "[" + tim.show()
          + "]");
  }
}
```

Spezielle Klassen und Möglichkeiten

- AWT, Swing — grafische Bedieneroberflächen
- JDBC (Java Database Connectivity) auf relationale Datenbanken — Schnittstelle für Zugriff
- RMI (Remote Method Invocation) — Aufruf von Methoden entfernter Objekte
- IDL (Interface Definition Language) — Java-IDL für Kommunikation auf CORBA-Basis
- Objektserialisierung — Speichern und rekonstruieren von Objekten/ Erstellen einer lokalen Version eines entfernten Objektes
- Threads — parallele Ausführung von Prozessen
- Beans — Java Komponentenarchitektur

Java Standard Packages

- Die Packages umfassen Klassen und Schnittstellen
- `java.lang`
  - Kern der Java Sprache
  - automatisch importiert von allen Übersetzungseinheiten
  - » `String`, `Object`, `Exception`, `Error`, `Thread`, `Math`
- `java.util`
  - versch. Hilfsklassen und -interfaces
  - Zufallszahlen, Systemeigenschaften, ...
  - » `Random`, `Date`, `Vector`, `Properties`, `Map`
- `java.io`
  - Ein- und Ausgabe bei Files und Streams
  - » `File`, `FileInputStream`, `FileOutputStream`, ...

Java Standard Packages

- `java.net`
  - Netzwerkkooperationen auf Socket- und URL-Ebene
- `java.awt` (inzwischen: `javax.swing`)
  - abstract windowing toolkit
  - Klassen für grafische Benutzeroberfläche
  - » `Font`, `Image`, `Color`, `Dialog`, `Event`, `Menu`, `List`, `Graphics`, `Frame`, `Window`, `Panel`
- `java.applet`
  - Einbettung in HTML, starten, Parameter ermitteln, ...
  - » `Applet`, `AppletContext`, `Component`, `Container`

Probleme

- schlechtes Laufzeitverhalten
  - verbessert durch Just-In-Time Compiler
  - Compiler in Maschinensprache (native Code)
  - Hardware-Emulatoren für die VM
- Sicherheit
  - auf dem Client wird eine unbekannte Applikation gestartet, die mit einem fremden Server kommunizieren kann
  - Zugriffe auf entferntes Dateisystem?
- in der Praxis nicht immer portabel

Entwicklung mit Java

- JDK (Java Development Kit) ist die offizielle Java-Entwicklungsumgebung
- besteht aus:
  - Java-Compiler: `javac`
  - Java-Interpreter: `java`
  - Applet-Anzeigewerkzeug: `appletviewer`
  - Java-Debugger: `jdb`
  - Java-Dokumentationserzeuger: `javadoc`
  - Java-Bytecode-Dissassembler: `javap`
  - Java-C-Stub-Erzeuger: `javah`
- verfügbar für Plattformen
  - Windows 95 und NT, 2000, XP, ...
  - Solaris (Sparc/x86)
  - Linux

Kleine Schritte...

- (Umgebungsvariablen setzen: `CLASSPATH`, `PATH`)
- Quelltext mit Texteditor **erstellen**
- für jede exportierte (***public***) Klasse eine neue Textdatei mit gleichem Namen anlegen: `Hallo.java` für Klasse `Hallo`
- **Übersetzen** mit **`javac`** `Hallo.java`
- **Ausführen**:
  - Applikation:

    `java` `Hallo`
  - Applet (hier nicht weiter betrachtet):
    - HTML-Datei erzeugen, die das Applet nutzt
    - Diese anzeigen lassen:

      `appletviewer` `Hallo.html`

      oder

      `irgendeinBrowser` `Hallo.html`

UNIX-Krücke

- Hilfe:

  `http://www.informatik.rwth-aachen.de/Pool/index.html`

  `http://www.informatik.rwth-aachen.de/Pool/hilfe.html`

- Einloggen: Monitor einschalten, Login-Name und Passwort eingeben
- Ausloggen: unterschiedlich, ggf. Monitor ausschalten.

!Auf **keinen** Fall Rechner ausschalten!

UNIX-Krücke (2)

- Programme mit `&` im Hintergrund starten:
  z.B. `/opt/rbi/x11/bin/mozilla &`
- Terminal: `xterm &`
- Editor: `xemacs &` oder `nano &`
- Achtung: Dateisystem unterscheidet Groß- und Kleinschreibung
- Wichtige Befehle:
  - Verzeichnisinhalt auflisten: `ls -la`
  - Verzeichnis wechseln:
    `cd ..` (Leerzeichen!)
    `cd` (Wechsel ins Homeverzeichnis)
    `cd tmp` (Wechsel ins Verzeichnis `tmp`)
  - Verzeichnis erstellen: `mkdir src`

Java-Applikationen

- besitzen eine statische **`main`**-Methode
- Beispiel:

```
public class Hello {
    public static void main (String[] args) {
        System.out.println("Hello World!");
    }
}
```

- Übersetzen **`javac`** `HelloWorld.java`

  Starten mit **`java`** `HelloWorld`

Java unterscheidet Groß- und Kleinschreibung!

Literatur

- Flanagan, David : *Java in a Nutshell,* O'Reilly & Associates,
  und *Java Examples in a Nutshell,*
- Krüger, Guido: *Go To Java 2,* Addison-Wesley, 2. Aufl., 2000, **`www.javabuch.de`**
- Javadoc
- ...

Links im WWW

- Java Seiten am Lehrstuhl
  `http://docs-i3.informatik.rwth-aachen.de:8080/`
  inkl. Online-Bücher
- Java Corner (Sunsite Aachen)
  `http://sunsite.informatik.rwth-aachen.de/java`
- Java Home-Pages
  `http://www.javasoft.com`
- Java Seiten (Deutschland)
  `http://java.pages.de`