
Java

Organisatorisches

- 2. Hälfte der ILV "Informatik und Programmieren"
- 15 Wochen à 2 LE = 30 LE = 2 SWS
- zum Stundenplan
  - 2 Wochentypen: Vorlesung, Übung
  - Geplanter Ablauf: VO-VO-UE, usw. ...

Montag Gr 1	Gr 2	Gr 3	Gr 4	Gr 5
			ET ILV Baumgartinger SR 22	
AuV I VO Wassermann Raffaseder HS 2				
JfEin VO Barta G Anfänger HS 2			INFuP ILV Grabo Fortgeschr. SchulungR.	
E ILV Zehetmayr SR 1	ET ILV Baumgartinger SR 21	INFuP ILV Barta G. SR2	INFuP ILV Grabo **B:** CR 2	E ILV Juhasz SR 22
ET ILV Baumgartinger SR 21	E ILV Juhasz SR 22	E ILV Zehetmayr SR 1	INFuP ILV Barta G.	INFuP ILV Grabo **A:** CR 2 **B:** CR 2

Dienstag Gr 1	Gr 2	Gr 3	Gr 4	Gr 5
	INFuP ILV Barta G. SR2	INFuP ILV Grabo **B**: CR 2	E ILV Zehetmayr SR 5	**WB I ILV Judmaier Schmiedl G. Seidl CR 3**
INFuP ILV Grabo **A:** HS 2 **B:** CR 2	**WB I ILV Judmaier Schmiedl G. Seidl CR 3**	ET ILV Zotlöterer SR 5	INFuP ILV Grabo **A:** HS 2	INFuP ILV Barta G. SR2
INFuP ILV Barta G. SR2	INFuP ILV Grabo **A:** HS 2 **B:** CR 2	INFuP ILV Grabo **A:** HS 2	**WB I ILV Judmaier Schmiedl G. Seidl CR 3**	ET ILV Zotlöterer SR 5

Unterlagen

- primäre Unterlage: eure Mitschrift (!)
- Buch "Java – Grundlagen der Programmierung"
- Website der LV im Rahmen des eCampus

Beurteilung

- Modus je nach Selbsteinschätzung:
  - Beginner: Mitarbeit und schriftlicher Test am Semesterende
  - Forgeschrittene: Projektarbeit

Inhalt

- Einführung in die OOP am Beispiel Java
- im Kontext bzw. als Grundlage von Software Engineering als übergeordnete Disziplin
- mit Blick in Richtung grössere Software-Systeme

