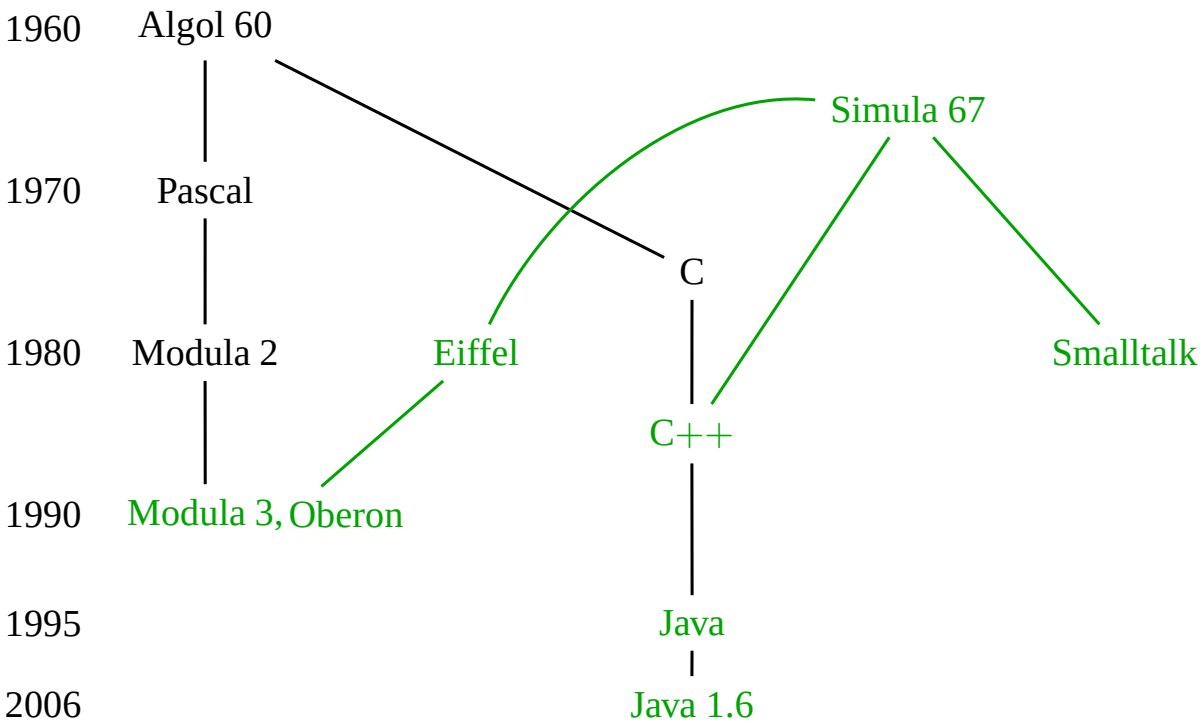
Abschnitt 1: Einführung

1. Einführung
1.1 Historischer Überblick: Objektorientierte Programmiersprachen
1.2 Java – Erste Schritte
1.3 Kommentare in Java

Überblick

1. Einführung
1.1 Historischer Überblick: Objektorientierte Programmiersprachen
1.2 Java – Erste Schritte
1.3 Kommentare in Java

Entwicklung objektorientierter Programmiersprachen



Algol 60 und Pascal

Algol 60

- Entwickelt von John Backus, F.L. Bauer, John McCarthy, Peter Naur, Alan J. Perlis, Heinz Rutishauser, Klaus Samelson.
- Imperative Sprache mit Blockkonzept, Call-by-Value und Call-by-Name, Syntaxdefinition in Backus-Naur-Form


F.L. Bauer, *1924, Diss. 1953 LMU, Entwickler von Algol 60, Patent auf Kellerprinzip, “Vater” der dt. Informatik


N. Wirth, *1931, PhD 1963 in Berkeley, Entwickler von Pascal, Modula, Oberon, Turing-Preis 1984

Pascal

- Entwickelt von Niklaus Wirth
- Imperative Sprache, Weiterentwicklung von Algol 60/68, mit Verbunddatentypen (Record)

