

In diesem Kapitel

- Einleitung
- Modul 1 : Zusatzinformationen zu Java
 - Virtuelle Maschine
 - Byte Code verifier
 - Class Loader
 - JIT Compiler
 - API Dokumentation
- Modul 2 : Java Building Blocks
 - Konversionen von Datentypen
- Modul 3 : Ausdrücke und Kontrollflusssteuerung
 - Zeichenketten verbinden
 - Shift Operatoren >>,<<,>>>
 - Konversionen, Casting
- Modul 4 : Arrays
 - Wiederholung
 - Kopieren von Arrays
- Modul 5 : Objekte und Klassen
 - Abstrakte Datentypen
 - Überladen von Methoden
 - Überschreiben von Methoden
 - Finalizer
 - Packages
 - import
- Modul 6 : Exceptions
 - try...catch...finally
 - Behandeln von Ausnahmen
 - Definition eigener Ausnahmen
- Modul 7 : Applets - Eine Einführung
 - Applets verstehen
 - Ein Applet schreiben
 - Der Appletviewer
 - Zusätzliche Applet Möglichkeiten
- Zusammenfassung

Java Programmierung - Eine Einführung

1.1. Über diesen einführenden Teil des Kurses

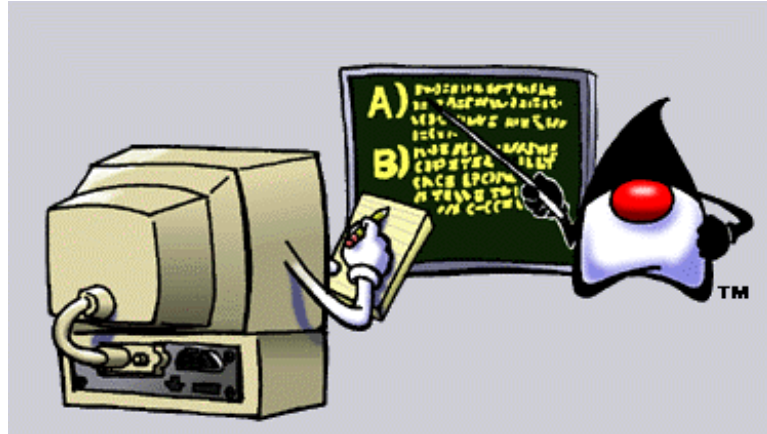
(Java Programmierung - eine Einführung) ist eine Einführung in grundlegende Konzepte der Java™ Programmiersprache.

Dieser Kurs wurde für Personen entwickelt, welche über limitiertes Wissen über Rechner und die Programmierung verfügen. Damit Sie diesen Kurs erfolgreich abschliessen können, sollten Sie den ersten Kurs, *Rechner- und Programmier- Grundlagen* besucht haben oder folgende Kenntnisse haben:

Teile dieses Kursteil wiederholen Konzepte aus der Kurseinheit *Programmieren mit Java 2*, andere vertiefen Aussagen, die wir dort bereits angetroffen haben:

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

insgesamt handelt es sich um eine Vertiefung, bei der einige neue Konzepte und Schlüsselworte eingeführt werden.

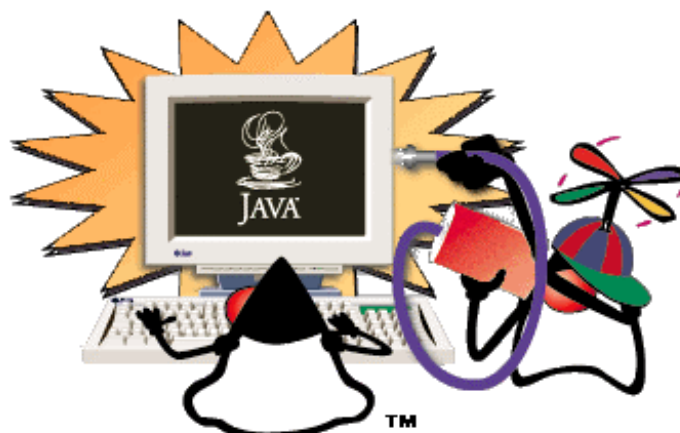


- grundlegende Prinzipien eines Rechners
- die Standardkomponenten eines Rechners
- elementare Programmierkenntnisse
- grundlegende Prinzipien der Software- und Produktentwicklung
- einfache Grundkenntnisse der objektorientierten Konzepte
- elementares Hintergrundwissen zu Java

1.2. *Kurs Einführung*

Herzlich willkommen zum Kurs *Java Programmierung - Eine Einführung*.

Java Technologie und Programme bestehen aus Klassen, aus denen Objekte gebaut werden. Das Verhalten dieser Objekte wird durch Methoden beschrieben und die individuellen Charakteristiken, welche Objekte voneinander unterscheiden, werden Attribute genannt.



1.2.1. Klassen

Klassen sind Templates, Platzhalter oder Vorlagen, aus denen Objekte gebildet werden. Klassen modellieren Ideen und Konzepte innerhalb des Problembereichs, der mit einem Programm gelöst werden soll. Vereinfacht gesagt wird es also so viele Klassen wie Konzepte innerhalb des Problembereichs geben. Klassen sollten nach Möglichkeit mit realen Konzepten korrespondieren. Beispiele wären also : eine Katze, eine Türe, usw. Klassen, die Sie definieren, hängen also von Ihrer Erfahrungswelt ab und dem Problem, welches Sie lösen möchten.

Eine Klasse ist ein Konstrukt, eine Programmierstruktur, mit deren Hilfe Daten und Funktionen (objektorientiert "Methoden" genannt) zusammengefasst werden und die in Form von Programmcode spezifiziert werden.

1.2.2. Objekte

Objekte sind Entitäten, welche gebildet werden, wenn Sie Ihr Programm starten. In Objekten werden die Daten gespeichert und manipuliert, welche benötigt werden, um Ihr Problem zu lösen. Die Daten werden mit Hilfe von Methoden, den Methoden des Objekts (welche in der Klasse definiert werden), manipuliert.

1.2.3. Methoden

Die Funktionen oder das Verhalten, welches ein Objekt zeigt, werden objektorientiert als Methoden bezeichnet. Methoden bestehen aus Anweisungen, welche bestimmte Aufgaben einer Klasse erfüllen.

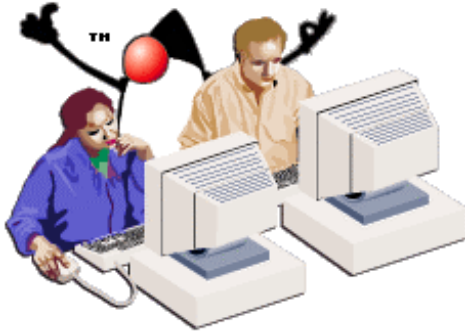
Beispielsweise könnte eine Methode eines Bankkontos dafür zu sorgen haben, dass der Kontostand abgefragt wird, also der Kontostand an einen externen Beobachter des Kontos übergeben wird. Die Aufgabe "abfragenKontostand()" ist eine Aufgabe, welche auf Stufe Bankkonto, also der Klasse, definiert wird.

1.2.4. Attribute

Attribute definieren / beschreiben ein Objekt. Attribute werden in Variablen innerhalb eines Objekts beschrieben. Jedesmal, wenn ein Objekt gebaut wird, werden (sollten) die Variablen initialisiert werden, also mit Werten belegt werden.

Beispielsweise in der Klasse Katze: bei der Bildung eines Objekts, welches zur Klasse Katze gehört (ein Katzenobjekt, eine konkrete Katze, eine Instanz der Klasse Katze), sollte der Name der Katze angegeben werden. Natürlich haben oder können unterschiedliche Katzen (Objekte) unterschiedliche Namen haben ("Stinky", "Peterli", "Käthi").

1.2.5. Kursziele



Nach dem Durcharbeiten diese Kurses sollten Sie in der Lage sein:

- die Regeln und Werkzeuge der Java Programmiersprache einzusetzen:
schreiben von Programmen
übersetzen von Programmen
starten einer kleinen Applikation
- Java Konstrukte einzusetzen, mit denen Sie Programme erweitern können.
- Programme zu schreiben, welche komplexere Kontrollstrukturen (bedingte Ausführung, Schleifen) benutzen
- Kapselungen einzusetzen, um Daten vor externen Einflüssen zu schützen.
- Arrays einzusetzen, um effizient mit Schleifen arbeiten zu können (Algorithmen [Schleife] und Datenstrukturen [Arrays] hängen voneinander ab).
- objektorientierte Werkzeuge einzusetzen, um Applikationsprogramme zu entwickeln.

1.3. Modul 1 : Informationen zu Java

1.3.1. Enleitung zu diesem Modul

Wir haben, indem wir uns auf das absolut notwendige beschränkten, im Schnellzugstempo eine Beschreibung der OO Techniken und Konzepte kennen gelernt, wie sie in Java implementiert wurden. Jetzt müssen wir das Wissen ergänzen, abrunden und vertiefen.

In diesem Modul geht es im Hintergrundwissen, also darum, was Java wirklich ist, um wichtige Eigenschaften von Java und der Java Umgebung.

