Java Tutorium WS01/02

Helge Janicke

21. Oktober 2001

Dieses Tutorium ist in mehrere Abschnitte unterteilt, die sich mit Aspekten der Sprache Java, Konzepten der Objektorientierung, sowie Grundlagen der Programmierung in Java beschäftigen.
In einigen Teilen wird die Theorie zum objektorienten Programmieren überwiegen (z.B. der erste Teil), andere werden sich mehr mit der Umsetzung von prozeduralen Elementen (Nassi-Schneidermann) in Java beschäftigen. Ein Teil handelt von den Schlüsselwörtern der Sprache Java, sowie von Operatoren.
Nach diesen Grundlagen wird sich noch ein Teil mit weiteren Konzepten der Objektorientierung beschäftigen (Interfaces, Abstrakte Klassen, Überladen von Methoden etc.). Dieser Teil wird allerdings erst gegen Ende der Veranstaltung behandelt, da er ein Grundverständis und die sichere Anwendung von Begriffen und Compiler voraussetzt.

Ein wichtiger Punkt bei diesem Tutorium ist, daß es auch kleinere Hausaufgaben geben wird. Wir bitten Euch, diese auch wirklich zu machen, da wir in den folgenden Veranstaltungen weiter darauf aufbauen. Eine weitere Bitte ist die regelmäßige Teilnahme, da es uns die Arbeit erleichtert und Euch einen schnellen Einstieg in das Programmieren mit Java ermöglicht.

Ansonsten freuen wir uns, Euch beim Tutorium zu sehen.

Helge Janicke, Niels-Peter de Witt.

Dieses Script basiert auf dem JAVA Tutorial von Sun. Einige Grafiken wurden aus dem Tutorial entnommen, um die Beispiele zu verdeutlichen. Das Tutorial ist frei erhältlich bei `java.sun.com`.

Inhaltsverzeichnis

1 Teil 1: Einführung in JAVA

1.1 Allgemein

Der folgende Abschnitt beschäftigt sich mit einigen grundlegenden Aspekten der Programmiersprache Java.

1.1.1 Was ist JAVA?

Java ist eine relativ neue objektorientierte Programmiersprache. Die Syntax von Java ähnelt der Syntax von C, so daß Programmierer, die schon Erfahrung mit C haben, leicht auch Programme in Java schreiben können.
Der wesentliche Unterschied besteht in der Objektorientierung. Diese bietet Möglichkeiten, reale Probleme leicht in ein Programm umzusetzen und bereits vorhandene Problemlösungen wiederzuverwenden.
Java-Programmierer können auf ein weites Umfeld von Klassenbibliotheken zurückgreifen, die es ihnen ermöglichen, in relativ kurzer Zeit auch größere Projekte umzusetzen.

1.1.2 Die Virtual Machine

Java-Quellcode wird beim Übersetzen nicht in Maschinencode umgewandelt. Stattdessen übersetzt der Java Compiler den Quellcode in ein Maschinenunabhängiges Format, den Java-Bytecode. Dieser Bytecode kann nun von einem Interpreter, der Virtual Machine (VM), ausgeführt werden. Damit ist es möglich, Programme zu schreiben, die auf einer Vielzahl von verschiedenen Systemen lauffähig sind. Die einzige Voraussetzung ist, daß für dieses System eine Virtual Machine verfügbar ist.

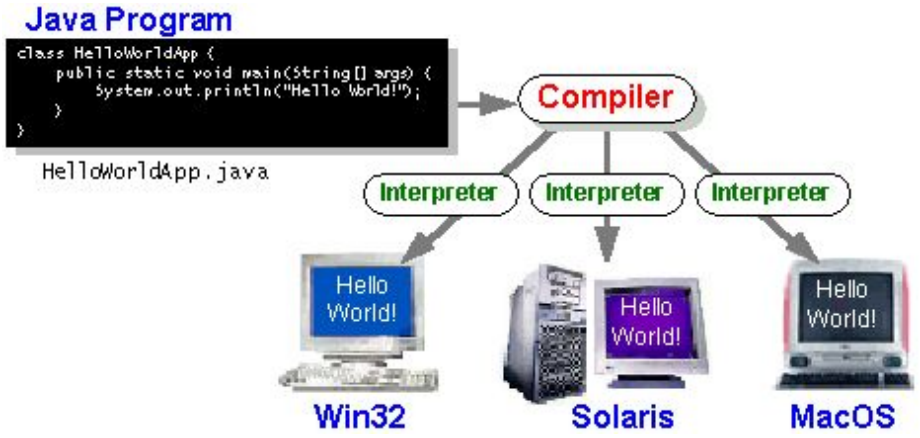

Abbildung 1: Die Virtual Machine [helloWorld.jpg]