Simula 67 und Smalltalk 80

Simula 67

- Entwickelt von Kristen Nygaard und Johan-Ole Dahl
- Erste OO-Sprache, Erweiterung von Algol 60 mit Klassen-Konzept, Vererbung (damals genannt “Prefixing")

J.O. Dahl, 1931-2002, Entwickler von Simula 67, 1. Informatik-professor in Norwegen (1968), Turing-Preis 2001

A. Kay, *1940, PhD 1969 Utah, Entwickler von Smalltalk, Dynabook (1.PC), Turing-Preis 2003

Smalltalk 80

- Entwickelt von Alan Kay und Adele Goldberg
- 1. dynamische OO-Sprache, mit Betriebssystem und Entwicklungsumgebung

Java und C++

Java

- Entwickelt von J. Gosling, Bill Joy, P. Naughton, u.a.
- Erste plattform-unabhängige OO-Sprache, insbesondere zur Programmierung von Internet-Applikationen
- Heute auch eingesetzt im Multimedia-Bereich und (beginnend) für eingebettete Systeme
- Erste Version 1.0 1995, heute Java 1.6 (auch: 6.0)
- Ursprünglicher Name: OAK

James Gosling, *1956, PhD 1983 CMU, Entwickler von Emacs, Java

B. Stoustrup, *1950 in Arhus (DK), PhD 1979 in Cambridge, Entwickler von C++

C++

- Entwickelt von Bjarne Stoustrup
- Effiziente OO-Sprache, Erweiterung von C

Überblick

1. Einführung
1.1 Historischer Überblick: Objektorientierte Programmiersprachen
1.2 Java – Erste Schritte
1.3 Kommentare in Java

Aspekte von Java

- objektorientiert: Klassenkonzept, strenge Typisierung
- plattformunabhängig: Übersetzung in Virtuelle Maschine (JVM)
- netzwerkfähig, nebenläufig
- Sicherheitsaspekt in der Entwicklung der Sprache wichtig

Nachteile:

Laufzeithandicap durch Interpretation (JVM), wird aber stetig verbessert

Vorteile:

- Plattformunabhängigkeit
- verteilte Anwendungen, Web-Anwendungen
- Rechnerunabhängigkeit von Graphikanwendungen

Grober Aufbau eines Java-Programms

- Ein Java-Programm besteht aus Klassen und Schnittstellen
- Eine Klasse besteht aus
  - Klassenvariablen: beschreiben Eigenschaften aller Objekte dieser Klasse.
  - Attributen (fields, Instanzvariablen): beschreiben den Zustand eines Objekts.
  - statischen Methoden: Prozeduren einer Klasse, unabhängig vom Zustand eines Objekts
  - Objekt-Methoden: Operationen, die ein Objekt ausführen kann, abhängig vom Zustand des Objektes
  - Konstruktoren: Operationen zur Erzeugung von Objekten einer bestimmten Klasse

Wir betrachten nun zunächst nur die statischen Elemente. Zu den Aspekten der Objektorientierung kommen wir zu einem späteren Zeitpunkt der Vorlesung.

Ein einfaches imperatives Java-Programm

Ein einfaches imperatives Java-Programm besteht aus nur einer Klassendeklaration und einer Methode “main”:

```
public class KlassenName
{
  public static void main(String[] args)
  {
    // Anweisungen ...
  }
}
```

Die Textdatei, die den Java-Code enthält, heißt `KlassenName.java`, also genauso wie die enthaltene Klasse, mit der Endung `java`.

Beispiel: `HelloWorld.java`

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


Konventionen

In der Java-Programmierung gibt es einige Konventionen. Deren Einhaltung erleichtert das Lesen von Programmen. Beispiele solcher Konventionen sind:

- Klassennamen beginnen mit großen Buchstaben.
  - `HelloWorld`
- Methodennamen, Attributnamen und Variablennamen beginnen mit kleinen Buchstaben.
  - Methoden: `main`, `println`,
  - Klassenvariable: `out`,
  - Variable: `args` (weder Instanz- noch Klassenvariable, sondern Parametervariable)
- Zusammengesetzte Namen werden zusammengeschrieben, jeder innere Teilname beginnt mit einem großen Buchstaben.
  - Klasse `HelloWorld`

