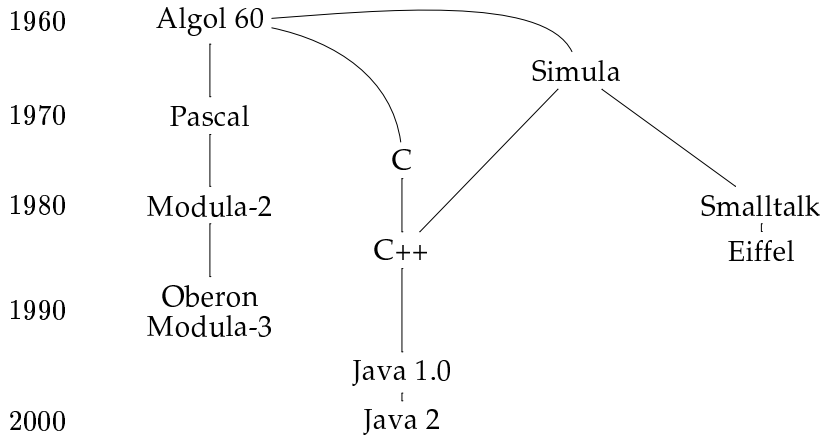
KAPITEL 2

Einführung in die Programmierung mit Java

Ziele:

- Geschichte der OO-Programmiersprachen
- Warum Java als Programmiersprache verwenden?
- Ein einfaches Java-Programm erstellen, übersetzen und ausführen
- Gut dokumentierte Java-Programme erstellen können

2.1 Entwicklung objektorientierter Programmiersprachen



Simula: erste OO-Sprache, entwickelt von den Norwegern O. Y. Dahl und K. Nygaard (Oslo) ab 1962

Smalltalk: entwickelt von Alan Kay, untypisierte OO-Sprache mit Betonung auf Objekten und deren Verstrickung mit Symbolen ab 1972, stabil ab 1980

C++: Weiterentwicklung von C durch Einführung objektorientierter Konzepte, konzipiert von B. Stroustrup (AT&T), 1982–85, Normierung seit 1988

Oberon: objektorientierte Weiterentwicklung von Modula 2, entwickelt von N. Wirth ($\sim$ 1987)

Modula 3: objektorientierte Weiterentwicklung von Modula 2 am DEC SRC, konzipiert als Systemprogrammierspache. Entwickler waren Luca Cardelli, Greg Nelson, Jim Donahue ... Erste Implementierung 1988, stabil ab 1990

Eiffel: erste objektorientierte Sprache, der ein systematisches Programmentwicklungskonzept zugrunde liegt und die z.B. Vor- und Nachbedinungen und Invarianten in die Sprache integriert, „Programming by contract“, entwickelt von Bertrand Meyer ab 1985

2.2 Java

Das Wort „Java“ hat insgesamt drei Bedeutungen:

- Insel in Süd-Ost-Asien
- Amerinkanisches, insbedondere in Californien verbreitestes Synonym für starken Bohnenkaffee
- Programmiersprache

Entwickelt wurde Java von J. Gosling und P. Naughton mit dem Ziel elektronische Geräte wie zum Beispiel Viedeorecorder zu programmieren. Aufgrund von wirtschaftlichen Gründen wurde dieses Ziel nie richtig erreicht und so erfolgte 1993 eine Abänderung der Zielsetzung. Java entwickelte sich zur ersten plattform-unabhängige OO-Sprache, die sich besonders gut zur Programmierung von Internet-Applikationen eignet, wodurch auch Sicherheitsaspekte verstärkt in den Vordergrund traten.

Die Version 1.0 kam 1995 heraus; Java 1.2 wurde umbenannt in Java 2. Die aktuelle Version ist Java 1.3. Der ursprüngliche Name war OAK.

