

RWTH Aachen
Lehrstuhl für Informatik III
Prof. Dr.-Ing. M. Nagl

Softwarepraktikum Webtechnologien

Java - Crashkurs

Simon Becker
Boris Böhlen
Bernhard Westfechtel

Wintersemester 2002/2003

Gliederung

- † Organisatorisches
- † Objektorientierung im Allgemeinen
- † Die Web-Programmiersprache Java
- † Java auf UNIX-Rechnern verwenden
- † Verfügbare Informationen (Literatur, URLs)

Organisatorisches

† Ablauf

- » Crashkurs Java/Webtechnologien diese Woche
- » Bearbeitung in Kleingruppen (zu dritt)
- » Wöchentliches Treffen im AH III
- » Wöchentliche Testate im CIP-Pool

† Voraussetzung für den Erwerb des Scheins:

- » Teilnahme an dem Crashkurs
- » sinnvolle, selbständige Bearbeitung aller Aufgaben
- » Vorführung und Erläuterung der Lösung im CIP-Pool (beim Testat)
- » Beteiligung an der Gruppenarbeit
- » Ein 'Joker' pro Gruppe, aber nicht beim letzten Aufgabenblatt

Organisatorisches (2)

- † Reservierungen im CIP-Pool (Raum 4U16):
 - » Mo. 13:00 – 17:00 Uhr
 - » Mi. 9:00 – 13:00 Uhr
 - » Do. 10:00 – 12:00 Uhr

- † Testat: Do. 10:00 – 12:00 Uhr

- † Wöchentliche Besprechung: Do. 14:00 – 15:00 Uhr, AH III

Kontakt

- † Assis per Mail: spws02be@i3.informatik.rwth-aachen.de

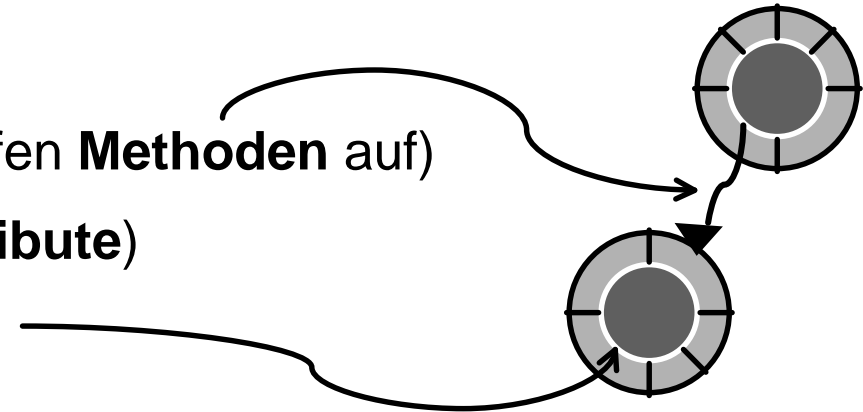
- † Teilnehmer per Mail: spws02@i3.informatik.rwth-aachen.de

- † Telefon:
 - » Simon Becker 80-21316
 - » Boris Böhlen 80-21312
 - » Bernhard Westfechtel 80-21310

OO - Grundlagen

† Objekte

- » tauschen Nachrichten aus (rufen **Methoden** auf)
- » können Zustände haben (**Attribute**)
- » können Innereien **verkapseln**