Plattformunabhängigkeit

- Bei vielen Sprachen (z.B. C/C++) erzeugt der Compiler plattformabhängigen Maschinencode (kann nur auf bestimmten Rechnerarchitekturen/Betriebssystemen ausgeführt werden).
- Sogenannte Skript-Sprachen (z.B. Perl, PHP, auch SML) werden interpretiert von einem plattformspezifischen Interpreter – die Programme sind plattformunabhängig, aber der Sourcecode bleibt unübersetzt und sichtbar, was in vielen Anwendungen nicht erwünscht ist.
- Plattformunabhängigkeit eines Java-Programmes wird durch einen Kompromiß erreicht:
  - Der Sourcecode wird übersetzt in Bytecode, der plattformunabhängig verwendet werden kann.
  - Bytecode wird von einer virtuellen Maschine ausgeführt (interpretiert).
  - Die virtuelle Maschine gibt es in verschiedenen Versionen für verschiedene Plattformen (JVM = Java Virtual Machine, Teil des JRE = Java Runtime Environment).

Übersetzung in Bytecode

- Aus einer Textdatei `KlassenName.java` erzeugt der Java Compiler `javac` eine Binärdatei `KlassenName.class`.
  Beispiel:

  ```
  javac HelloWorld.java
  ```

  erzeugt die Binärdatei `HelloWorld.class`.
- Die Binärdatei `KlassenName.class` enthält den Bytecode für die JVM.
- Der Compiler `javac` ist Teil des JDK (= Java Development Kit). Das JDK enthält JRE, Sie benötigen also das JDK für die Übungen zu dieser Vorlesung.

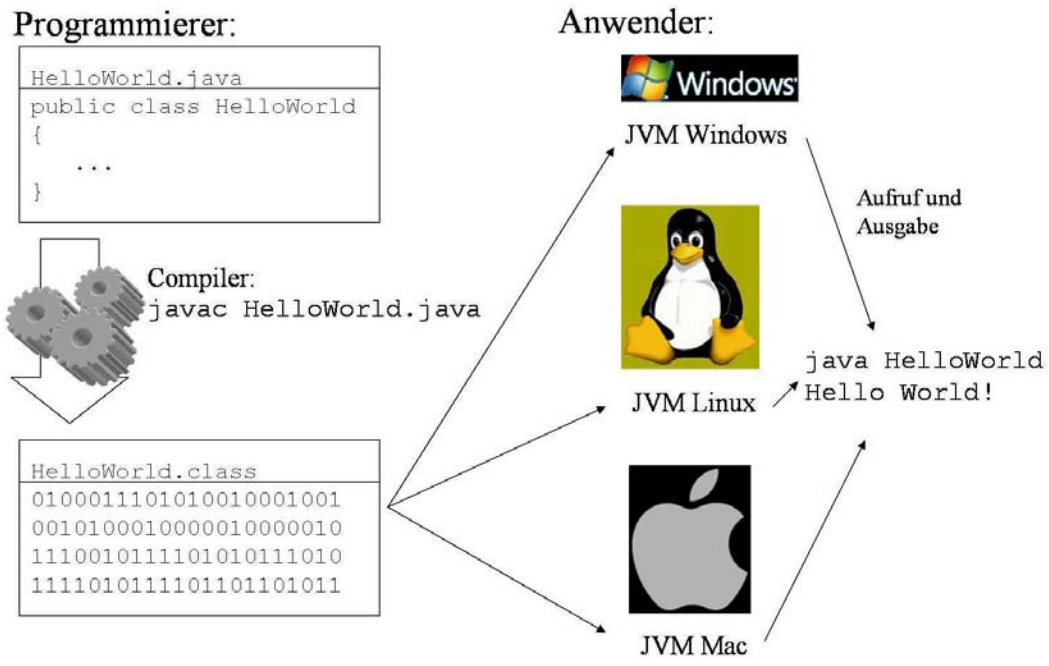
Ausführen des Programms

Die Binärdatei `KlassenName.class` wird der JVM übergeben und von dieser ausgeführt (interpretiert). Durch den Aufruf `java KlassenName` wird die `main`-Methode der Klasse `KlassenName` aufgerufen.
Beispiel:

```
java HelloWorld
```

gibt “Hello, World!” auf dem Bildschirm aus.