1.3.1.1. Lernziele

Nach dem Durcharbeiten dieses Moduls sollten Sie in der Lage sein,

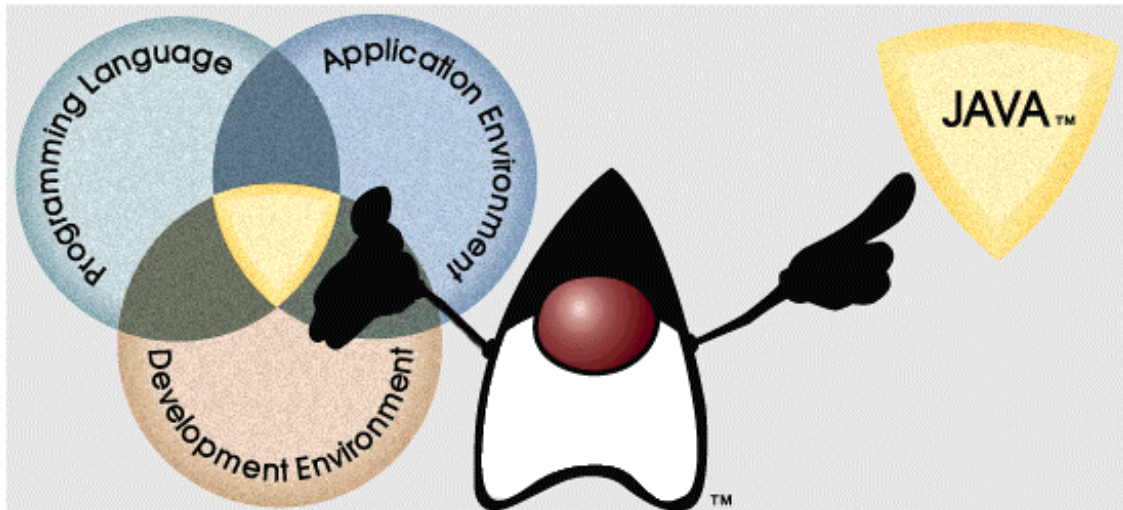
- die Schlüsselmerkmale der Programmiersprache Java aufzählen zu können.
- den grundsätzlichen Aufbau der Java Virtual Machine erklären können.
- erklären können, wie der Garbage Collector im Prinzip funktioniert.
- zu verstehen, warum Java sicherer als viele andere Programmiersprachen ist.
- mit der Java API Dokumentation umzugehen.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.3.2. Java ist ...

Java im Zusammenhang mit diesem Kurs ist:

- eine Programmiersprache
- eine Entwicklungsumgebung
- eine Applikationsumgebung



Java ist das Ergebnis der Suche nach einer Sprache, welche die Effizienz von C++ mit der Sicherheit von Smalltalk kombiniert.

C++ wurde in den 80er Jahren entwickelt, bei den Bell Labs in den USA. Die Sprache gilt als sehr effizient. Die Sprache kennt Klassen (und Objekte) mit Member Funktionen (Methoden) und Member Daten (Objektdaten) und viele weitere Konzepte, die auch in Java wieder zu finden sind.

Da C++ auf C basiert wurden viele Konzepte aus C übernommen, leider einige, die sich in der Praxis als gefährlich, wenn auch effizient, erwiesen haben.

Smalltalk wurde von Alan Kay am PARC (Palo Alto Research Center der Xerox) entwickelt. In Smalltalk ist alles ein Objekt, auch die Addition. Smalltalk Konzepte basieren zum Teil auf Simula, einer norwegischen Programmiersprache, die für Simulationen eingesetzt wurde. Da Objekte die Realität simulieren, ist es naheliegend, dass die Konzepte aus der Simulation dem OO Bereich ganz gut angepasst sind.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.3.3. Ziele von Java

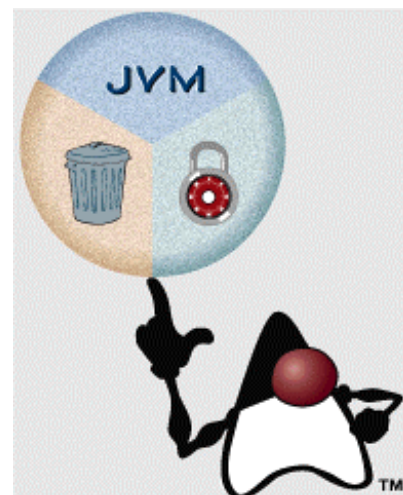
Die Hauptzielsetzungen von Java sind in einem White Paper von Sun Microsystems festgehalten. Sie finden auf dem Web Site zu diesem Kurs Links zu diesen Unterlagen.

Java sollte

- eine objektorientierte Sprache sein
- interpretierbar sein, also nicht übersetzt, sondern interpretiert werden, aus zwei Gründen:
 - a) schnellere Entwicklungszeiten als bei übersetzten Sprachen
 - b) Portabilität - dank einer Laufzeitumgebung, in Java 'Virtual Machine' genannt
- Programmier Techniken eliminieren, welche zu instabilem Programmcode führt
 - a) Pointer Arithmetik (direkte Manipulation des Speichers)
 - b) Speicher Belegung und Freigabe
- Threads unterstützen. Threads sind kleinere Kontrollflüsse, welche innerhalb eines andern Prozesses selbständig ablaufen können. Dadurch ist man in der Lage, mehrere Aktivitäten (Threads) gleichzeitig ausführen zu lassen.
- die Möglichkeit bieten, dynamisch (also während der Ausführung) neuen Programmcode zu laden (aus dem Internet oder lokal) und auszuführen.
- die Möglichkeit bieten, geladenen Programmteile gezielt zu kontrollieren, um die Sicherheit garantieren zu können.

Die drei Grundsäulen auf denen Java unter diesen Voraussetzungen entwickelt wurde, sind:

- die Java Virtual Machine
eine portierbare Laufzeitumgebung, welche viele weitere Aufgaben übernehmen kann, wie wir noch sehen werden.
- Garbage Collection
die automatische Verwaltung des Speichers. Nicht mehr benötigte Programmteile, Variablen und Objekte werden automatisch aus dem Speicher entfernt.
- Code Sicherheit
bevor Programme auf der Virtuellen Maschine ausgeführt werden, kann der Code analysiert und verifiziert werden. Dadurch kann die Sicherheit der Applikationen erhöht werden.

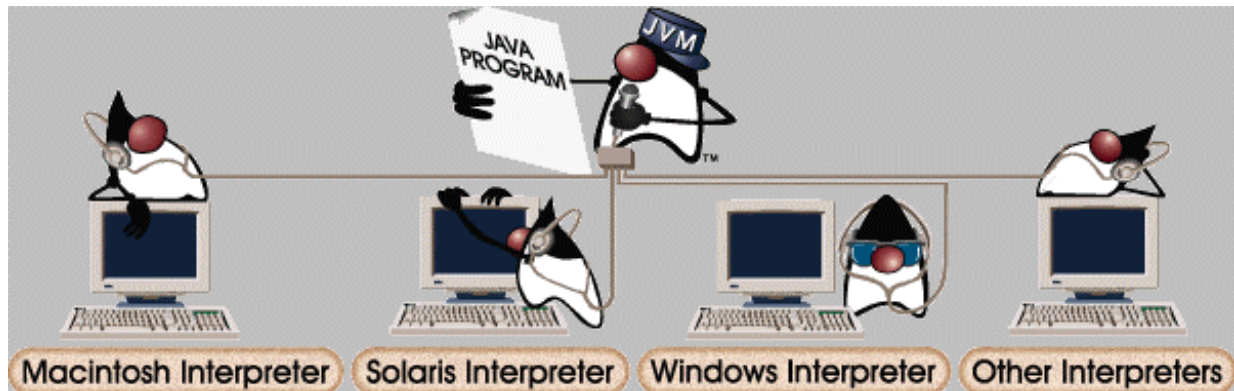


Jedes dieser Konzepte werden wir im Folgenden kennen lernen.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.3.3.1. Die Java Virtual Machine

Die Java Virtual Machine wurde in einem Dokument spezifiziert, welches allgemein zugänglich ist. Sie finden einen Link auf das Dokument auf dem Kurs Web.



In der *The Java Virtual Machine Specification* wird die JVM spezifiziert als:
"... an imaginary machine, that is implemented by emulating it in software on a real machine. Code for the Java Virtual Machine is stored in `.class` files, each of which contains code for, at most, one public class"

The Java Virtual Machine spezifiziert die Plattform, für die der gesamte Java Programmcode kompiliert wird. Die Spezifikation ermöglicht es, plattformunabhängigen Code zu schreiben, weil die Übersetzung für diese virtuelle, keine reale Maschine gemacht wird. Alles was man zu tun hat, ist die JVM auf unterschiedliche Hardware Plattformen zu portieren, damit der Bytecode (in der Class-Datei) interpretiert werden kann.

Die *JVM Spezifikation* definiert insbesondere

- Instruktionssatz
also die Befehle, die dem Programmierer zur Steuerung eines Prozessors, einer Maschine zur Verfügung stehen.
- Registersatz
einen Satz mit Standardregistern
- das Format der `class` Dateien
- einen Stack
also ein Speicherbereich, auf den auf eine First In Last Out Art und Weise Daten gespeichert werden können. Die meisten Prozessoren unterstützen Stacks.
- Garbage Collector verwalteten Heap
also ein Speicherbereich, auf dem dynamisch Daten gespeichert werden können. Die meisten Prozessoren unterstützen Heaps.
- Speicherbereiche

Die JVM versucht den Bytecode möglichst kompakt und effizient zu speichern und zu verwalten. Da in Java die Datentypprüfung ein fixer Bestandteil ist, der Code aber möglichst

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

portabel sein soll, wird die Typenprüfung weitestgehend bereits beim Übersetzen durchgeführt.

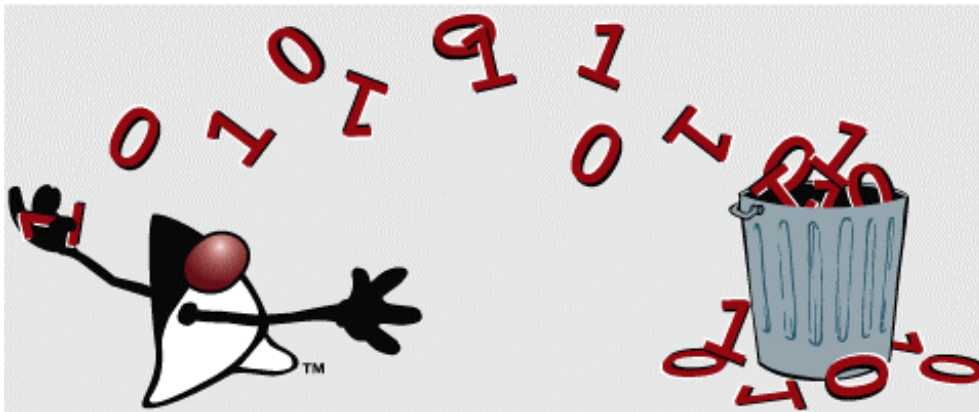
Jeder Java Interpreter muss in der Lage sein, das in der Spezifikation angegebene Bytecodeformat korrekt zu interpretieren.