† Klassen

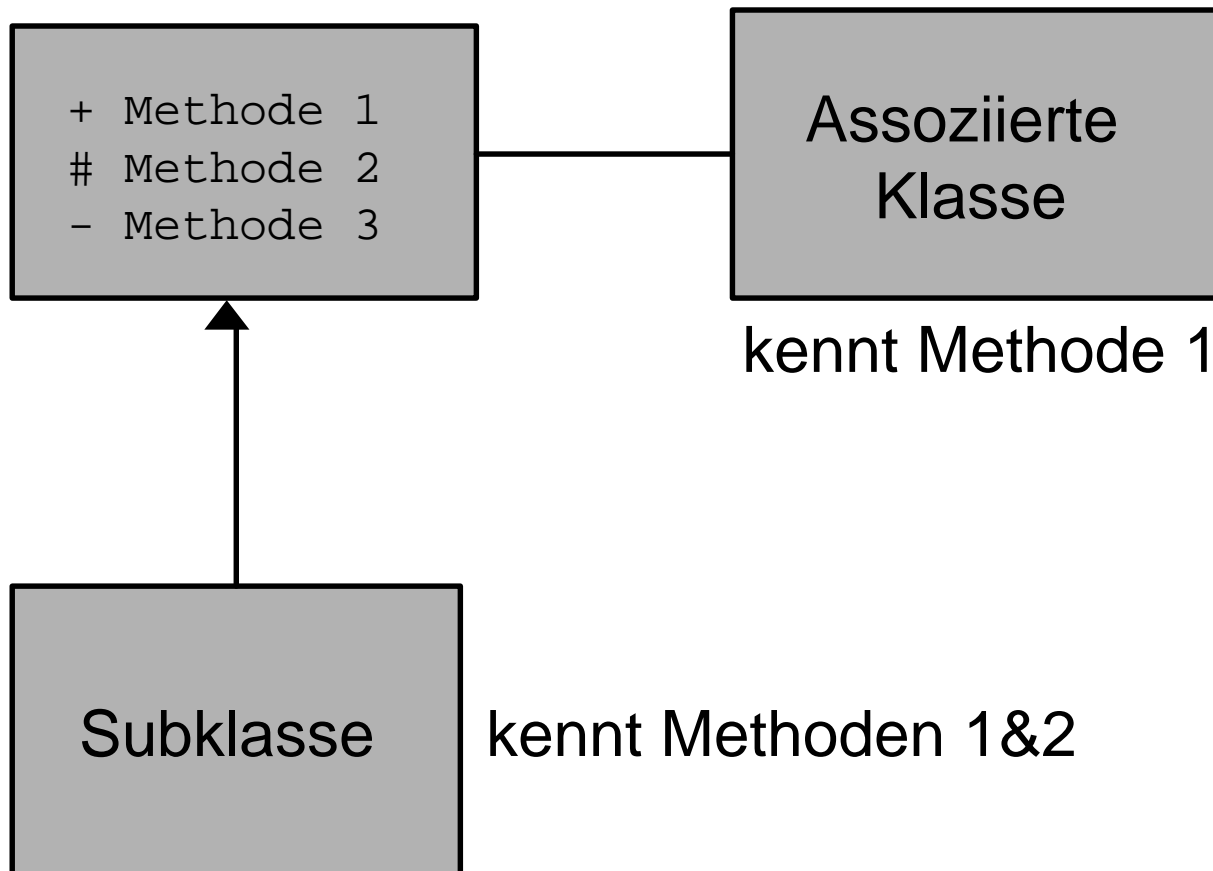
- » gruppieren gleich strukturierte Objekte (**Instanzen**)
- » definieren verfügbare Methoden und Attribute (**Elemente**)

† Sichtbarkeiten: `public`, `protected`, `package`, `private`

† Implizites Verhalten

- » Objekte werden durch **Konstruktoren** initialisiert
- » Vor Zerstörung wird **Destruktor** aufgerufen

Sichtbarkeiten



Klassen und Objekte

- † Klassen sind Konstrukt der Programmierung, Instanzen (Objekte) sind Konstrukte der Laufzeit.
- † Klassen sind statisch, Objekte sind dynamisch und verändern ihren Zustand zur Laufzeit.
- † Die gesamte Objektkonfiguration zur Laufzeit spiegelt den Zustand des Systems wieder.
- † Objekte werden aus Klassen instantiiert.
- † Im Allgemeinen können beliebig viele Objekte aus einer Klasse instantiiert werden.