1.1.3 Vorteile und Nachteile von Java

Im vorangegangenen Abschnitt wurde die Plattformunabhängigkeit von Java-Programmen angesprochen. Dies ist natürlich einer der größten Vorteile, die

Java zu bieten hat ("write once, run anywhere"). Aber auch andere Vorteile bieten sich durch die Verwendung von Java:

- Java unterstützt im hohen Maße die Netzwerkprogrammierung.
- Es gibt sehr umfangreiche Bibliotheken, die frei verfügbar sind (im JDK1.2 waren es 1520 Klassen in 59 Paketen)
- Die eingebaute Speicherverwaltung (Garbage Collection) schützt vor "Speicherfressern" (memory leaks). Die Speicherverwaltung wird vollständig vom Java System übernommen, man muß also nicht mehr verwendeten Speicher nicht explizit freigeben.
- Java ist eine sichere Sprache, viele Konzepte (z.B. die Verwendung von Entwurfsmustern) schützen vor eigenen Fehlern.

Nachteile bei der Verwendung von Java:

- Eine im Verhältnis zu C-Programmen schlechtere Performance. Erfahrungsmäßig sind Java-Programme wesentlich langsamer als C-Programme (Faktor 2-3 bei Floatingpoint Operationen.)
- Die Objektorientierung macht es Anfangs schwer eigene Programme zu schreiben, bzw. die Programme anderer zu verstehen.

1.2 Objektorientiert programmieren

In diesem Teil werden Begriffe aus der objektorientierten Programmierung erläutert. Diese Begriffe sind äußerst wichtig zum Verständnis der folgenden Teile, in denen es um Objektorientierung geht.

1.2.1 Objekte

In der realen Welt gibt es eine Unmenge von Objekten. Beispiele sind "dein Hund", "dein Fahrrad" etc. Alle diese Objekte besitzen Zustände und Verhalten. Zustände für dein Fahrrad sind zum Beispiel:

- aktueller Gang
- aktuelle Geschwindigkeit
- Anzahl der Gänge

Unter dem Verhalten eines Fahrrads kann man folgende Dinge verstehen:

- bremsen
- beschleunigen
- schalten

Bei der Softwareentwicklung werden die Zustände von Objekten durch Variablen, das Verhalten der Objekte mit Methoden beschrieben.

Definition Ein Objekt ist ein Stück Software, das Variablen und auf diese Variablen bezogene Methoden enthält.

Ein Software Objekt, das ein Fahrrad repräsentiert, hat dann die Variablen *aktuellerGang*, *aktuelleGeschwindigkeit* und *anzahlGänge*. Methoden, die sich auf das Fahrrad beziehen wären dann *bremsen*, *beschleunigen* und *schalten*.
Ein Objekt enthält nur die Variablen und Methoden, die unmittelbar mit dem Objekt zu tun haben. Es würde also keinen Sinn ergeben, wenn unser Fahrrad bellen könnte. Diese Methode ist mit Sicherheit in dem Objekt Hund besser aufgehoben.