1.3.3.2. Garbage Collection

Viele Programmiersprachen gestatten die dynamische Speicherverwaltung zur Laufzeit. Wie dies genau geschieht, hängt von der Programmiersprache ab. Aber das Grundmuster ist immer das Gleiche: es wird eine Adresse in Form eines sogenannten Pointers an das Programm zurück gegeben. Diese Adresse bestimmt den Startpunkt des reservierten Speicherbereichs.

Sobald der Speicherbereich nicht mehr benötigt wird, ist es sinnvoll, ihn frei zu geben und für andere Aufgaben einzusetzen. Sonst könnte es passieren, dass mit der Zeit kein Speicher mehr zur Verfügung steht, aber auch kein Speicher mehr aktiv im Einsatz ist.

In C und C++ muss der Programmierer selbst dafür sorgen, dass der Speicherplatz frei gegeben wird. Es ist aber oft nicht so einfach zu entscheiden, ob ein bestimmter Bereich noch benötigt wird. Daher wird es immer wieder passieren, dass Speicherplatz "verloren" geht und sogenannte *Memory Leaks* entstehen.



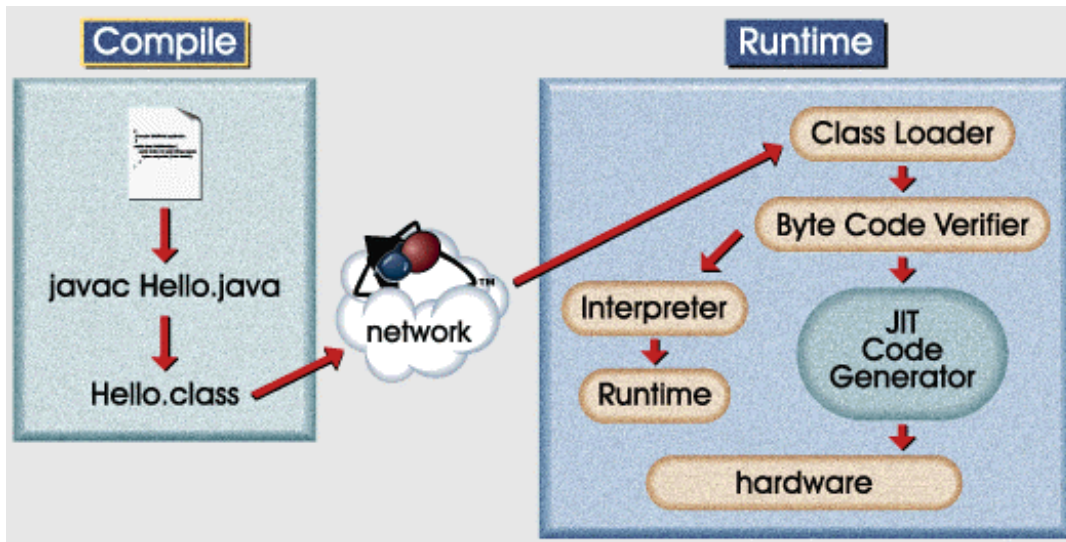
Java verlagert die Verantwortung für diese Aufgabe an einen speziellen Thread, der in der Virtuellen Maschine dauernd im Einsatz ist und dessen Aufgabe einfach ist, aufzuräumen, also nicht mehr benötigten Speicherplatz freizugeben. Dies geschieht mit Hilfe sogenannter Referenzzähler: es wird überprüft, ob auf diesen Bereich zugegriffen wird. Falls mehrere Programmteile dies tun, wird pro Programmteil der Zugriffszähler um eins erhöht. Falls ein Programmteil den Zugriff nicht mehr benötigt, wird er um eins erniedrigt. Falls er auf 0 ist, also nichts mehr auf den Bereich verweist, dann kann der Garbage Collector aktiv werden und diesen Bereich wieder frei geben.

Der Garbage Collector muss nicht extra gestartet werden, obschon auch dies möglich ist (mitten in Ihrem Programm).

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.3.3.3. Sicherheit der Java Programmierumgebung

In der folgenden schematischen Darstellung erkennen Sie den Ablauf des Übersetzens und Ausführens eines Java Programms.



An Hand dieses Bildes wollen wir nun die Sicherheitsaspekte besprechen.

Der erste Schritt ist einfach: Java Quellcode wird in Bytecode übersetzt, also eine Class Datei generiert. Dieser Code ist maschinenunabhängig.

Wo der Bytecode ausgeführt wird, kann später entschieden werden. Dies kann irgendwo auf dem Netzwerk sein.

Zur Laufzeit (rechte Seite der obigen Skizze) wird der Bytecode vom Class Loader (einer speziellen Klasse, die Sie auch noch selber modifizieren können) geladen, anschliessend vom Byte Code Verifier überprüft und schliesslich für das Zielsystem optimiert (interpretiert oder mit Hilfe des JIT (Just in Time) Code Generators optimiert).

Um den gesamten Prozess besser verstehen zu können, betrachten wir die einzelnen Teile des Laufzeitsystems etwas genauer, speziell den Interpreter, den Class Loader und den JIT Compiler.

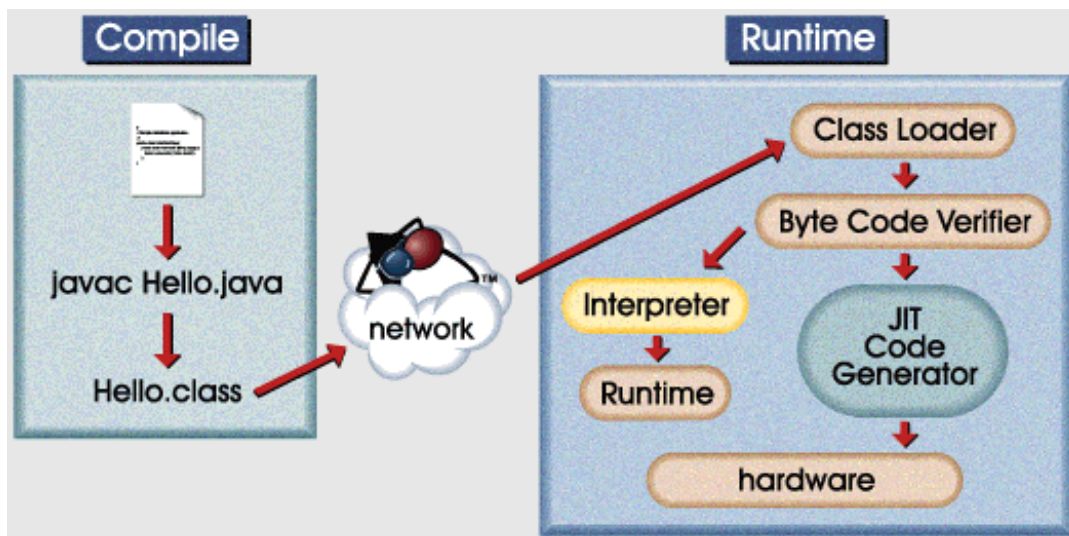
JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.3.3.3.1. Interpreter

Als erstes betrachten wir den **Interpreter**, als eine der zentralen Komponenten der JVM.

Die Hauptaufgaben des Interpreters sind:

- das Laden des Bytecode - dies geschieht mit Hilfe des Class Loaders
- das Verifizieren des Bytecodes - dies geschieht mit Hilfe des Byte Code Verifiers
- das Ausführen des Bytecodes - das ist die eigentliche Aufgabe des Interpreters, das Interpretieren.

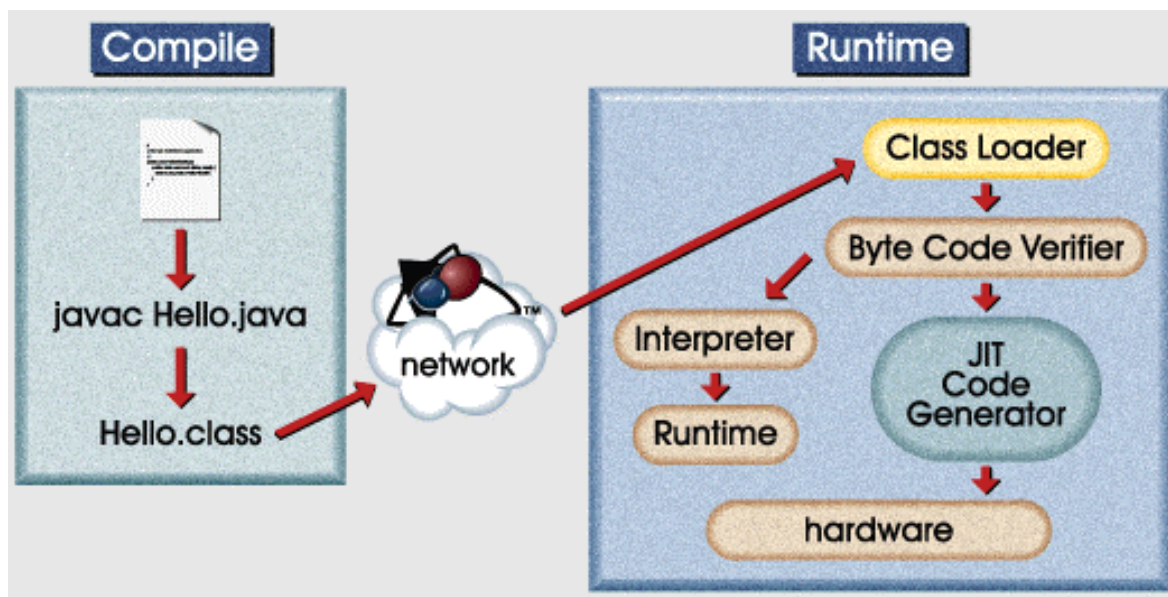


JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.3.3.3.2. Klassenlader

Der **Class Loader** hat folgende Aufgaben.