OO vs. Module (imperative PS)



† In der Softwaretechnik Unterscheidung verschiedener Modultypen:

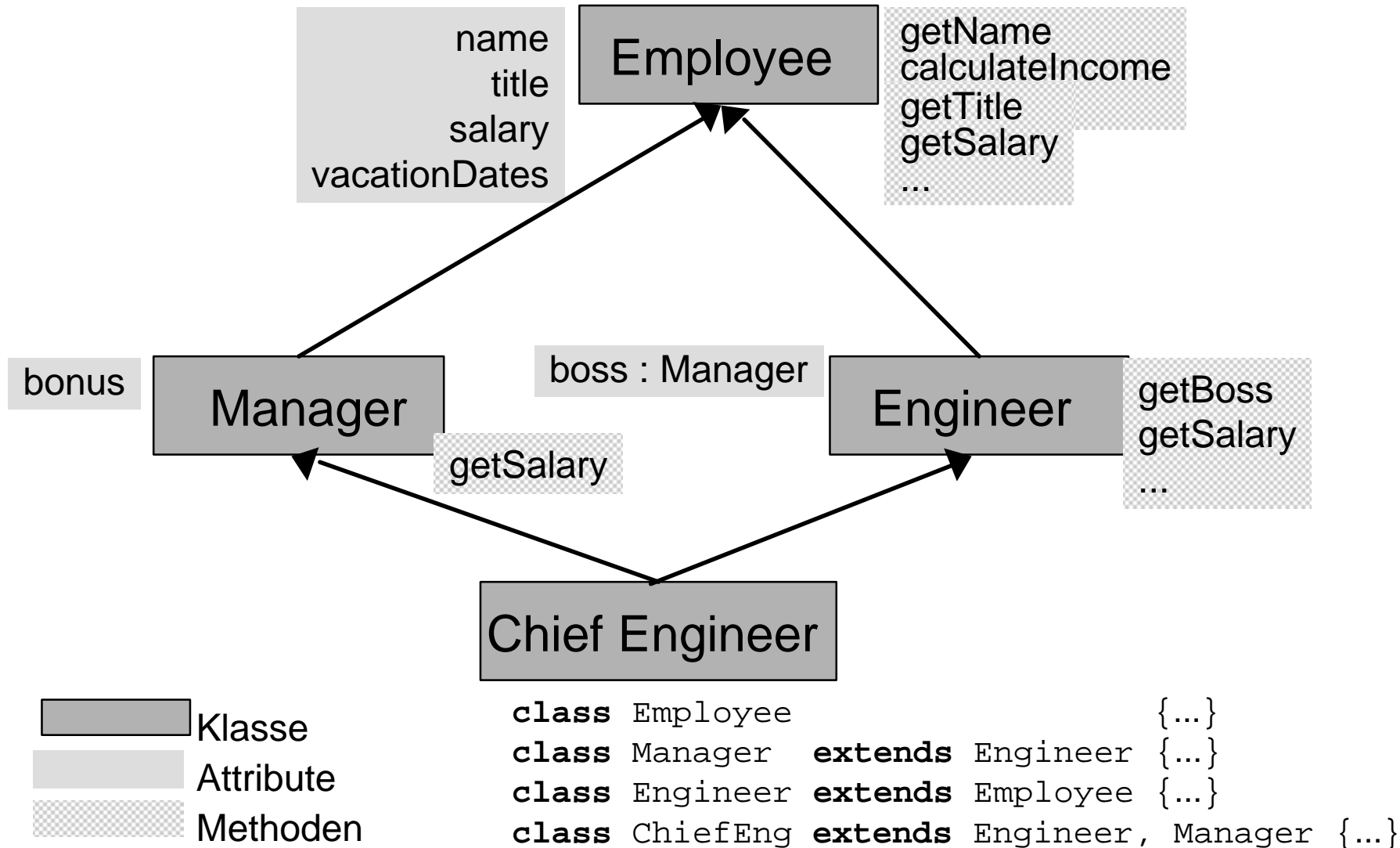
- » Funktional: Transformation, kein Gedächtnis:
 - ☞ `double sin(double); ...`
- » Datenobjekt: Datenablage, nur eine 'Instanz':
 - ☞ `StandardInput`
- » Datentyp: Vorlage für mehrere Instanzen:
 - ☞ `Person`