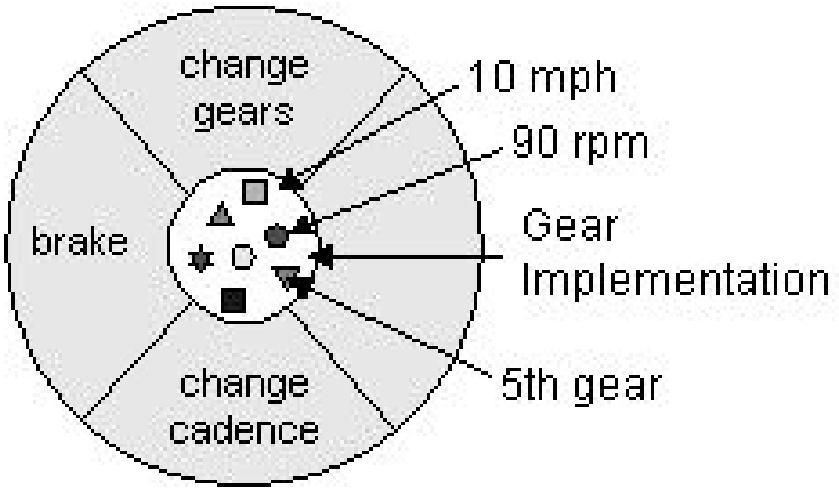

Abbildung 2: Das Objekt Fahrrad [brake3.jpg]

Die Variablen des Objektes bilden sozusagen dessen Kern. Die Methoden kann man als Hülle um diesen Kern auffassen, über welche die Variablen geändert werden können.
So bewirkt die Methode *beschleunigen* eine Veränderung der Variablen *aktuelleGeschwindigkeit*.
Die Vorteile dieser Technik liegen in der Modularität, und der Unabhängigkeit von der Implementierung (Stichwort *information hiding*).
Aus Performance-Gründen kann es aber auch sinnvoll sein, einen direkten Zugriff auf die Variablen zu erlauben. Dieser Punkt wird später bei der Beschreibung der Zugriffsrestriktionen näher beschrieben.

1.2.2 Verständigung von Objekten (Message)

Objekte der realen Welt können sich verständigen, bzw. interagieren. Bei Software-Objekten wird diese Möglichkeit mit Hilfe von Botschaften (Messages) beschrieben.
Botschaften werden durch Methodenaufrufe übermittelt. Welche Botschaften an ein Objekt gesendet werden können, wird direkt über die Methoden festgelegt, die dieses Objekt bereitstellt. Unser Fahrrad könnte also zum Beispiel die Botschaft *beschleunigen* oder *schalten* empfangen.

Angenommen, Du fährst mit Deinem Fahrrad den Berg hinauf, und möchtest schalten. Dieser Vorgang sähe, wenn man Dich und Dein Fahrrad als Objekte betrachten würde, so aus:

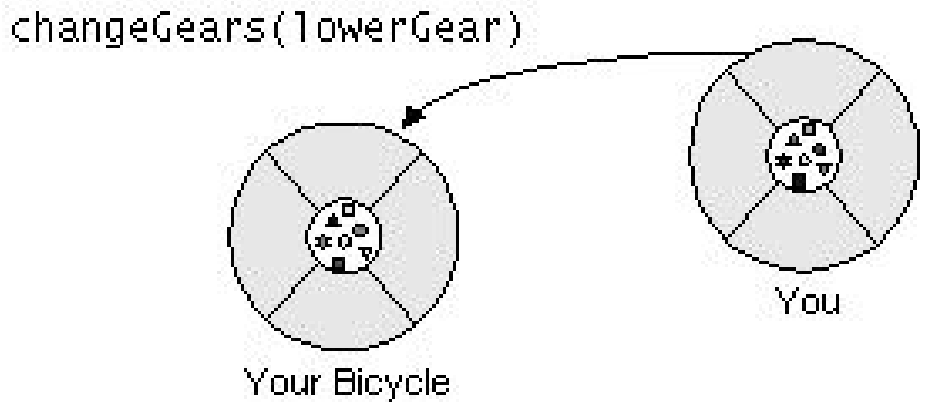

Abbildung 3: Botschaften [messageb5.jpg]

Eine Botschaft besteht aus 3 Teilen:

1. Dem Addressat der Botschaft (Dein Fahrrad).
2. Dem Namen der auszuführenden Methode (schalten)
3. Die zum Ausführen der Methode notwendigen Parameter (den Gang, in den geschaltet werden soll).