Hauptaufgabe des Class Loaders ist das Laden der Klassen, des Bytecodes, wie der Name schon besagt. Der Class Loader unterteilt dabei aber auch noch gleich den Bytecode in solchen, welcher aus einer lokalen Quelle, Datei stammt und solchen, welcher von entfernten Quellen, also aus dem Netzwerk, stammt. Diese Unterscheidung erleichtert die Überprüfung der Rechte: lokale Programme dürfen wesentlich mehr, als Programmcode, den wir in Form von Bytecode aus dem Internet herunterladen. Dadurch werden Trojanische Pferde vermieden.

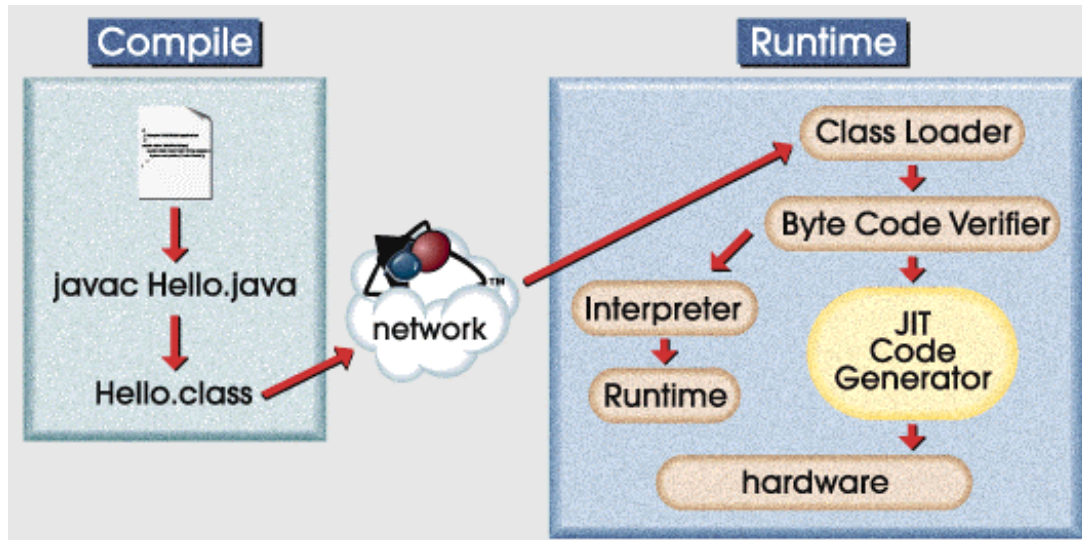


Sobald alle Klassen geladen sind, kann mit der Speicherverwaltung begonnen werden. Es kann überprüft werden, ob unerlaubte Speicherzugriffe vorgesehen sind. Dadurch wird die Sicherheit des Programms und damit des gesamten Laufzeitsystems deutlich erhöht.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.3.3.3. Just In Time Code Generator

Alternativ zum Interpreter, kann auch der **Just In Time (JIT)** Code Generator ausgeführt werden.



JIT übersetzt den Bytecode in Maschinsprache, bevor der Code ausgeführt wird. Wenn Sie beispielsweise ein Applet herunterladen, kann diese Code Generierung einen kurzen Augenblick dauern. Aber die anschließende Ausführung wird dadurch deutlich beschleunigt.

1.3.3.4. Code Verifikationsprozess

Der Java Code durchläuft mehrere Prüfschritte, bei denen der Programmcode überprüft wird, bevor er schliesslich als ausführbar (interpretiert oder mit dem JIT Code Generator übersetzt) akzeptiert wird.

Insbesondere werden überprüft:

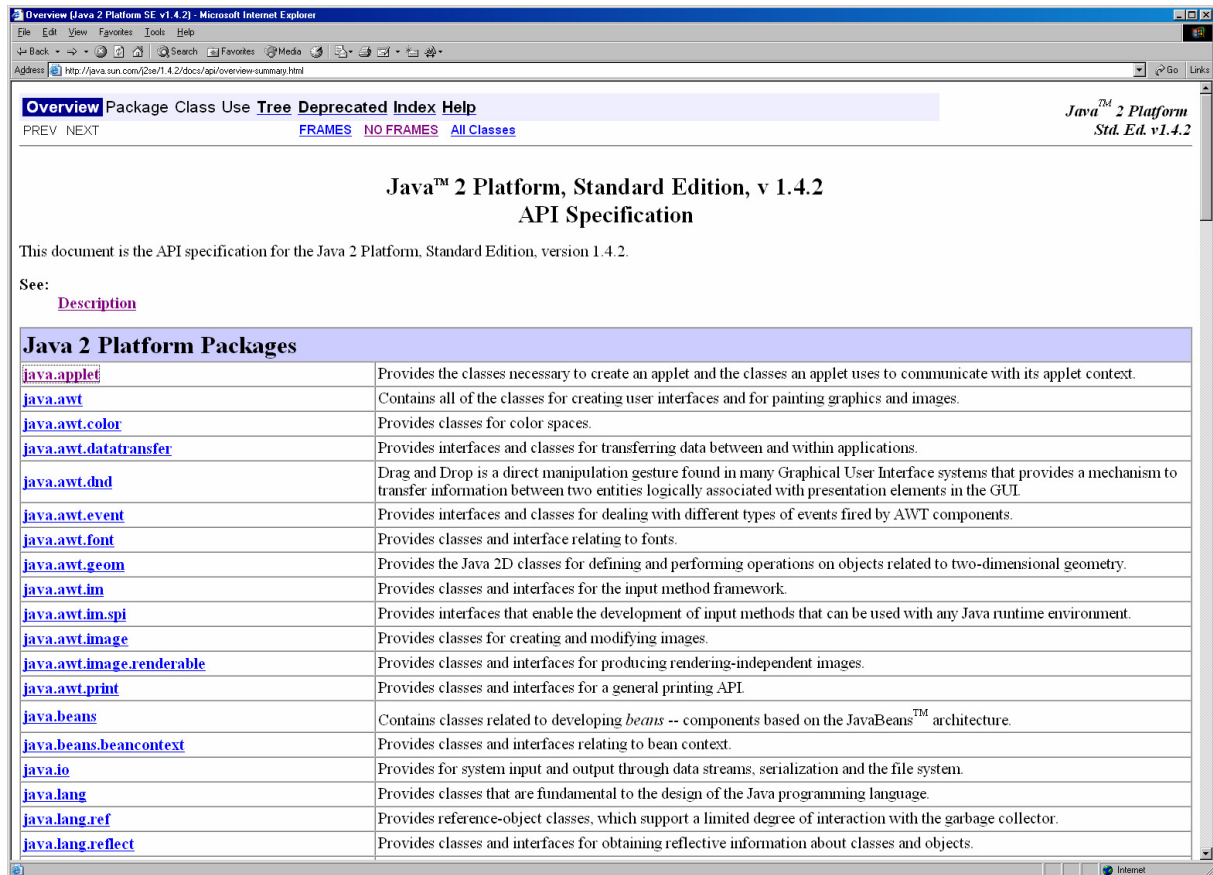
- versuchte, illegale Speicherzugriffe (Verletzungen)
- unberechtigter Zugriff auf einzelne Objekte
- der Versuch Objekte oder Klassen illegal zu verändern.

Der gesamte Bytecode, der über das Netzwerk an die Virtuelle Maschine gesendet wird, wird insgesamt vierfach überprüft, ob der Bytecode der JVM Spezifikation entspricht und die System Integrität nicht verletzt.

Falls der Verifikationsprozess erfolgreich verlief, also keine Fehlermeldung generiert wurde, können Sie davon ausgehen, dass

- die Klassen der Class Dateispezifikation der JVM entsprechen
- es keine Zugriffsrechtsverletzungen gibt
- es keine Stack Overflows oder Underflows gibt
- Parameteranzahl und Parametertypen korrekt spezifiziert sind
- keine illegalen Datentypkonversionen geschehen oder fälschlicherweise geplant sind
- Zugriffe auf die Objektefelder (Methoden und Daten) korrekt sind.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG



1.3.4. Die Java API Dokumentation

API steht für Application Programming Interface, also einfach gesagt, die Schnittstellenspezifikation oder noch einfacher: die Spezifikation von dem, was Java bereits bietet, in Form von Klassen, Packages (mehrere Klassen, die zusammengehören, um ein Problem zu lösen, oder eine bestimmte Funktionalität anzubieten).

Das JDK (Java Development Kit) kommt mit der gesamten Dokumentation der Standardklassen. Diese Dokumentation ist in HTML Form, also leicht auf (fast) allen Rechnern einsetzbar.

Das Gute an der Dokumentation ist, das sie immer genau gleich aufgebaut ist und bei sauberer Programmdokumentation automatisch erzeugt werden kann.