2.2.1 Aspekte von Java

Objektorientiert: Außer Basis-Datentypen wie Zahlen und Zeichen ist alles in Java ein Objekt. Es gibt keine Mehrfachvererbung für Implementierungen, wohl aber für Schnittstellen.

Unabhängig von Plattform: Der Java-Bytecode kann ähnlich wie ein Bild unabhängig von der speziellen Rechnerplattform benutzt werden. Zum Ablauf wird ein Interpreter bzw. Compiler benötigt.

Interpretierend (p-Code): Ein Compiler erzeugt den sog. Java-Bytecode. Dieser ist eine symbolische Sprache, die unabhängig vom Rechner ist. Der Bytecode kann bei Bedarf auch compiliert werden. Sogenannte Just-In-Time-Compiler übersetzen den Bytecode beim Ladevorgang in die Maschinensprache. Die Bezeichnung p-Code steht für Postfix-Code. Bei dieser Darstellung werden zunächst die Operanden, dann die Operationen abgespeichert. Beispiel: aus `a+b*c` wird in Postfix `a b c * +`

Netzwerkfähigkeit: Zugriff auf Funktionen der Browser, Programmierung von Sockets, verteilte Anwendungen können aufgebaut werden.

Sicherheit: Viren oder trojanische Pferde würden eine Übertragung von Java-Programmen äußerst gefährlich machen. Denn anders als bei der Datenübertragung per Diskette werden unzählige Programme aus unzähligen Quellen mit enormer Geschwindigkeit weltweit verteilt. Deswegen wurden für Applets Sicherheitsmodelle in das Konzept integriert.

Threads: Java unterstützt von Haus aus parallele Abläufe. Zum Transfer von Bildern o.ä. ist dies im Internet auch nicht anders vorstellbar. Zur Synchronisierung wurde das Monitor-Konzept von C. A. R. Hoare gewählt. Damit könnten die Programmierfehler im Zusammenhang mit Threads minimiert werden.

Modularer Aufbau: Problemlose Verteilung von Code auf mehrere Dateien ohne Header-Files. Die Konsistenz der Module wird direkt gegen den tatsächlichen Java-Bytecode geprüft.

Graphik: GUI's (Graphical User Interface) unabhängig von der Plattform, Vektorgraphik unabhängig von der Plattform.

Anbindung von Routinen anderer Programmiersprachen: Wird ausdrücklich unterstützt, ist aber speziell für die in Browser eingebundenen Applets aus Sicherheitsgründen unmöglich.

2.2.2 Was ist neu an Java?

Unabhängigkeit von der Plattform: Java ist eine Programmiersprache, die in ihren elementaren Daten und Kontrollstrukturen auf C zurückgreift. Die objektorientierten Konstrukte werden Kennern von C++ leicht in der Anwendung fallen. Im Gegensatz zu C bzw. C++ wurde Java vollständig spezifiziert, z.B. ist das Ergebnis der modulo-Operation in Java unabhängig von der Plattform, auch beim Schieben von Bits nach rechts gibt es keine Überraschungen mehr bei der Portierung. Auch bei der Bewertung von Ausdrücken wurden die Spielräume der Compilerbauer auf ein Minimum beschnitten.

Sicherheit: Da Programmabstürze bekanntlich ein offenes Scheunentor für Angriffe von Eindringlingen im Internet darstellen, mußten sie eliminiert werden. Dies war

nur erreichbar, indem für Programmierer der Zugriff auf Zeiger eingeschränkt wurde. Natürlich kann man in Java mit Zeigern programmieren, man kann sie aber weder verbiegen, noch den Freispeicher korrumpieren. Daher überrascht es nicht, daß Java einen Speicherbereinigungsalgorithmus („Garbage Collection“) benutzt, der nicht mehr erreichbare Daten automatisch löscht und die Speicherzellen wieder freigibt. Das Sicherheitskonzept schließt auch Überprüfungen zur Laufzeit ein: Datentypen, natürlich Indizes, Casting. Diese Konzepte fußen auf Mechanismen zur Verifizierung von Java-Bytecode bei der Übertragung. Erst auf diese elementaren Sicherheitsmechanismen können dann in Java implementierte Zugriffsbegrenzungen für Schreiben und Lesen von über das Internet verteilten Anwendungen sicher gegründet werden. Dennoch bleibt ein Restrisiko zu beachten, wie die Diskussion im Internet zeigt. Jeder Anwender von Java-Programmen muß hier seine persönliche Strategie für eine optimale Sicherheit finden.