† *normale Klasse ? Datentypmodul*

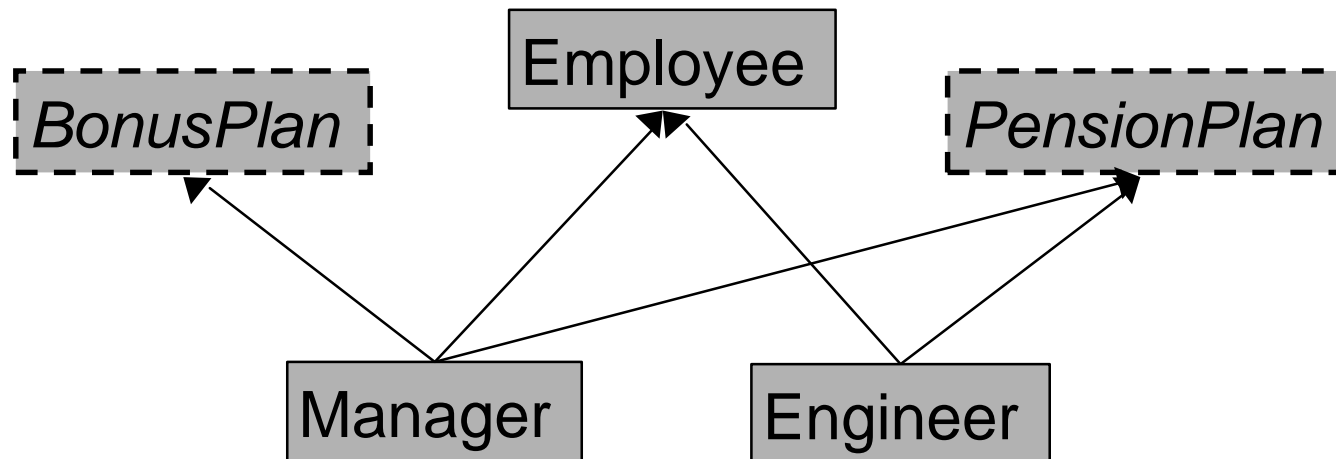
OO - Vererbung (1) - Konzepte

- † eine Klasse drückt Ähnlichkeiten von Objekten in Elementen aus.
- † **Spezialisierungen** einer Klasse A sind Klassen, die von A erben aber spezielle Änderungen der Implementierung oder eine erweiterte Schnittstelle benötigen (“Kinder”, Erben, abgeleitete Klassen)
- † **Generalisierung** von mehreren Klassen ist die Superklasse (“Vater”). Umkehrkonzept der Spezialisierung
- † **Substitution**: wo Superklasse steht, darf Subklasse verwendet werden
- † Methoden von Vorgängerklassen können überschrieben werden, überschriebene Methoden werden mit `super.Name` aufgerufen

OO - Vererbung (2) - Beispiel



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

- † **Aggregation:** ein Objekt enthält andere Objekte.
- † **Assoziation:** ein Objekt kennt 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