Die Dokumentation ist hierarchisch aufgebaut. Die Klassenhierarchie können Sie einblenden. Sie steht auch immer am Anfang der Klassenbeschreibung.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

Die Klassenbeschreibung umfasst folgende Bereiche, die auch fehlen können:

- die Klassenhierarchie
- eine Beschreibung der Klasse und deren Einsatz
- eine Liste der Member Variablen
- eine Liste der Konstruktoren
- eine Liste der Methoden

dann folgt die Detaillierung:

- eine detailliertere Beschreibung der Member Variablen
- eine detailliertere Beschreibung der Konstruktoren
- eine detailliertere Beschreibung der Methoden

Die einzelnen Bereiche sind mit Hyperlinks verbunden.

In vielen Entwicklungsumgebungen, beispielsweise JBuilder, erhalten Sie kontextsensitive Hilfe: falls Sie auf eine Variable oder ein Konzept zeigen, mit der Maus, und dann F1 drücken, wird die entsprechende Beschreibung aus der JavaDoc eingeblendet.

Die Java Dokumentation enthält auch Hinweise für den praktischen Einsatz verschiedener Konzepte. Diese sind in den User Guides zusammengefasst.

Wertvoll ist auch die Liste der deprecated, also im Laufe der Zeit verworfenen Konzepte. In der Regel finden Sie dort auch Hinweise, wie man den Einsatz dieser Konzepte, die man nicht mehr einsetzen sollte, umgehen kann, also wie man zu einer besseren Lösung gelangen kann.

Selbsttestaufgabe 1

Schauen Sie nach, was in der Java Dokumentation unter `System` steht. Sie verwenden ja in Ihren Programmen häufig die `System` Ausgabe.¹

Erläutern Sie, was in der Anweisung `System.out.println()` die einzelnen der drei Komponenten bedeuten, was sie konkret sind.

¹ System gehört ins Package `java.lang`

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

Beispiel

Zuerst die Beschreibung der Klasse

Class java.lang.String

[java.lang.Object](#)

```
|
+----java.lang.String
```

public final class **String**

extends [Object](#)

implements [Serializable](#) The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
String str = "abc";
```

is equivalent to:

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

Here are some more examples of how strings can be used:

```
System.out.println("abc");
String cde = "cde";
System.out.println("abc" + cde);
String c = "abc".substring(2,3);
String d = cde.substring(1, 2);
```

The class String includes methods for examining individual characters of the sequence, for comparing strings, for searching strings, for extracting substrings, and for creating a copy of a string with all characters translated to uppercase or to lowercase.

The Java language provides special support for the string concatenation operator (+), and for conversion of other objects to strings. String concatenation is implemented through the StringBuffer class and its append method. String conversions are implemented through the method toString, defined by Object and inherited by all classes in Java. For additional information on string concatenation and conversion, see Gosling, Joy, and Steele, *The Java Language Specification*.

See Also:

[toString](#), [StringBuffer](#), [append](#), [append](#), [append](#), [append](#), [append](#), [append](#), [append](#), [append](#), [append](#), [append](#)

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

Dann folgt eine Liste aller Konstruktoren

Constructor Summary

[String](#)()

Initializes a newly created `String` object so that it represents an empty character sequence.

[String](#)(byte[] bytes)

Constructs a new `String` by decoding the specified array of bytes using the platform's default charset.

[String](#)(byte[] ascii, int hiByte)

Deprecated. *This method does not properly convert bytes into characters. As of JDK 1.1, the preferred way to do this is via the `String` constructors that take a charset name or that use the platform's default charset.*

[String](#)(byte[] bytes, int offset, int length)

Constructs a new `String` by decoding the specified subarray of bytes using the platform's default charset.

[String](#)(byte[] ascii, int hiByte, int offset, int count)

Deprecated. *This method does not properly convert bytes into characters. As of JDK 1.1, the preferred way to do this is via the `String` constructors that take a charset name or that use the platform's default charset.*

[String](#)(byte[] bytes, int offset, int length, [String](#) charsetName)

Constructs a new `String` by decoding the specified subarray of bytes using the specified charset.

[String](#)(byte[] bytes, [String](#) charsetName)

Constructs a new `String` by decoding the specified array of bytes using the specified charset.

[String](#)(char[] value)

Allocates a new `String` so that it represents the sequence of characters currently contained in the character array argument.

[String](#)(char[] value, int offset, int count)

Allocates a new `String` that contains characters from a subarray of the character array argument.

[String](#)([String](#) original)

Initializes a newly created `String` object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.

[String](#)([StringBuffer](#) buffer)

Allocates a new string that contains the sequence of characters currently contained in the string buffer argument.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

Und Methoden:

Method Summary

char	charAt (int index) Returns the character at the specified index.
int	compareTo (Object o) Compares this String to another Object.
int	compareTo (String anotherString) Compares two strings lexicographically.
int	compareToIgnoreCase (String str) Compares two strings lexicographically, ignoring case differences.
String	concat (String str) Concatenates the specified string to the end of this string.
boolean	contentEquals (StringBuffer sb) Returns true if and only if this String represents the same sequence of characters as the specified StringBuffer.
static String	copyValueOf (char[] data) Returns a String that represents the character sequence in the array specified.
static String	copyValueOf (char[] data, int offset, int count) Returns a String that represents the character sequence in the array specified.
boolean	endsWith (String suffix) Tests if this string ends with the specified suffix.
boolean	equals (Object anObject) Compares this string to the specified object.
boolean	equalsIgnoreCase (String anotherString) Compares this String to another String, ignoring case considerations.
byte[]	getBytes () Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.
void	getBytes (int srcBegin, int srcEnd, byte[] dst, int dstBegin) Deprecated. <i>This method does not properly convert characters into bytes. As of JDK 1.1, the preferred way to do this is via the <code>getBytes()</code> method, which uses the platform's default charset.</i>
byte[]	getBytes (String charsetName) Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte array.
void	getChars (int srcBegin, int srcEnd, char[] dst, int dstBegin) Copies characters from this string into the destination character array.
int	hashCode () Returns a hash code for this string.
int	indexOf (int ch) Returns the index within this string of the first occurrence of the

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

	specified character.
int	<u>indexOf</u> (int ch, int fromIndex) Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
int	<u>indexOf</u> (<u>String</u> str) Returns the index within this string of the first occurrence of the specified substring.
int	<u>indexOf</u> (<u>String</u> str, int fromIndex) Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
<u>String</u>	<u>intern</u> () Returns a canonical representation for the string object.
int	<u>lastIndexOf</u> (int ch) Returns the index within this string of the last occurrence of the specified character.
int	<u>lastIndexOf</u> (int ch, int fromIndex) Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
int	<u>lastIndexOf</u> (<u>String</u> str) Returns the index within this string of the rightmost occurrence of the specified substring.
int	<u>lastIndexOf</u> (<u>String</u> str, int fromIndex) Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.
int	<u>length</u> () Returns the length of this string.
boolean	<u>matches</u> (<u>String</u> regex) Tells whether or not this string matches the given <u>regular expression</u> .
boolean	<u>regionMatches</u> (boolean ignoreCase, int toffset, <u>String</u> other, int ooffset, int len) Tests if two string regions are equal.
boolean	<u>regionMatches</u> (int toffset, <u>String</u> other, int ooffset, int len) Tests if two string regions are equal.
<u>String</u>	<u>replace</u> (char oldChar, char newChar) Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.
<u>String</u>	<u>replaceAll</u> (<u>String</u> regex, <u>String</u> replacement) Replaces each substring of this string that matches the given <u>regular expression</u> with the given replacement.
<u>String</u>	<u>replaceFirst</u> (<u>String</u> regex, <u>String</u> replacement) Replaces the first substring of this string that matches the given <u>regular expression</u> with the given replacement.
<u>String</u> []	<u>split</u> (<u>String</u> regex) Splits this string around matches of the given <u>regular expression</u> .
<u>String</u> []	<u>split</u> (<u>String</u> regex, int limit)

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

	Splits this string around matches of the given regular expression .
boolean	startsWith (String prefix) Tests if this string starts with the specified prefix.
boolean	startsWith (String prefix, int toffset) Tests if this string starts with the specified prefix beginning a specified index.
CharSequence	subSequence (int beginIndex, int endIndex) Returns a new character sequence that is a subsequence of this sequence.
String	substring (int beginIndex) Returns a new string that is a substring of this string.
String	substring (int beginIndex, int endIndex) Returns a new string that is a substring of this string.
char[]	toCharArray () Converts this string to a new character array.
String	toLowerCase () Converts all of the characters in this String to lower case using the rules of the default locale.
String	toLowerCase (Locale locale) Converts all of the characters in this String to lower case using the rules of the given Locale .
String	toString () This object (which is already a string!) is itself returned.
String	toUpperCase () Converts all of the characters in this String to upper case using the rules of the default locale.
String	toUpperCase (Locale locale) Converts all of the characters in this String to upper case using the rules of the given Locale .
String	trim () Returns a copy of the string, with leading and trailing whitespace omitted.
static String	valueOf (boolean b) Returns the string representation of the boolean argument.
static String	valueOf (char c) Returns the string representation of the char argument.
static String	valueOf (char[] data) Returns the string representation of the char array argument.
static String	valueOf (char[] data, int offset, int count) Returns the string representation of a specific subarray of the char array argument.
static String	valueOf (double d) Returns the string representation of the double argument.
static String	valueOf (float f) Returns the string representation of the float argument.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

static String	valueOf (int i) Returns the string representation of the <code>int</code> argument.
static String	valueOf (long l) Returns the string representation of the <code>long</code> argument.
static String	valueOf (Object obj) Returns the string representation of the <code>Object</code> argument.

Methods inherited from class [java.lang.Object](#)

[clone](#), [finalize](#), [getClass](#), [notify](#), [notifyAll](#), [wait](#), [wait](#), [wait](#)

.und deren detaillierten Beschreibungen:

Field Detail

1.3.5. `CASE_INSENSITIVE_ORDER`

public static final [Comparator](#) `CASE_INSENSITIVE_ORDER`

A `Comparator` that orders `String` objects as by `compareToIgnoreCase`. This comparator is serializable.

Note that this `Comparator` does *not* take locale into account, and will result in an unsatisfactory ordering for certain locales. The `java.text` package provides *Collators* to allow locale-sensitive ordering.

Since:

1.2

See Also:

[Collator.compare\(String, String\)](#)

Constructor Detail

1.3.6. `String`

public **String**()

Initializes a newly created `String` object so that it represents an empty character sequence. Note that use of this constructor is unnecessary since `Strings` are immutable.

..
...

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.3.7. Quiz - Java Grundlagen

Zur Abwechslung noch etwas zur Festigung Ihres Wissens. Sie sollten jetzt Grundkenntnisse über das Java Programmiersystem haben. Wie gut Ihr Wissen ist, können Sie gleich testen.



1. Was sind die Hauptaufgaben von `java` im JDK?²
 - a) Programmcode übersetzen
 - b) Programme laden
 - c) Code verifizieren
 - d) Code ausführen

2. Ihr Programm hat die Prüfung durch den Code Verifier bestanden. Was bedeutet dies?³
 - a) es gibt keine Zugriffsverletzungen
 - b) es gibt keine Stack Overflows und Underflows
 - c) Anzahl und Datentyp der Parameter sind korrekt
 - d) es gibt keine Laufzeitfehler

3. Die Java Dokumentation enthält folgende Bereiche⁴
 - a) eine Liste der Member Variablen
 - b) eine Liste der Konstruktoren
 - c) eine Liste der Methoden
 - d) eine Liste der lokalen Variablen
 - e) eine Liste mit detaillierteren Beschreibungen

4. Die *Java Virtual Machine Specification* beschreibt⁵
 - a) den Registersatz
 - b) den Instruktionssatz
 - c) Class Datei Formatspezifikation
 - d) Code Sicherheit
 - e) Speicherbereiche

² b)c)d) übersetzt wurde mit `javac`

³ a) b) c) ; die Laufzeitfehler stammen vermutlich von Ihnen

⁴ a,b,c,e; d ist Teil Ihres Programms

⁵ a, b, c, e : d wird im Programm selber realisiert unter Zuhilfenahme von Java Klassen

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.3.8. Zusammenfassung - Java Grundlagen

In diesem Modul haben Sie mehrere Konzepte kennen gelernt, welche Ihnen Hintergrundwissen über Java vermitteln.

Nach dem Durcharbeiten dieses Moduls sollten Sie in der Lage sein:

- einige der Schlüsseigenschaften von Java aufzählen zu können
- die Elemente der Java Virtual Machine zu beschreiben
- die grundsätzliche Funktionsweise des Garbage Collectors zu beschreiben
- mit der Java Dokumentation umzugehen.

1.4. Modul 2 : Java Building Blocks

Wir haben bereits einige der Building Blocks der Java Programmierumgebung kennen gelernt. Jetzt wollen wir dieses Wissen vertiefen.

1.4.1. Einleitung

Wir bauen auf bestehendem Know How auf, das heisst, den Basisdatentypen, einfachen Kontrollstrukturen und ähnlichen Konstrukten, den grundlegenden syntaktischen Elementen. Aber wir werden auch Ihr Wissen über Klassen und Objekte vertiefen. Dieser Modul ergänzt jenen Modul, in dem Sie lernten, wie man Kommentare schreibt, welche Basisdatentypen existieren, wie Literale verwendet und umgesetzt werden und vieles mehr.

1.4.1.1. Lernziele

Nach dem Durcharbeiten dieses Moduls sind Sie in der Lage:

- einfache Datentypen besser zu verstehen.
- Datentypen und die Konstruktion von Objekten besser zu verstehen.
- auf Member Variablen zuzugreifen, auch falls diese in Oberklassen definiert wurden.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.4.2. Primitive Datentypen besser verstehen

Gelegentlich sind Sie gezwungen hexadezimale oder oktale Darstellungen von Werten zu verwenden.

Java bietet dazu folgende Konstrukte:

Integrale Basisdatentypen, also Byte, Short, Integer, Long (*byte, short, int, long*), können durch Voranstellen einer *0 (Null) oktal* angegeben werden, durch Voranstellen von *0x (Null x) hexadezimal* angegeben werden:

Beispiel:

Integer Wert sei 2 (zwei)

Spezifikation oktal : 077

Spezifikation hexal: 0xFEFE

Sie können diese Darstellungen auch für Long einsetzen. In diesem Fall hätten Sie:

Long Wert sei 2: 2L

Spezifikation in oktal : 077L

Spezifikation in hexal: 0xFEFE L

1.4.2.1. Konversion von Basisdatentypen

Die Konversion der Basisdatentypen basiert auf folgender Regel:

***Erweiterungen sind erlaubt
Genauigkeitsverlust ist verboten.***

Wir werden uns mit Datenkonversionen, automatisch oder gewollt noch beschäftigen.

1.4.3. Objekte besser verstehen

Einen der Hauptfehler in der Programmierung entsteht, wenn Programmierer alle Datenelemente einzeln aufführen und damit auch verwalten müssen.

Beispiel:

Um die Daten zu erfassen, könnten wir drei Datenelemente definieren