1.2.3 Klassen

Meist gibt es in der realen Welt mehr als ein Objekt gleichen Typs (es gibt außer Deinem Fahrrad noch viele andere). Bei Software-Objekten spricht man von Objekten der gleichen Klasse.
Eine Klasse ist sozusagen eine Bauanleitung (Blaupause) für Objekte. Sie beschreibt die möglichen Zustände von konkreten Objekten und auch deren Verhaltensweisen. Da z.B jedes Fahrrad bremsen, beschleunigen und schalten kann, ist es ausreichend, diese Merkmale ein einziges Mal für alle Fahrräder zu beschreiben.

Definition Eine Klasse ist die Bauanleitung für das Erzeugen bestimmter Objekte. Sie beschreibt die Variablen und Methoden der Objekte.

Es ist also notwendig um Objekte erzeugen zu können, erst einmal eine Bauanleitung für diese Objekte zu schaffen. Mit Hilfe dieser Bauanleitung können dann beliebig viele Objekte dieser Klasse hergestellt werden, die alle die selben Attribute (jedes Fahrrad hat eine Farbe) besitzen, aber unterschiedliche Eigenschaften (blaues, grünes oder rotes Fahrrad).

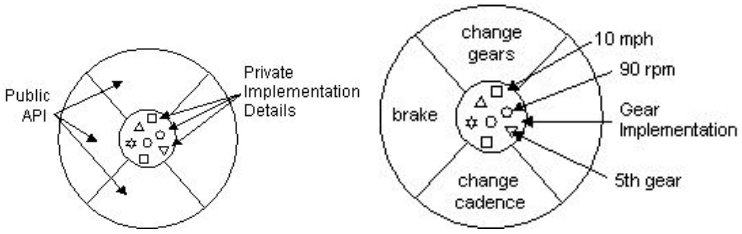

Abbildung 4: Klasse vs Objekt [class6.jpg, brake7.jpg

1.2.4 Vererbung

Die Vererbung ist ein mächtiger Mechanismus der Objektorientierung. Er ermöglicht es, bereits vorhandene Klassen zu erweitern, oder aber bestimmte Eigenschaften zu ändern. Als Beispiel sollen hier die Klassen Hund, Säugetier, Lebewesen gelten. Ein Lebewesen hat folgende Zustände:

- *Alter*
- *Größe*
- *Anzahl der Nachkommen*

Jedes Säugetier ist ein Lebewesen, es besitzt also alle Merkmale, die ein Lebewesen auch besitzt. Es wäre also wünschenswert, wenn man, anstatt die Zustände Alter, Größe und Anzahl der Nachkommen sowie die dazugehörigen Methoden für jedes speziellere Lebewesen erneut zu benennen, einfach sagen könnte: "Jedes Säugetier ist ein Lebewesen".
In diesem Fall würde es genügen, die Eigenschaften die ein Säugetier von anderen Lebewesen unterscheiden, zu spezifizieren. Zum Beispiel die *Trage-* bzw. *Säugezeit*. Alle anderen Eigenschaften werden in der Klasse Lebewesen spezifiziert.
Die Reihe läßt sich nun natürlich weiter fortsetzen: ein Hund ist ein spezielles Säugetier, ein Dalmatiner ein spezieller Hund etc. Auf Grund dieser **"ist ein"** Beziehungen entsteht eine Baumstruktur, die Vererbungsbaum bzw. Vererbungshierarchie genannt wird.