2.2.3 Vor- und Nachteile

Nachteile: Als interpretierte Sprache mit eingebauten hohen Sicherheitsanforderungen muß Java mit einem Laufzeit-Handicap leben. Zum Beispiel ergibt sich auf Solaris gegenüber C bzw. C++ ein Faktor von 15[1], d.h. ein Java-Programm reduziert die Rechnerleistung auf ca. 6,66%. Die Wintel-Allianz ist so dominierend, daß der Stellenwert einer plattformunabhängigen Programmierung reduziert wird. Anwendungen für Benutzeroberflächen erfordern eine hohe Leistung auf der Seite der Systemschnittstelle und auf der Seite der Werkzeuge.

Vorteile: Aber viele Anwendungen im Internet – und im Intranet – können mit der relativ hohen Performanz von Java ausgeführt werden. Just-In-Time Compiler werden Java-Bytecode während des Ladevorgangs übersetzen, wodurch die Nachteile bei der Laufzeit stark reduziert werden. Administrationsprobleme in Client-Server-Umgebungen etwa bei der Distribution von Software können reduziert werden. Auch die Programmierung von Graphiken wird unabhängig vom Rechner. Web-Seiten lassen sich auch bei der beschränkten Bandbreite des Netzes interaktiv gestalten. Mit CORBA-Implementierungen, z.B. von Sun im JDK mit Jini und RMI („Remote Method Invocation“), erhält Java einen „fliegenden Teppich“: verteilte Anwendungen werden ohne aufwendige Programmierung von Sockets möglich.

2.3 Der grobe Aufbau eines Java-Programmes

Grundsätzlich unterscheiden wir zwei Arten von Java-Programmen:

- selbsständige Programme
  Sie benötigen eine Klasse mit einer Methode `main` und werden von einer Konsole („shell“) gestartet

[1] 1997. Durch Optimierung der Java-Interpreter hat sich in letzter Zeit die Performanz von Java-Anwendungen verbessert

- unselbständige Programme
  Diese werden als Byte-Code über das Internet übertragen und durch einen Browser gestartet.

Ein Java-Programm besteht jedoch immer aus einer Menge von Klassen. Jede Klasse enthält zwei Arten von Bestandteilen, nämlich *Attribute* („fields“) und *Methoden* („methods“).

Die Attribute beschreiben den *Zustand* einer Klasse bzw. eines Objekts (Instanz einer Klasse). Die Methoden beschreiben die *Operationen*, die ein Objekt ausführen kann. Sie bestehen aus Anweisungen, die den Zustand (der Attribute) verändern.

Technisch besteht ein (einfaches, imperatives) Java-Programm aus einer Klassendeklaration mit einer einzigen Methode `main`:

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

Dabei sind `<KlassenName>` und `<Anweisungen>` Platzhalter. **`public`** und **`static`** sind sogenannte „Modifier“ für den Zugriff auf den vereinbarten Namen, **`void`** bezeichnet den Resultattyp von `main` und bezeichnet das 0-elementige Tupel, das nur das leere Tupel () als Element beinhaltet, d.h. `main` liefert ein triviales Resultat.