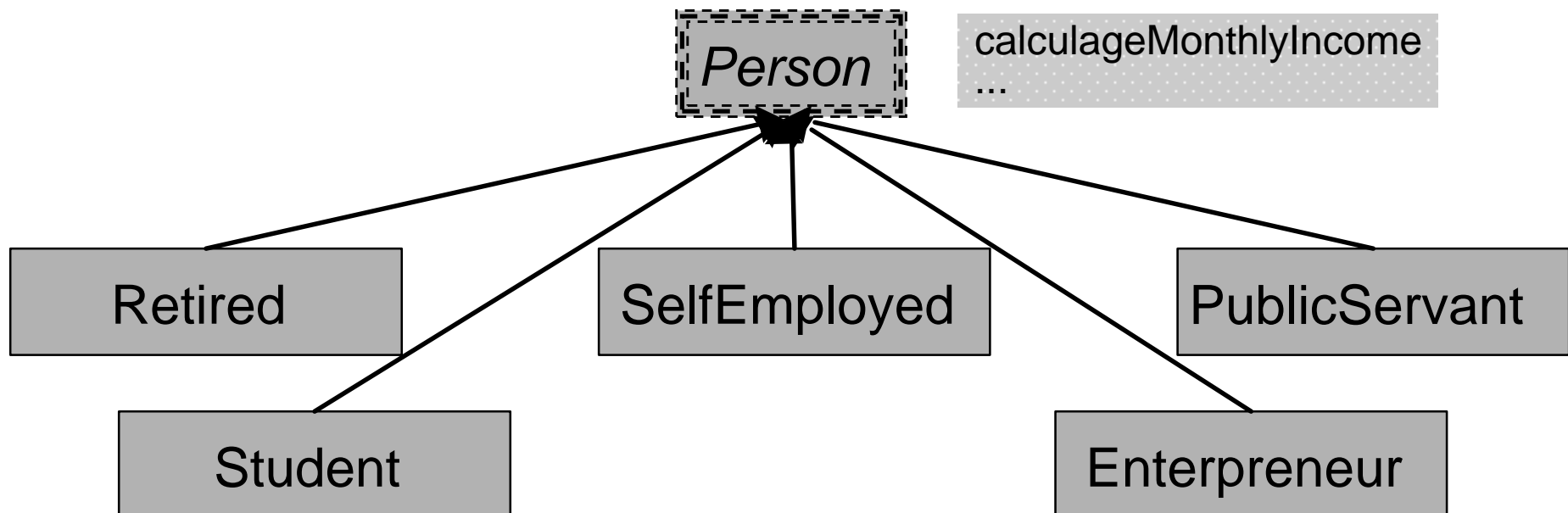
```
Boss      aldritch = new Boss(...);
Engineer joe      = new Engineer(...);
Employee emp;

emp = aldritch ;      // aldritch gets payd
emp.getSalary();     // by Income!
emp = joe;           // joe gets payd
emp.getSalary();     // only wage!
```

- † Zweck:
z.B. Liste von Angestellten, unabhängig von speziellen Merkmalen wie Stellung, Gehaltsberechnung, ...
- † Problem:
ist auch die Rückzuweisung `graham = wageRecipient;` möglich?

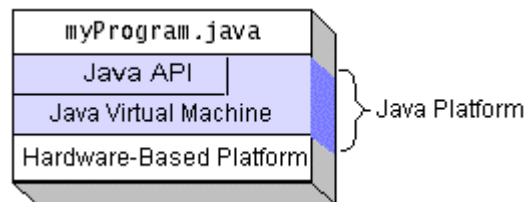
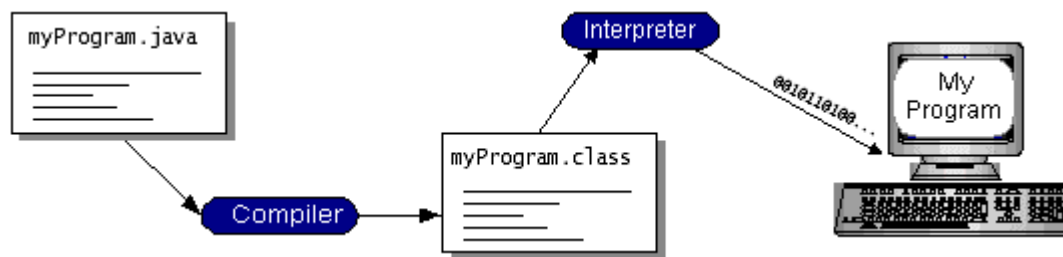
OO Programmierung - abstrakte Klassen

- † **Abstrakte Methode:** Methode ohne Implementierung
- † **Abstrakte Klasse:** unvollständige Klasse
(oft mit abstrakten Methoden)
- † von abstrakten Klassen können keine Objekte instantiiert werden
- † Objekte können nur von abgeleiteten, nicht mehr abstrakten Klassen erzeugt werden.
- † Zweck: Bereitstellung von Schnittstellen für eine Menge verschiedener Klassen



Was ist Java?