Ziele

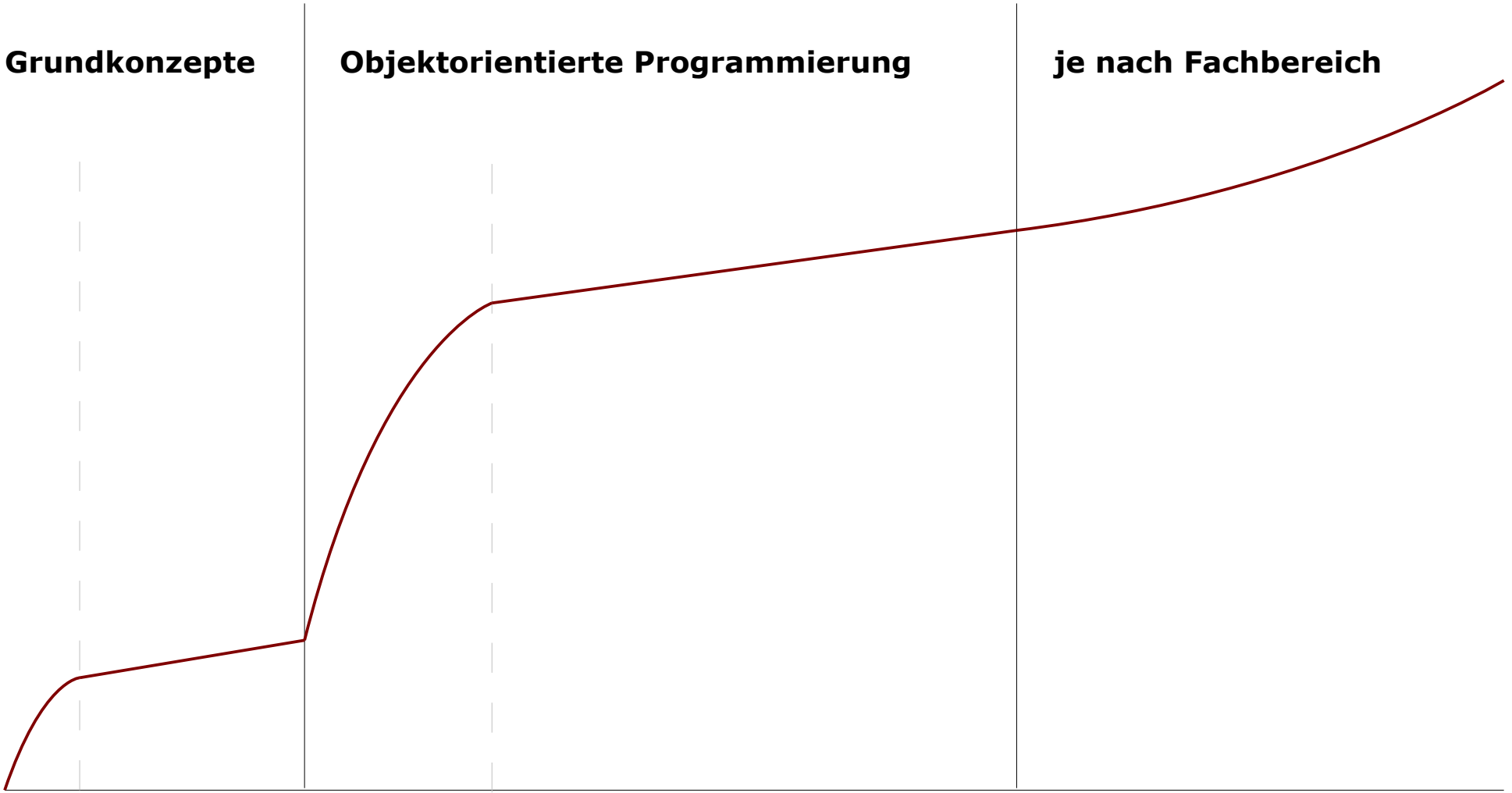
- solide Grundkenntnisse der OOP
- Verstehen und Beurteilen von kleinen Software Systemen
- Implementieren und Warten von kleinen Software Systemen

Überblick

- Einführung in das Programmieren (S. 1-140, S. 181-226, = 185 Seiten)
  - Grundlagen (Kapitel 1)
  - Datentypen und Operatoren (Kapitel 2)
  - Kontrollanweisungen (Kapitel 3)
  - Arrays (Kapitel 5)
- Objektorientierte Programmierung (S. 141-180, S. 227-388, = 200 Seiten)
  - Klassen, Objekte Methoden (Kapitel 4)
  - Genauere Betrachtung der Methoden und Klassen (Kapitel 6)
  - Vererbung und Polymorphie (Kapitel 7)
  - Pakete und Schnittstellen (Kapitel 8)
  - Ausnahmebehandlung (Kapitel 9)

Mehrteilung der Lernkurve

(Privatmeinung)



Überblick

- Einführung in das Programmieren (S. 1-140, S. 181-226, = 185 Seiten)
  - Grundlagen (Kapitel 1, **im Schnelldurchlauf!**)
  - Datentypen und Operatoren (Kapitel 2)
  - Kontrollanweisungen (Kapitel 3)
  - Arrays (Kapitel 5)
- Objektorientierte Programmierung (S. 141-180, S. 227-388, = 200 Seiten)
  - Klassen, Objekte Methoden (Kapitel 4)
  - Genauere Betrachtung der Methoden und Klassen (Kapitel 6)
  - Vererbung und Polymorphie (Kapitel 7)
  - Pakete und Schnittstellen (Kapitel 8)
  - Ausnahmebehandlung (Kapitel 9)

Ziele

- Programmiersprache, was?
- Geschichte und Philosophie von Java
- Grundprinzipien der objektorientierten Programmierung
- Allgemeiner Aufbau eines Java Programms
- Variablen, Datentypen
- Kontrollanweisung
- Umgang mit Codeblöcken
- Java Schlüsselwörter

Programmiersprache, was?