`String[]` bezeichnet den Parametertyp (ein Feld von Worten) und `args` den Namen des (formalen) Parameters. Normalerweise benötigen wir diese Argumente nicht.[2]

In Beispiel 2.3 ist `System.out.println("Hallo, Welt!");` die einzige Anweisung. Sie stellt einen Methodenaufruf dar, wobei `out` Objekt der Klasse `System`, `println` Methode der Standardklasse `System` und `"Hallo, Welt!"` der aktuelle Parameter ist. Die Methode `println` druckt den String `"Hallo, Welt!"` auf den Bildschirm gegeben durch das Standardobjekt `out` der Klasse `System`. Um dieses Objekt zu benutzen, muß auch der Klassenname angegeben werden: `System.out`. `println` macht am Ende einen Zeilenvorschub („`ln`“), man kann auch Zahlen drucken.

Methodenaufruf

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

Aufruf der Methode `methodName` des Objekts `object` und den aktuellen Parametern `parameters`. Durch den anschließenden Strichpunkt wird die Methode zur Anweisung.

Bemerkungen: Namen (bzw. Identifikatoren) dienen als frei gewählte Bezeichner für Klasse, Attribute, Methoden etc.

[2] Die Argumente dienen zur Eingabe von der Konsole (siehe später)

Beispiel 2.1 Methodenaufruf

```
System.out.println(3+4);
```

Ein Name beginnt mit einem (großen oder kleinen) Buchstaben gefolgt von endlich vielen Buchstaben, Ziffern, Unterstrich und $, also zum Beispiel:

`Wirsing`, `Wirsing_Martin`, `WS_96_97`,

keine Namen sind: `1more`, `3$`, `a:b`, `wirsing@de`

Konventionen: selbstgewählte Klassennamen beginnen mit großen Buchstaben, Methodennamen, Variablennamen mit kleinen Buchstaben, Konstantennamen bestehen nur aus großen Buchstaben.

Eine Reihe von Namen sind vordefiniert und reserviert („Schlüsselwörter“) und dürfen nur in ihrer intendierten Bedeutung benützt werden. Beispiele dafür sind **`class`**, **`public`**, **`void`**, **`static`**, ...

2.4 Erstellung von Java-Programmen

Um ein Java-Programm zu erstellen und auszuführen, schreibt man es zunächst in eine Textdatei und geben ihr einen Namen der Form `Name.java`. Für unser Beispiel 2.3 auf der vorherigen Seite ist dies

```
Hallo.java
```

(Der Name muß mit dem Klassennamen übereinstimmen, der die Methode `main` enthält). Dann wird der Übersetzer mit dem Befehl

```
javac Hallo.java
```

aufgerufen. Ist das Programm syntaktisch korrekt, wird der Quellcode aus `Hallo.java` in sogenannten Byte-Code übersetzt, der in der Datei `Hallo.class` gespeichert wird. Der Byte-Code enthält Maschinencode des Programms sowie weitere Informationen, z.B. wie das Programm in den Speicher zu laden ist. Mit dem Kommando

```
java Hallo
```

wird der Java-Interpreter aufgerufen, der den Byte-Code aus `Hallo.class` und die notwendigen Bibliotheksinformationen lädt und dann das Programm ausführt. In unserem Beispiel wird der String `"Hallo, Welt!"` auf den Bildschirm gedruckt.

Eine weitere Möglichkeit ist, ein Java-Programm in einer eingebetteten Anwendung in einem Internet-Browser als sog. „Applet“ ablaufen zu lassen (siehe später).

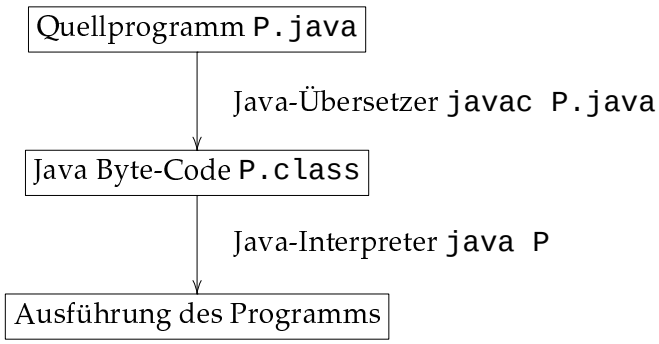


Abbildung 2.1: Selbständig laufende Anwendung

2.5 Kommentare in Java

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,
Hints on Programming Language Design (1973)

Es gibt zwei Darstellungen für Kommentare in Java.