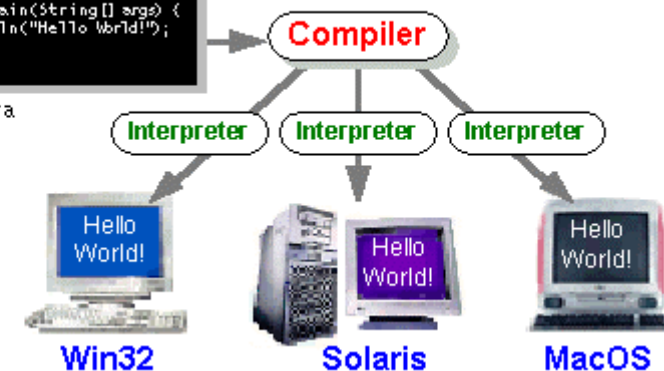
- † “neue” Sprache, entwickelt von SUN seit 1990
- † objektorientierte Programmiersprache
- † angelehnt an C++
- † jedoch keine Trennung von Deklaration und Rumpf (.h, .c / .cpp)
- † Plattformunabhängiger Bytecode für Internet / WWW



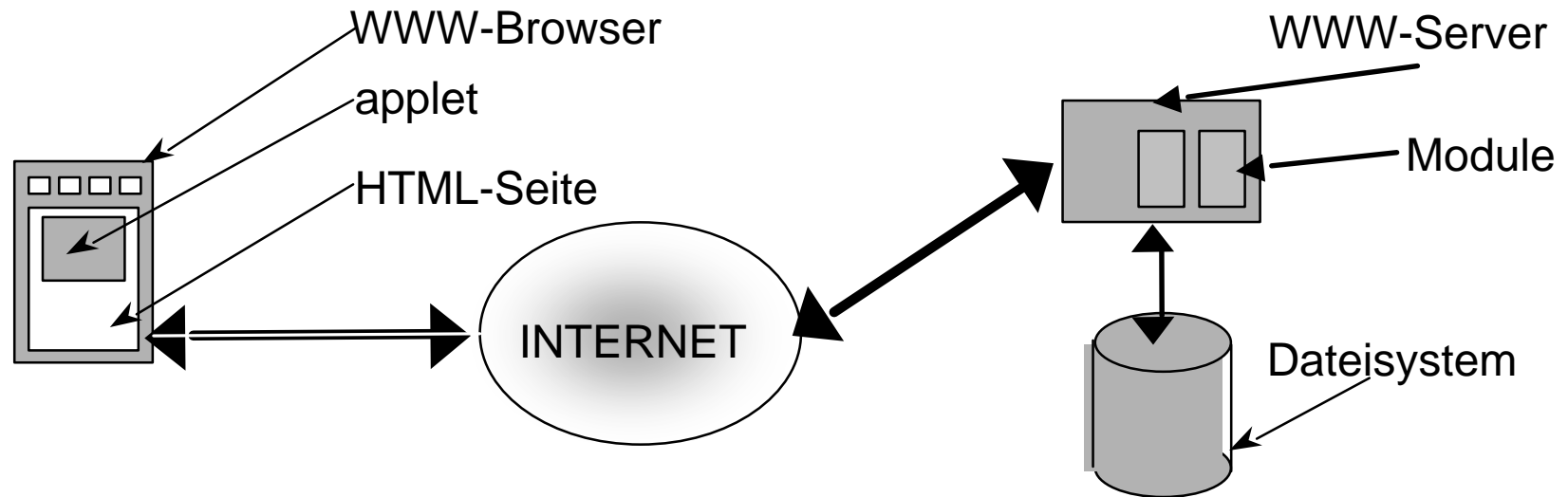
Java Program

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

HelloWorldApp.java



Internet WWW-Client und -Server



Internet-Protokolle:

ftp: file transfer protocol

http: hypertext transfer protocol

gopher, ...