```
int day, month, year
```

Nun haben wir die grösste Flexibilität: wir können jedes Datenelement, jedes Attribut einzeln verwalten und verändern. Aber....

Falls wir beispielsweise zwei Geburtstage beschreiben müssen, steht uns einiges bevor:

```
int myBirthDay, myBirthMonth, myBirthYear;  
int yourBirthDay, yourBirthMonth, yourBirthYear;
```

Die Methode scheint also etwas umständlich zu werden, beispielsweise wenn wir eine arabische Grossfamilie verwalten müssen.

Die Methode hat zwei Schwächen:

1. sie wird aufwendig, falls wir viele Daten verwalten müssen
2. jede dieser Variablen ist unabhängig, obschon alle zusammen ein Datum darstellen, eben zusammen, nicht einzeln.

1.4.3.1. Aggregierte Datentypen

Die meisten Programmiersprachen unterstützen das Konzept des Datentyps. Ein Wert kann beispielsweise ein Integer, Float oder Character sein.

Das reicht aber offensichtlich nicht aus. Wir sollten die Möglichkeit haben, neue Datentypen zu definieren, welche unserem Problem angepasst sind, beispielsweise "Date".

Diese Problem lösen viele Programmiersprachen mit sogenannten *aggregierten Datentypen*. In einzelnen Programmiersprachen bezeichnet man aggregierte Datentypen als *strukturierte Datentypen* oder *Recordtypen*.

Aggregierte Datentypen werden vom Programmierer im Quellcode definiert. Nachdem die Datentypen definiert sind, kann man sie wie Basisdatentypen verwenden. Die Beschreibung des Datentypen geschieht, indem die einzelnen Teile des aggregierten Datentyps beschrieben werden.

Betrachten wir eine einfache Deklaration:

```
int day;
```

Java erkennt, dass ein bestimmter Speicherplatz reserviert werden muss und ist in der Lage, den Inhalt des Speichernereichs zu interpretieren.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

Wenn wir also neue Datentypen definieren wollen, müssen wir

1. angeben, wieviel Speicherplatz benötigt wird
2. wie der Speicherinhalt interpretiert werden soll

In der Regel reserviert man Speicherplatz nicht als Anzahl Bytes oder Bits, sondern einfach durch Angabe der Datentypen, aus denen der neue Datentyp zusammengesetzt werden soll.

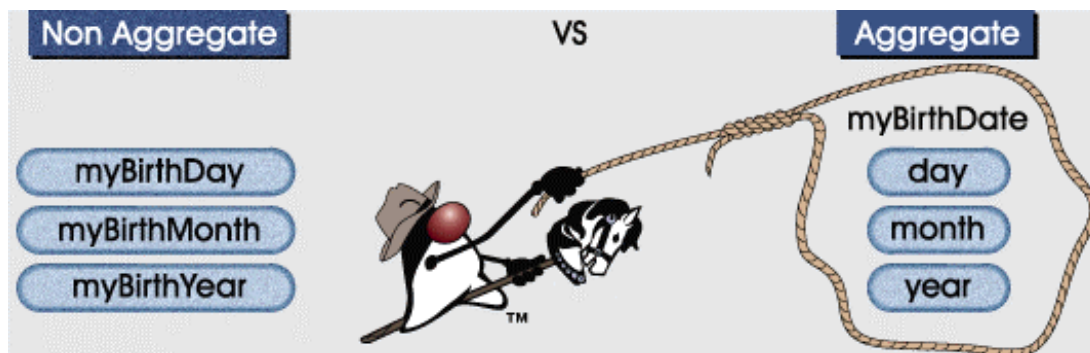
Beispielsweise, um unser Datum zu definieren:

wir benötigen genügend Speicherplatz, um drei Integer Variablen speichern zu können. Zudem müssen die Daten korrekt gemäss dem Aufbau eines Datums interpretiert werden.

In Java könnten Sie eine Klasse definieren (Date gibt es allerdings schon im Package java.util):

```
public class Date {  
    int day;  
    int month;  
    int year;  
}
```

Dabei sollten Sie sich bewusst sein, dass Klassen weit mehr sind als nur abstrakte Datentypen.



Jetzt könnten wir unsere Geburtstage neu definieren als:

```
Date myBirthDay, yourBirthDay;
```

Jetzt müssen wir noch auf die einzelnen Daten zugreifen und unsere Geburtstage initialisieren:

```
myBirthDay.day=11;  
myBirthDay.month=11;  
myBirthDay.year=2011;
```

Wir verwenden dazu einfach den dot (.) Operator.

Mit den aggregierten Datentypen ist es viel einfacher, die Daten zu verwalten. Die einzelnen Datenelemente werden Teil eines Ganzen, sind also nicht mehr isoliert. Damit wird auch der Namensraum kleiner, das heisst, dass weniger Namen verwaltet werden müssen.

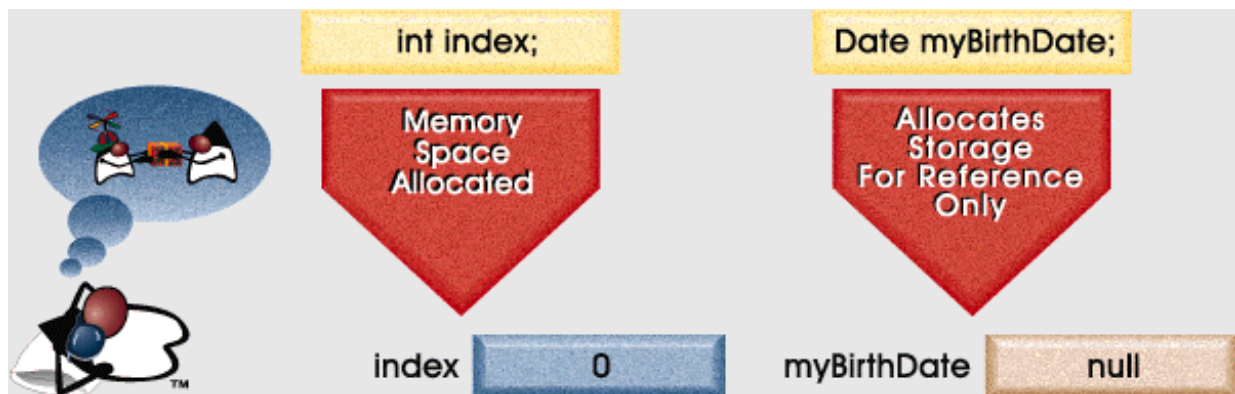
Die Elemente, aus denen die Klasse aufgebaut wird, werden als *Member Variablen* bezeichnet. Das Objekt in Java ist eine Instanz des Aggregierten Datentyps. Der Geburtstag ist eine Instanz der Klasse Date, einem Objekt mit drei Member Variablen.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.4.3.2. Kreieren eines Objekts - Deklaration

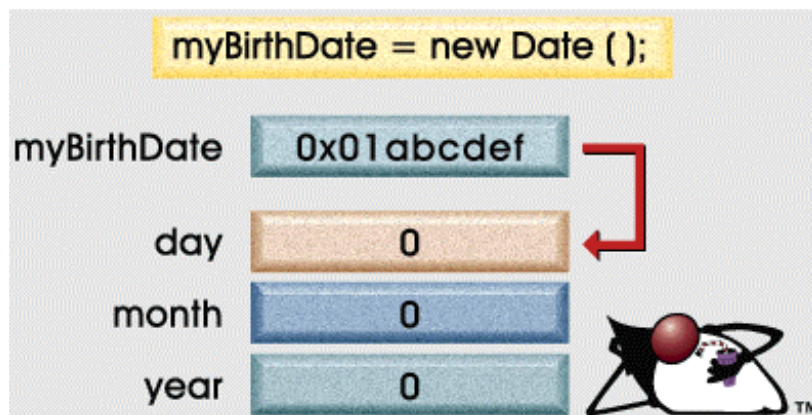
Wenn Sie eine Variable eines Basisdatentyps deklarieren, wie beispielsweise `boolean`, `byte`, `short`, `char`, `int`, `long`, `float`, `double` wird der Speicherplatz gleich als Teil der Deklaration angelegt.

Wenn Sie eine Variable eines Klassentyps, also ein Objekt, anlegen, dann wird mit der Deklaration der Speicherplatz nicht angelegt. In Wirklichkeit ist die Variable, die wir als Objekt deklarieren, eine *Referenz* auf die Daten, nicht die Daten selbst, sondern bloss ein Pointer auf den Speicherbereich, in dem das Objekt verwaltet wird.



1.4.3.3. Kreieren von Objekten - die `new` Anweisung

Bevor Sie die Variable, das Objekt benutzen können, müssen Sie den Speicherplatz dafür



festlegen. Dies geschieht in Java mit Hilfe der `new` Anweisung.

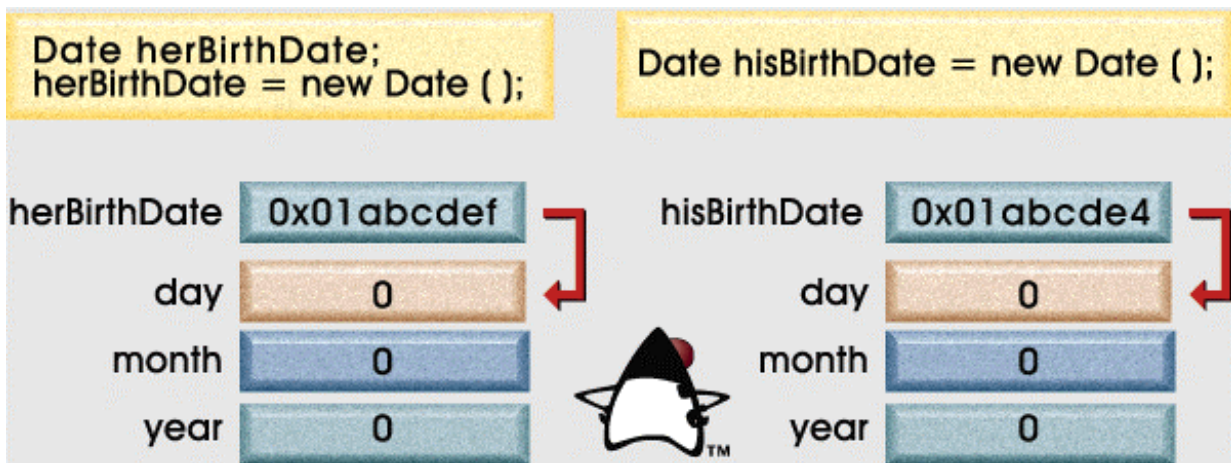
```
Date myBirthDate;  
myBirthDate = new Date();
```

Die erste Anweisung reserviert den Speicherplatz für die Referenz, die zweite für die Objektdaten. In unserem Beispiel wird mit `new` Platz reserviert für drei `Integer` Variablen. Diese werden auch gleich mit dem Standardwert `0` initialisiert.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

Generell werden die Objekte mit `new Konstruktor()` kreiert. Jedes Objekt erhält seine eigene Referenz, mit dessen Hilfe es angesprochen werden kann. Und mit der Referenzvariable und dem *dot* Operator kann man dann auch auf die Datenfelder und Methoden des Objekts zugreifen.

Das Beispiel in der folgenden Abbildung zeigt dies anschaulich. Wir haben pro Objekt eine Referenzvariable und dann noch die einzelnen Members.



1.4.3.4. Arbeiten mit Referenzvariablen

Java behandelt Variablen, welche von einem bestimmten Klassentyp sind, als Referenzvariablen. Dies hat Konsequenzen. Betrachten wir dazu ein einfaches Beispiel:

Wir deklarieren und initialisieren einige Variablen und Zeichenketten (also Objekte vom Datentyp oder Objekttyp `String`).

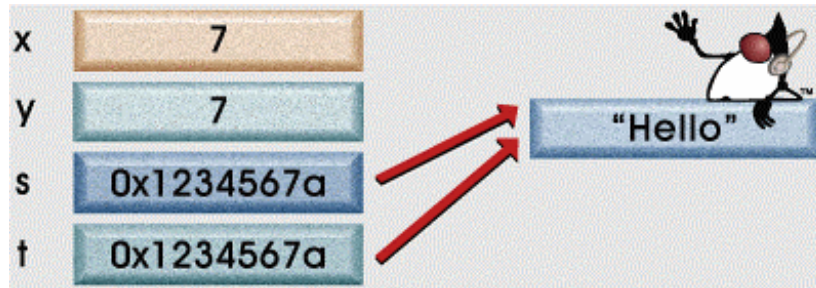
```
int x = 7;  
int y = x;  
String s = "Hello";  
String t = s;
```

Von den vier Variablen sind zwei Integer, also Basisdatentypen, und zwei Zeichenketten, also Objekte.

Die Zuweisung `y = x` ist eine echte Zuweisung, 'echt' weil der Wert von `x` der Variable `y` zugewiesen wird.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

Die Zuweisung `t = s` bezieht sich auf die Referenzvariable, die zum Zeichenkettenobjekt `s`



gehört. In `t` ist also nicht die Zeichenkette "Hello" abgespeichert, sondern der Wert der Referenzvariable `s`, welche auf das Objekt mit dem Wert "Hello" weist.

1.4.3.5. Default Initialisierung und `null` als Referenzwert

Immer wenn Sie `new` ausführen, um Speicherplatz für ein Objekt zu reservieren, initialisiert Java den Speicherbereich und setzt ihn auf '`null`' zurück. Was heißt zurücksetzen?

- falls es sich bei der Variable, die initialisiert werden soll, um eine numerische Variable handelt, dann ist der Initialisierungswert numerisch `0`.
- falls es sich um eine Boole'sche Variable handelt, ist der Initialisierungswert `false`
- falls es sich um eine Referenzvariable handelt, ist der Initialisierungswert `null`.

`null` ist ein spezieller Wert, der anzeigt, dass die Referenz auf *kein* Objekt verweist. Damit kann man einfach ausdrücken, dass ein Objekt nicht mehr benötigt wird, oder nicht existiert. Das Laufzeitsystem kann Objekte suchen, deren Referenzwert `null` ist, auf die also nicht mehr gezeigt wird. Diese Objekte kann der Garbage Collector vernichten und aus dem System entfernen.

Die Standardinitialisierung geschieht lediglich für Member Variablen, nicht bei Methoden oder lokalen Variablen. Wir kommen auf diesen Punkt nochmal zurück und erklären die Unterschiede noch genauer.

Variablen, welche innerhalb einer Methode definiert werden, werden auch als *automatic*, *local*, *temporary* oder *stack* Variablen bezeichnet, je nach Präferenz des Sprechers.

Member Variablen werden kreiert, falls Sie `new XYZ()` aufrufen. Diese Variablen existieren, solange wie das Objekt existiert.

Automatische Variablen werden kreiert, falls eine Methode aufgerufen wird; sie werden zerstört, sobald die Methode wieder verlassen wird. Deswegen bezeichnet man solche Variablen auch als temporär.

1.5. Modul 3 : Ausdrücke und Kontrollflusssteuerung II

Wir haben einfache Ausdrücke und einfache Kontrollstrukturen kennen gelernt, beispielsweise für die Bearbeitung von Schleifen. Jetzt wollen wir unser Wissen vertiefen und ausbauen.

1.5.1. Einleitung

Falls Sie sich mit C / C++ bereits auskennen, werden Sie kaum viel Neues lernen. Viele der folgenden Konzepte stammen aus diesen Sprachen.

Sollten Sie trotzdem weiterlesen und nicht einfach diesen Modul überspringen, werden Sie einige der bereits besprochenen Themen wieder aufgreifen und vertiefen.

1.5.1.1. Lernziele

Nach dem Durcharbeiten dieses Moduls sollten Sie in der Lage sein:

- den Unterschied zwischen Member und automatischen Variablen erklären zu können
- beschreiben können, wie Member Variablen initialisiert werden
- den Fehler "possible reference before assignment" erkennen und korrigieren können
- einige Java Operatoren erläutern können
- unterscheiden zu können zwischen legalen und illegalen Zuweisungen von Basisdatentypen
- kompatible Zuweisungen zu erkennen und Basisdatentypen casten können

Dies ergänzt Ihr Wissen über Kontrollstrukturen und Basisdatentypen.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.5.2. Variablendeklarationen

Bisher haben wir folgende zwei Arten kennen gelernt, Variablen zu beschreiben:

- entweder durch direkte Angabe eines Datentyps (Basisdatentyp) : Zeile 2 unten
- oder durch Definition einer Klasse : Zeile 5 unten

Sie haben auch gelernt, dass man Variablen an zwei Orten deklarieren kann:

- innerhalb einer Methode (Zeile 5) : innerhalb der Methode `main`
- innerhalb einer Klassendefinition (Zeile 2) : für die Klasse `Point`