"Eine Programmiersprache ist eine künstlich geschaffene Sprache zur Darstellung von Berechnungen. Programmiersprachen sollen Berechnungen sowohl in einer für einen Computer, als auch in einer für den Menschen lesbaren und verständlichen Form ausdrücken.Programmiersprachen sind notwendig, da die natürlichen Sprachen für eine genügend detaillierte und präzise Beschreibung von Computerberechnungen zu vieldeutig sind."

http://de.wikipedia.org/wiki/Programmiersprache

- es existiert eine Vielzahl von Programmiersprachen
- PS unterliegen einer kontinuierlichen, "evolutionären" Entw.
  siehe auch http://www.oreilly.de/artikel/prog_sprachen_poster.pdf

Geschichte von Java

- Java ist nicht nur, aber auch eine Programmiersprache
- wurde Anfang der 90er von Sun entwickelt
- aktuell in der Version 1.5, gerne "Java 5" genannt
- die grossen Drei: C, C++, Java

- eine "beste" Sprache gibt es nicht: jede Sprache hat Stärken und Schwächen und ist für bestimmte Problembereiche besser oder schlechter geeignet

- immerhin: wer Java kann, der hat 'ne Menge verstanden

Philosophie von Java (1)

- Portabilität
  - plattformunabhäniger Bytecode statt ~abhängiger Übersetzung
  - Bytecode wird lokal von einer "Virtual Machine" ausgeführt

- Sicherheit
  - Virtual Machine überwacht die Ausführung
  - verhindert ggf. unbefugten Zugriff auf die reale Maschine

- Schwerpunkte kein Zufall: der Fokus bei der Entwicklung der Sprache lag auf Plattform-heterogenen Umfeldern (Internet!)

- aus der Anfangszeit kommen die vielzitierten Applets, i.m.o. gescheitert im ersten Anlauf; vielleicht kommen sie nochmal?
  nettes Beispiel für Applets siehe http://www.map24.at

Philosophie von Java (2)

- einige Schlagwörter
  - einfach
  - sicher
  - portabel
  - objektorientiert
  - robust
  - architekturneutral (plattform~)
  - interpretiert
  - leistungsstark (?!)

Paradigmen in der Programmierung (1)

- Warum Evolution? Programme und Projekte werden immer komplexer – aktuelle Methoden stossen an ihre Grenzen

- nota bene: es geht nicht um Machbarkeit sondern um Verständlichkeit, Wartbarkeit, Wiederverwendbarkeit, Entkoppelung von Anforderungen usw...

Paradigmen in der Programmierung (2)

- objektorientierte Programmierung als vorläufige "Krönung der Evolution" der Paradigmen in der Programmierung:
  - Maschinensprachen
  - strukturierten Sprachen
  - prozedurale Sprachen
  - objektorientierte Sprachen
- ... was kommt als nächstes? AOP? Architekturen?

Grundkonzepte der OOP

- früher "datenverarbeitende Programme/Maschinen"
- heute Versuch der Abstraktion und Virtualisierung: Fokus liegt nicht mehr auf der Struktur des Codes, sondern auf der Struktur der Daten, der "Objekte des Problembereichs"
- 3 Grundkonzepte:
  - Kapselung (Encapsulation)
  - Vererbung (Inheritance)
  - Polymorphie (Polymorphism)

Kapselung

- Fachlich zusammengehörende Daten und darauf operierender Programmcode werden in einer logischen Einheit gekapselt.

- Komplexes Problem wird in kleinere Probleme zerteilt.
- Jede Einheit löst genau ein Problem, nämlich sein eigenes.