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
 - » einfacher als C++, aber syntaktisch ähnlich
- † dynamisch
 - » nur relevante Klassen laden und binden
- † plattformunabhängig (Quellcode + Bytecode)
 - » keine Portierung notwendig
- † lediglich Web-Browser zur Ausführung notwendig

Versprechungen von Java (2)

- † unterstützt Verteilung im WWW
- † erlaubt interaktive Anwendungen
- † bringt 'Intelligenz' zum WWW-Client (fat clients)
 - » im Web bisher lediglich Datenaustausch, keine Programme
 - » HTML: statische Daten
- † unterstützt bewährte Konzepte anderer Sprachen
 - » Ausnahmen
 - » Garbage-Collection
 - » Package-Konzept
 - » Concurrency

- † Sicherheit?

Objektorientierte Programmierung mit Java

- † Objekte
- † Methoden, Variablen (Elemente)
- † Verkapselung
- † Klassen
- † Vererbung
- † Polymorphie
- † abstrakte Klassen und Methoden

- † Aggregation vs. Assoziation
- † Überladen

Java - Sprachumfang

- † Bezeichner
- † Kommentare
- † Basisdatentypen
- † Variablen
- † Konstanten
- † Arrays
- † Operatoren
- † Kontrollstrukturen
 - » if
 - » switch
 - » for-Schleife
 - » while-Schleife
- † Klassen - Vererbung
- † Objekterzeugung u. -zugriff
- † Methoden
- † Konstruktoren
- † Überladung von Methoden
- † Importierung von Klassen
- † Exceptions
- † Schnittstellentypen
- † Pakete
- † Applets
- † Applikationen

Sprachumfang - Datentypen

† Bezeichner:

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

† Kommentare

/ ... */* oder
// ... (bis Zeilenende)

† Basisdatentypen

- » **boolean**
- » **byte**
- » **short**
- » **int**
- » **long**
- » **char**
- » **float**
- » **double**

† 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;
```

† Vordefinierte Werte, aber nicht bei lokalen Variablen!

Konventionen

- † Klassen und Schnittstellen: großgeschr. Substantive (`Person`)
- † Methoden: kleingeschr. Verben (`calculateIncome`), mit Spezialfällen:
 - » `get...` und `set...` für attributähnlichen Zugriff
 - » `is...` für Abfragen
 - » `to...` für Konvertierungen
 - » `length` für Längen
- † Attribute: kleingeschr. 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 ints
```

```
int vectorB[] = { 1, 2, 3, 4}; // gleich mit Werten
```

† Felder von Objekten

```
Person friends[];           // Deklaration  
friends = new friends[10]; // Platz für 10 Person-Obj.  
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) expr++ expr--
einstl. Operatoren	++expr , --expr, +expr, -expr, !, ~
Erzeugung, Casting	new , (type)expr
Multiplikation	* / %
Addition	+ -
Shift	>> << >>>
Vergleich	< > <= >= instanceof
Gleichheit	== !=
Bitweises Und	&
Bitweises Oder	
logisches Und	&&
logisches Oder	
Wenn a dann i sonst j	a?i:j
Zuweisungen	= += -= /= %= = &= ...

Sprachumfang - if, switch

if-else

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

Beispiel:

```
int i, j;  
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;  
}
```

switch

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

Beispiel:

```
int i;  
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

for-Schleife

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

bei Beginn: Initialisierung

in jedem Durchlauf:

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

```
for(int i = 0;i < 5;++i)
    {} // i = 5;
for(int i = 0;i < 9;++i)
    {i = i * 2;} // i = 15;
```

while-Schleifen

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

Überprüfung vor Schl.-durchlauf

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

Überprüfung vor Wiederholung

? mind. 1 Durchlauf

break abbrechen und hinter der
Schleife fortfahren

continue abbrechen, nächsten
Schleifendurchlauf starten

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 mögl.

abstract

Methoden nicht vollst.
implementiert, keine

Instanzbildung möglich

extends

erbt von ...

implements

implementiert

Methoden von...