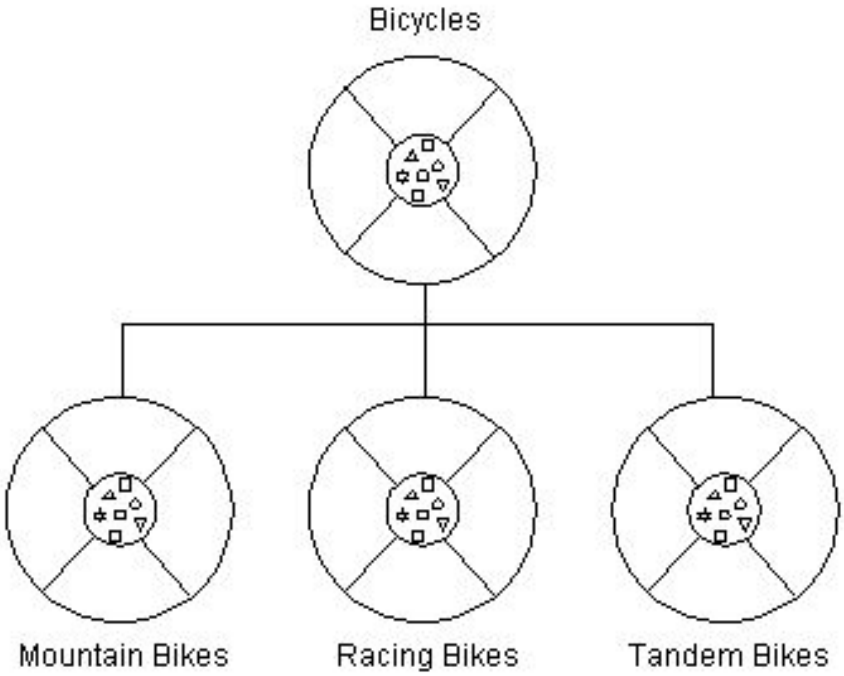

Abbildung 5: Vererbungsbaum [bike8.jpg]

1.2.5 Klassen-Attribute vs. Objekt-Attribute

Alle Attribute und Methoden, die wir bisher betrachtet haben, sind konkreten Objekten zugeordnet gewesen. Verbinden wir ein Attribut mit einem konkreten Objekt, so spricht man von einem Objekt-Attribut. Wird ein Objekt-Attribut geändert, so gilt die Änderung nur für dieses eine Objekt.
Erhöhe ich z.B. die aktuelle Geschwindigkeit meines Fahrrads, so wird auch nur mein Fahrrad schneller. Gleiches gilt auch für Objekt-Methoden, die nur Auswirkung auf das zugehörige Objekt haben.

Es gibt andererseits auch die Möglichkeit sogenannte Klassen-Attribute, bzw. Klassen-Methoden zu definieren. Für diese gilt dann, daß eine Änderung des Klassen-Attributs für alle Objekte der Klasse wirksam ist. Analog gilt gleiches auch für Klassen-Methoden.
Wäre z.B. die aktuelle Geschwindigkeit von Fahrrädern ein Klassen-Attribut, so würden, wenn ich mein Fahrrad beschleunige, alle Fahrräder beschleunigt.
Zugegeben, für Fahrräder wäre ein solches Verhalten unsinnig. Aber man könnte zum Beispiel die verschiedenen Fahrradtypen, wie z.B Damenfahrrad, Herrenfahrrad als Klassen-Attribute definieren:

```
// Klassen-Attribute
static int DAMEN_FAHRRAD = 1;
static int HERREN_FAHRRAD = 2;
```

.

Damit diese Klassen-Attribute auch in der Klasse einen Sinn ergeben definieren wir in der Klasse Fahrrad noch das Objekt-Attribut

```
// Objekt-Attribut, (Attribut)
int fahrradTyp = DAMEN_FAHRRAD;
```

.

Jedes Fahrrad kann also einen speziellen Typ (Attribut fahrradTyp) besitzen. Sinnvolle Typen werden durch die Klassen-Attribute *DAMEN_FAHRRAD* und *HERREN_FAHRRAD* definiert. Dieses Vorgehen ist praktisch, da

```
fahrradTyp == DAMEN_FAHRRAD
```

im Programm wesentlich besser lesbar ist als

```
fahrradTyp == 1
```

In einem späteren Teil werden wir auch besprechen, wie man *DAMEN_FAHRRAD* als Konstante deklariert.
Klassen-Attribute, bzw. Methoden werden mit dem Schlüsselwort static gekennzeichnet. Wird von einem anderen Objekt aus auf ein Klassen-Attribut (bzw. Methode) zugegriffen, so wird als Addressat der Klassenname angegeben.

1.3 Java programmiert

Einmal "Hello World" unter der Lupe:

```
/**
 * The HelloWorldApp class implements an application that
 * simply displays "'Hello World!"' to the standard output.
 */
class HelloWorldApp {
  public static void main(String[] args) {
    System.out.println("'Hello World!"'); //Display the string.
  }
}
```

1.3.1 Compiler

Wenn das JDK (Java Development Kit) auf Eurem Rechner installiert ist solltet Ihr bei der Eingabe von:
`>javac`
eine Meldung bekommen. Um eine Java-Quelldatei zu übersetzten wechselt einfach in das Verzeichnis der Datei und ruft:
`>javac DateiName.java`
auf. Wenn alles fehlerfrei übersetzt wurde könnt ihr das Programm starten.