- Logische Einheit ist die "Klasse"
- eine Klasse besteht aus Attributen und Methoden
- diese Elemente können privat oder öffentlich sein

- Beispiel Auto:
  - öffentliches Attribut *color*, öffentliche Methode *stepOnGas()*
  - privates Attribut *cylinderCount*, private Methode *injectGas()*

Objekt ist Instanz einer Klasse

Vererbung

- Erstellen von Klassenhierarchien
- Vererbung von Attributen und Methoden einer Elternklasse auf Kindklassen vereinfacht das Schreiben von für die Problemstellung möglichst brauchbaren Klassen
- Im Detail:
  - Attribute und Methoden werden standardmäßig vererbt
  - das Überschreiben von Attributen und Methoden ist möglich
  - neue Attribute und Methoden können hinzugefügt werden
- Beispiel:
  Granny Smith <- Apfel <- Frucht <- Nahrungsmittel

Polymorphie

- Beispiel Lenkrad eines Autos:
  - der Lenker interagiert mit dem Auto über das Lenkrad
  - der tatsächlich implementierte Lenkmechanismus ist egal

- Beispiel Thermostat

Hands on!

- Voraussetzungen
  - Runtime Environment (JRE) versus Development Kit (JDK)
  - JRE, um Programme auszuführen
  - JDK, um Programme zu entwickeln (= JRE + Compiler, etc...)

- Der Reihe nach
  - Programm schreiben
  - Programm kompilieren
  - Programm ausführen

Schreiben

```
/*
    Dies ist ein einfaches Java-Programm.
    Speichern wir es unter "HelloWorld.java".
*/

class HelloWorld {

    // Ein Java Programm beginnt mit dem Aufruf von main()

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

- Erklärung später!

Kompilieren

- Compiler liest HelloWorld.java und erzeugt HelloWorld.class

- HelloWorld.class ist Bytecode der Klasse HelloWorld
- Bytecode ist nicht direkt ausführbar – er wird von der Virtual Machine ausgeführt

```
c:\Dokumente und Einstellungen\Benutzername> c:

c:\Dokumente und Einstellungen\Benutzername> cd c:\Java

c:\Java> javac.exe HelloWorld.java

c:\Java> java -classpath . HelloWorld
HelloWorld!

c:\Java>
```

Ausführen

- Bytecode wird vom Interpreter in einer VM ausgeführt:
  - Interpreter sucht Datei mit dem BC der angegebenen Klasse,
  - lädt die Klasse aus dieser Datei,
  - sucht darin nach einer statischen Methode "main(...)",
  - und führt sie aus; coole Sache.

```
c:\Dokumente und Einstellungen\Benutzername> c:

c:\Dokumente und Einstellungen\Benutzername> cd c:\Java

c:\Java> javac.exe HelloWorld.java

c:\Java> java -classpath . HelloWorld
HelloWorld!

c:\Java>
```

Zeile für Zeile

```
01: /*
02:    Dies ist ein einfaches Java-Programm.
03:    Speichern wir es unter "HelloWorld.java".
04: */
05:
06: class HelloWorld {
07:    // Ein Java Programm beginnt mit dem Aufruf von main()
08:
09:    public static void main(String[] args) {
10:       System.out.println("Hello World");
11:    }
12: }
```

Syntaxfehler

- Rechtschreibung und Zeichensetzung müssen richtig sein
- Java unterscheidet zwischen Gross- und Kleinschreibung

- Compiler meldet syntaktische Fehler im Quelltext
- Die Angaben sind als Erklärungs*versuche* zu verstehen

  (im Allgemeinen triffts der Compiler aber ganz gut und der eigentliche Fehler befindet sich zumindest in der Nähe der Stelle, wo's den Compiler wirft; Zeilenangabe beachten!)

Zweites Beispielprogramm

```
class Example2 {
    public static void main(String[] args) {
        int var1; // Variablendeklaration
        int var2;

        var1 = 1024; // var1 erhält den Wert 1024

        System.out.println("var1 hat den Wert " + var1);

        var2 = var1 / 2;

        System.out.print("var2 hat den Wert ");
        System.out.println(var2);
    }
}
```

Wichtige Datentypen (Auswahl!)

- int für ganze Zahlen: 1, -12, 4232, ...
- double für Gleitkommazahlen: 1.24, -400.123, ...
- boolean für boolesche Werte: true, false
- String für Zeichenketten: "Oi! Oi!", ...