Beispiel:

```
public class Beispiel {
```

```
    public int x = 100;
```

```
}
```

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) z.B. `x.length();`

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");
    }
}
```

Ausgabe: vorher nachher main

Sprachumfang - Methodendeklaration

† Methodendeklaration

[**public** | **protected** | **private**]

[**static**] [**abstract** | **final**]

[**synchronized**]

Resultattyp *Bezeichner* {[]}

([*Parameterliste*]) {[]}

[*Ausnahmeerzeugung*]

Methodenkörper

static: definiert Klassen-
methode statt Objekt-
methode, kann nicht
redefiniert werden

public: Meth. importierbar

private: lokale Verwendung

protected: nur für abgeleitete
Klassen zugreifbar

abstract: müssen in Ableitung
definiert werden.

final: dürfen in abgel. Kl. nicht
überschrieben werden

synchronized: für Threads (s.u.)

Resultattyp: *Datentyp* | **void**

Parameterliste:

Typ *bez* {[]}

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

Ausnahmeerzeugung: s.u.

Methodenkörper: { . . . } | ; (; = leer)

Sprachumfang - Konstruktoren (1)

- † spezielle Methoden zur Objekterzeugung
- † Methodename = Klassentypname, kein Rückgabewert
- † Verkettung via `this()` möglich
- † Gegebenenfalls 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
- » Gegebenenfalls impliziter Aufruf eingefügt

```
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 gleichen 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 zilch = 0;
        Employee joe = new Employee(...);
        try {
            joe.setWage (zilch);
        } 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:

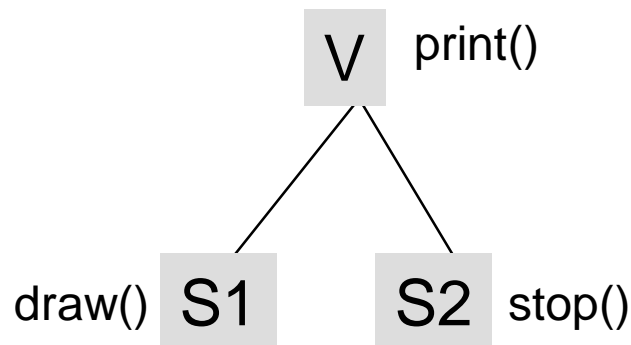
```
public void keepBusy (TakingOrders to) {  
    to.takeOrder(new Task(GIANT));  
}
```

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
CPK/money/receivers (für eine CLASSPATH -Komponente CPK)
- † 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)
 - » subvariable = superinstanz : nur mit downcast
 - » Typüberprüfung mit `instanceof`
 - » `ClassCastException` bei Fehlern



```
V v;  
S1 s1;  
S2 s2;
```

```
v = s1;  
v.print();
```

```
v = s2;  
v.print();
```

```
s1 = (S1) v;  
s1.draw();
```

```
s2 = (S2) v;  
s2.stop();
```

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, sAnew;
    SonB sB, sBnew;
    sA = new SonA();
    sB = new SonB();

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

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

    // Exception:
    sAnew = (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 Super {  
    abstract String specific();  
    void general () {  
        System.out.println(">"  
            + specific + "<");  
    }  
}  
  
class Sub extends Super {  
    String specific () {  
        return "Sub";  
    }  
}
```

```
class TestAbstract {  
    public static void main  
        (String args []) {  
        Sub sub = new Sub();  
        sub.general();  
  
        Super sup = new Sub();  
        System.out.println(  
            "[" + sup.specific  
            + "]" );  
    }  
}
```

Spezielle Klassen und Möglichkeiten von Java

† AWT, Swing	grafische Benutzeroberflächen
† JDBC (Java Database Connectivity)	Schnittstelle für Zugriff auf relationale Datenbanken
† 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
† Netzwerkklassen	Netzwerk-Kommunikation
† Beans	Java Komponentenarchitektur

Java Standard Packages

- † Die Packages umfassen Klassen und Schnittstellen
- † `java.lang`
 - » Kern der Java Sprache
 - » autom. 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`
 - » Netzwerkoperationen 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 von Java

- † 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
 - » 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
```

Keine UNIX-Einführung

† 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!

† Programme mit & im Hintergrund starten:

z.B. `/opt/rbi/x11/bin/mozilla &`

† Terminal: `xterm &`

† Editor: `xemacs &`

† Das Dateisystem unterscheidet Groß- und Kleinschreibung

Java-Applikationen

- † **Applikationen**
- † besitzen eine `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>