1.3.2 Starten des Programms

Der vom Compiler erzeugte Java-Bytecode muß zum Ausführen interpretiert werden. Startet dazu die JVM (Java Virtual Machine) mit folgenden Befehl:
`>java DateiName`

1.3.3 Kommentare

Über der Klassendefinition befindet sich meist ein Kommetar, der Zweck und Inhalt der Klasse beschreibt. Es gibt drei verschieden Arten von Kommentaren:

// ... Zeilenkommentar
Der Text bis Zeilenende ist ein Kommentar.

/* ... */ Blockkommentar
Der Text "..." ist das Kommentar. Das Kommentar kann sich über mehrere Zeilen erstrecken.

/ ... */** JavaDoc Kommentar
Wie Blockkommentar, doch der Inhalt kann von einem speziellen Dokumentations-Tool (javadoc) analysiert werden.

1.3.4 Klassendefinition

Die Klassendefinition besteht aus den Teilen:

```
class HelloWorldApp { ... }
```

class Schlüsselwort, spezifiziert den Beginn einer Klassendefinition.

HelloWorldApp Klassenname, der Name der Klasse ist frei wählbar, in Klassennamen sollten keine Umlaute / Sonderzeichen enthalten sein. Sie müssen mit einem Buchstaben beginnen, der laut Konvention groß geschrieben wird. Es ist wichtig den Dateinamen genau gleich zu schreiben wie den Klassennamen (hier also "HelloWorldApp.java").

{...} Der Rumpf der Klasse steht in geschweiften Klammern. Es dürfen keine Anweisungen (Statements) außerhalb einer Klassendefinition stehen. Die einzige Ausnahme ist die Initialisierung von Variablen bei der Deklaration.

1.3.5 Methoden

Die Methodendefinition besteht aus folgenden Teilen:

```
public static void main(String[] args) { ... }
```

public Zugriffsmodifikator
Wird in einem der folgenden Teile näher erläutert.

static optionales Schlüsselwort.
Gibt an, das es sich um eine Klassen-Methode handelt.

void Rückgabewert
: void gibt an, daß nichts zurückgegeben wird.

main Methodenname
: Die Methode, die ein Java-Programm ausführbar macht heißt *main*. Sie wird beim Starten des Programms von der JVM aufgerufen.

(...) Parameterliste
: Hier wird ein Feld (Array) von Strings übergeben. Das Feld wird innerhalb der Methode unter dem Namen args angesprochen.

{...} Statements
: Die Statements, die beim Aufruf der Methode ausgeführt werden stehen in geschweiften Klammern.

1.3.6 Statements

Ein häufig gebrauchtes Statement ist das:

```
System.out.println("Hello World!);
```

System.out Addressat
: Der Addressat ist die Standardausgabe des Systems. Der Ausgabekanal out ist ein Klassenattribut der Klasse *System*.

println(...) Methodenname
: Eine Methode des Ausgabekanals, die eine Zeichenkette ausgibt.

"Hello World!" String
: Ein String ist eine Zeichenkette, eine Folge von Buchstaben (character).

; Abschluß
: Das ; schließt ein Statement ab.

1.4 Hausaufgabe

Zu Beginn einmal ganz einfach...

1. Wer noch kein HelloWorld programmiert hat tue das bis zum nächsten Mal!!!

2. Klassen:
   Überlege Dir drei Klassen aus dem Um feld Lebewesen mit Attributen und Methoden, beschreibe einige Objekte der Klassen.

3. Methoden:
   Was meldet der Compiler, wenn man bei der Methode *main(...)* (Siehe oben) den Rückgabewert void wegläßt? Beschreibe die Compiler-Meldung. Was folgerst Du aus dieser Meldung?

4. Klassen-Attribute:
   Überlege Dir eine weitere Anwendung für ein Klassen-Attribut.