1. Durch

   ```
   // bla, bla
   ```

   wird eine Zeile oder ein Rest einer Zeile zum Kommentar.

2. Zur Erzeugung von Kommentaren zu Klassen und Methoden werden die Klammern

   `/**` und `*/`

   verwendet. Solche Kommentare werden in den mit dem Befehl `javadoc` erzeugten Report mit aufgenommen.

Der erste Kommentar in Beispiel 2.2 auf der nächsten Seite erklärt die Aufgabe der Klasse `Hallo`, der zweite Kommentar beschreibt die Wirkung der Methode `main`.

Es gehört zu einem guten Programmierstil, jede Klasse und jede Methode zu kommentieren. Dabei ist es vorteilhaft, die Klammern (`/**`, `*/`) zu verwenden, da diese robust gegenüber Änderungen sind.

Mit dem Befehl

Beispiel 2.2 Hallo Welt – dokumentiert

```
/**
    Diese Klasse dient nur zum Ausdrucken des Strings "Hallo, Welt!"
    auf den Bildschirm
 */
public class Hallo
{
    /**
        Die Methode main...
     */
    public static void main (String[] args)
    {
        System.out.println("Hallo, Welt!");
    }
}
```

```
javadoc Hallo.java
```

wird automatisch eine Beschreibung der Klasse `Hallo` erzeugt und in die Datei `Hallo.html` geschrieben. In der Java-Dokumentation gibt es mehrere spezielle Variablen, die mit einem @-Zeichen gekennzeichnet werden:

@see	für Verweise
@author	für Namen des Autors
@version	für die Version
@param	für die Methodenparameter

Ein Beispiel zu diesen Variablen ist in Beispiel 2.3 auf der nächsten Seite zu finden.

2.6 Zusammenfassung

1. Objektorientierte Programmiersprachen haben eine lange Tradition beginnend mit Simula 1967. Ende der 80er Jahre wurde die OO-Programmierung populär mit Smalltalk und C++. Heute sind C++ und Java die kommerziell vorherschenden Programmiersprachen.

2. Java ist eine moderne OO-Programmiersprache, die vor allem zur Programmierung im Internet eingesetzt wird. Java ist interpretierend, plattformunabhängig, unterstützt Sicherheitskonzepte und besitzt eine reichhaltige API („Application Programming Interface“, Klassenbibliothek).

3. Ein Java-Programm besteht aus einer oder mehreren Klassen. Klassen enthalten die Definitionen von Methoden. Eine Methode besteht aus einer Sequenz von

Beispiel 2.3 Hallo Welt Applet – erweitert dokumentiert

```
/**
    Diese Klasse ist die Applet-Version von Hallo zur Demonstration von
    javadoc
    @see java.applet.Applet
    @author Martin Wirsing
    @version 1.1
 */
public class HalloApplet extends Applet
{
    /**
        Diese Methode dient nur zur Illustration der Parameterbehandlung
        durch javadoc.
        @param value ist ein Eingabeparameter
     */
    public void m (int value)
    {
        ...
    }
}
```

Anweisungen, die den Berechnungsablauf festlegen. Jede selbstablaufende Java-Anwendung enthält eine Methode `main`.

4. Ein Java-Programm wird mit einem Übersetzer in Byte-Code übersetzt, der dann mit einem Interpreter ausgeführt wird.

5. Java-Programme sollten gut dokumentiert werden. Mit `javadoc` kann automatisch eine übersichtliche Dokumentation erzeugt werden.