```
int i = 17;
double d = 1.24;
boolean b = false;

String s = "Servus";
String t;

d = d / i;    // 0.07294117647058823
t = s + d;    // "Servus0.07294117647058823"
i = d * 2;    // kompiliert nicht
d = i - b;    // kompiliert nicht
```

Steueranweisungen

- Anweisungen einer Methode werden d. Reihe nach ausgeführt
- spezielle Steueranweisungen beeinflussen den "Kontrollfluss"

- wichtige Steueranweisungen zum Beispiel:
  - die if-Anweisung
  - die for-Schleife

- (weitere Steueranweisungen im Einführungskurs!)

die if-Anweisung

- Form: if (Bedingung) Anweisung;
- die Bedingung muss sich zu einem booleschen Wert (also *true* oder *false*) auswerten lassen
- Vergleichsoperatoren: >, >=, <, <=, !=, ==
- logische Operatoren: & (und), | (oder), ! (nicht)

```
if (10 < 12) System.out.println("Converge");
if (12 == 1) System.out.println("Jane");

boolean b = true;
if (b) System.out.println("Doe");
if (b & false) System.out.println("At");
if (!false)) System.out.println("The");
if (!(1 < 2)) System.out.println("Gates");
```

die for-Schleife

- Form: for (Initialisierung; Bedingung; Iteration) Anweisung;

- Ablauf:
  1. Initialisierung wird ausgeführt
  2. falls Bedingung wahr: Anweisung wird ausgeführt; sonst Ende
  3. Iteration wird ausgeführt
  4. gehe zu 2.

```
int i;

// Zahlen von 0 bis 9 ausgeben
for (i = 0; i < 10; i = i + 1) System.out.println(i);

// Problematisch, warum?
for (i = 0; i < 10; i = 5) System.out.println(i);
```

Codeblöcke

- kapselt ein oder mehrere Anweisungen in logischer Einheit
- Anweisungen werden in { und } eingeschlossen
- kann verwendet werden, wo eine Anweisung erwartet wird
- beispielsweise als Ziel einer if- oder for-Anweisung

```
// Codeblock mit 2 Anweisungen
if (x < 10) {
    x = x + 10;
    System.out.println("Wert von x wurde um 10 erhöht.");
}

// Codeblock mit nur einer Anweisung
// (überflüssig, aber besser lesbar)
if (x < 10) {
    System.out.println("Wert von x ist kleiner als 10.");
}
```

Codeblöcke und Gültigkeitsbereich von Variablen

- Nota bene: Variablen sind ausschliesslich innerhalb des Codeblocks/der Methode gültig, in dem/der sie deklariert wurden!

```
public static void main(String[] args) {
    int i = 17;
    System.out.println("i hat den Wert " + i);

    if (i > 12) {
        int j = 3;
        i = i + j;
        System.out.println("i hat jetzt den Wert " + i);
    }

    // Fehler: die Variable j ist hier nicht definiert
    System.out.println("j hat den Wert " + j);
}
```

Zusammenfassendes Beispiel

```
class Fahrenheit {
    public static void main(String[] args) {
        boolean foundSameValue = false;
        double sameValue = 0;

        for (int c = -100; c <= 100; c = c+1) {
            double f = ((9.0/5.0) * c) + 32;
            System.out.print(c + " Grad Celsius sind ");
            System.out.println(f + " Grad Fahrenheit.");

            if (c == f) {
                foundSameValue = true;
                sameValue = c;
            }
        }

        if (foundSameValue) {
            System.out.println("C. und F. gleich bei: " + sameValue);
        }
    }
}
```