Übersetzung und Ausführung



Überblick

1. Einführung
1.1 Historischer Überblick: Objektorientierte Programmiersprachen
1.2 Java – Erste Schritte
1.3 Kommentare in Java

Prinzip: Programmcode und Kommentar gehören zusammen

"The view that documentation is something that is added to a program after it has been commissioned seems to be wrong in principle, and counterproductive in practice.

Instead, documentation must be regarded as an integral part of the process of design and coding."

C. A. R. Hoare (Turing-Preisträger):
Hints on Programming Language Design, 1973

C. A. R. Hoare, *1934
Erfinder von Quicksort, Hoare Logik, Strukt. Programmierung, CSP, Occam
Turing-Preis 1980

Verschiedene Arten von Kommentaren

Es gibt in Java 3 Arten von Kommentaren:

- Einzeilige Kommentare beginnen mit `//` und enden am Ende der aktuellen Zeile
- Mehrzeilige Kommentare beginnen mit `/*` und enden mit `*/`
- Dokumentationskommentare beginnen mit `/**` und enden mit `*/` und können sich ebenfalls über mehrere Zeilen erstrecken.

Kommentare derselben Art sind nicht schachtelbar. Ein Java-Compiler akzeptiert aber einen einzeiligen innerhalb eines mehrzeiligen Kommentars und ungekehrt.

Dokumentationskommentare dienen dazu, Programme im Quelltext zu dokumentieren. Sie werden in den mit dem Befehl `javadoc` erzeugten Report mit aufgenommen.

Beispiel: Die Klasse `HelloWorld` dokumentiert

```
/**
 * HelloWorld Klasse um eine einfache Benutzung einer java-Klasse zu illustrieren.
 *
 * Diese Klasse dient nur dem Anzeigen des Strings
 * "Hello, world!" auf dem Bildschirm
 *
 * @author Arthur Zimek
 */
public class HelloWorld
{

    /**
     * Die main-Methode wird automatisch aufgerufen, wenn die Klasse mit
     * <code>java HelloWorld</code>
     * aufgerufen wird.
     *
     * Die Methode main druckt "Hello, world!" auf die Standard-Ausgabe.
     *
     * @param args Array mit Parametern - wird von dieser Methode nicht verwendet.
     */
    public static void main(String[] args)
    {
        // Ausgabe von "Hello World!" auf die Standard-Ausgabe
        System.out.println("Hello World!");
    }

}
```

Erzeugen der Dokumentation

Mit dem Befehl

```
javadoc HelloWorld.java
```

wird automatisch eine Beschreibung der Klasse `HelloWorld` erzeugt und in die Datei

```
HelloWorld.html
```

geschrieben.
Die Klassenbeschreibung wird eingebettet in eine organisierte Menge von html-Dokumenten:

`http://www.dbs.ifi.lmu.de/Lehre/Info2/SS07/skript/programmbeispiele/einfuehrung/helloWorld/index.html`

Diese Dokumentation kann auch für viele Klassen gleichzeitig erfolgen (`javadoc *.java`).

Spezielle Variablen (tags) für javadoc

Die durch @ eingeleiteten Elemente in einem Dokumentationskommentar haben eine besondere Bedeutung, z.B.:

- `@see` für Verweise
- `@author` für Name des Autors / Namen der Autoren
- `@version` für die Version
- `@param` für die Methodenparameter
- `@return` für die Beschreibung des Ergebnisses einer Methode

Java Standard Edition API Dokumentation

Auch die Bibliothek der Standard Edition ist mit javadoc erzeugt:
`http://java.sun.com/javase/6/docs/api/`
Für das fortgeschrittene Programmieren mit Java ist diese API Doc ein sehr wichtiges Hilfsmittel.

Zusammenfassung

Sie können jetzt:

- ein einfaches, imperatives Java-Programm schreiben,
- ein Java-Programm in Bytecode übersetzen,
- ein kompiliertes Java-Programm ausführen,
- ein Java-Programm dokumentieren.