```
1 public class Point {
2     int x, y;
3
4     public static void main (String args[]) {
5         Point start, end;
6
7         start = new Point ();
8         end = new Point ();
9         start.x = 10;
10        start.y = 10;
11        end.x = 20;
12        end.y = 30;
13
14        System.out.println ("Start Point = X: " + start.x +
15                             " Y: " + start.y);
16        System.out.println ("End Point = X: " + end.x +
17                             " Y: " + end.y);
18
19        Point stray;
20
21        stray = end;
22
23        System.out.println ("End Point = X: " + end.x +
24                             " Y: " + end.y);
25        System.out.println ("Stray Point = X: " + stray.x +
26                             " Y: " + stray.y);
27
28        stray.x = 100;
29        stray.y = 50;
30
31        System.out.println ("End Point = X: " + end.x +
32                             " Y: " + end.y);
33        System.out.println ("Stray Point = X: " + stray.x +
34                             " Y: " + stray.y);
35
36    }
37 }
```


JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.5.2.1. Automatische und Member Variablen

Variablen, welche innerhalb einer **Methode** (beispielsweise im obigen Beispiel der Methode **main**) definiert werden, werden als *automatische, lokale, temporäre* oder *Stack Variablen* bezeichnet, je nach Präferenz des Sprechers.

Member Variablen werden kreiert, wenn das **Objekt** angelegt wird, mittels des `new XYZ()` Aufrufs. Diese Variablen existieren, solange das Objekt existiert.

Automatische Variablen werden kreiert, wenn die Methode ausgeführt wird; sie werden vernichtet, sobald die Methode verlassen wird. Deswegen bezeichnet man sie ja auch als temporär.

1.5.2.2. Variablen Initialisierung

In Java darf keine Variable einen undefinierten Wert besitzen. Deswegen werden alle Variablen sofort nach dem Kreieren eines Objekts initialisiert. Als Werte werden jene aus der unten aufgeführten Tabelle genommen.

Type	Initialisierungswert
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
boolean	false
Alle Reference Types	null

Beachten Sie, dass selbst Referenzvariablen initialisiert werden. Eine Referenz mit Nullwert zeigt auf kein Objekt. Falls Sie ungeschickterweise ein solches Objekt einsetzen möchten, wird eine **Exception** geworfen. Exceptions sind Fehler zur Laufzeit, wie wir später noch sehen werden.

Automatische Variablen (Methodenvariablen) müssen initialisiert sein, bevor sie eingesetzt werden können. Der Compiler prüft den Programmcode und bestimmt, ob die Variablen auch wirklich initialisiert werden bevor sie eingesetzt werden. Falls dies nicht der Fall ist, wird der Compiler einen Fehler anzeigen.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

Beispiel

```
public void doComputation() {
    int x = (int)(Math.random() * 100);
    int y;
    int z;
    if (x > 50) {
        y = 9;
    }
    z = y + x;    // die Variable y wird eingesetzt bevor sie initialisiert
                // wurde, da die if Abfrage an der Initialisierung
                // vorbeiführen könnte (falls x<50 ist)
}
```

1.5.3. Operatoren

Die grundlegenden Java Operatoren sind ähnlich wie jene in C oder C++. Wichtig ist aber, dass im Gegensatz zu C/C++ die Auflösungsreihenfolge eindeutig festgelegt wurde.

Die folgende Tabelle zeigt die Operatorpräferenz. ("L to R" bedeutet von links nach rechte (left-to-right) assoziativ; "R to L" bedeutet right-to-left assoziativ).

Operatoren mit der selben Präzedenz erscheinen auf der selben Zeile in der Tabelle.

postfix operators	[] . (params) expr++ expr--	
unary operators	++expr --expr +expr -expr ~ !	R to L
creation or cast	new (type)expr	R to L
multiplicative	* / %	L to R
additive	+ -	L to R
shift	<< >> >>>	L to R
relational	<> = <= instanceof	L to R
equality	== !=	L to R
bitwise AND	&	L to R
bitwise exclusive XOR	^	L to R
bitwise inclusive OR		L to R
logical AND	&&	L to R
logical OR		L to R
conditional	?:	R to L
assignment	= += -= *= /= %= = <<= = &= ^= =	R to L

1.5.3.1. Zeichenketten mit + verbinden

Der + Operator verbindet String Objekte und liefert eine neues Objekt vom Typ String:

Beispiel

```
String salutation = "Dr.";
String name = "Pete" + "Seymour";
String title = salutation + name;
```

Das Ergebnis in der String Variable title wäre ein Wert von "Dr. Pete Seymour".

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

Falls einer der Operanden des + Operators ein String Objekt ist, dann wird das andere Argument automatisch in eine Zeichenkette umgewandelt.

Beispiel

```
int value1 = 10;int value2 = 20;
System.out.println ("Value 1 = " + value1 + " and Value 2 = "+ value2);
```

Diese implizite Konversion eines Basisdatentyps oder eines Objekts in eine Zeichenkette geschieht nur im Falle des + oder des += Operators in Ausdrücken, in denen String Objekte vorhanden sind. Falls Sie die Ausgabe als Zeichenkette erzwingen wollen, schreiben Sie einfach `System.out.println(""+.....);`

1.5.4. Evaluation Relationaler und logischer Operatoren

1.5.4.1. Short-Circuit logische Ausdrücke

Java unterstützt die üblichen relationalen und logischen Operatoren. Diese Operatoren liefern einen `boolean` Wert zurück. int Umwandlung in `boolean` geschieht *nicht*, also im Gegensatz zu C/C++.

Beispiel

```
int i = 1;
if (i) {} // Compilerfehler : 1 wird nicht als true interpretiert
if (i != 0) {} // korrekt
```

Die Operatoren `&&` und `||` führen sogenannte *short-circuit* logische Ausdrücke aus.

Beispiel

```
String unset = null;
if ((unset != null) && (unset.length() > 5)) {
    // jetzt wird unset eingesetzt
}
```

Die *Evaluation* der Ausdrücke geschieht *schrittweise*:

falls beispielsweise in einer UND Verknüpfung (`&&`) ein Ausdruck `false` ist, wird die weitere Evaluation *abgebrochen*, da das Ergebnis so oder so `false` (`false && x = false`) liefern würde.

Ebenso bei einer ODER Verknüpfung (`||`) . Falls eine der Bedingungen `true` ist, wird nicht weiter evaluiert, da `true || x = true` ist.

1.5.4.2. Bitweise Evaluation

Bei der Verwendung von `&` und `|` wird die Evaluation vollständig und bitweise ausgeführt. Dies ist aber nur bei `int` Variablen möglich. Zur Illustration hier ein einfaches Java Programm, welches die bitweise Verknüpfung illustriert:

```
public class BitweiseOperatorenDemo {
    public static void main(String[] args) {
        ...
        System.out.println(" "+iInt.toBinaryString(iV1&iV2)+ "=
        "+iInt.toBinaryString(iV1)+" & "+iInt.toBinaryString(iV2)+" :
        "+(iV1&iV2)+" = "+iV1+" & "+iV2);
        ...
    }
}
```


JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

```
-4= -8 >>1 :
1111111111111111111111111111111100 = 11111111111111111111111111111111000 >> 1

-2= -4 >>1 :
1111111111111111111111111111111110 = 11111111111111111111111111111111100 >> 1
-1= -2 >>1 :
1111111111111111111111111111111111 = 1111111111111111111111111111111110 >> 1
```

Links Shift : hinten wird eine 0 angefügt

```
10= 5 <<1 : 1010 = 101 << 1
20= 10 <<1 : 10100 = 1010 << 1
40= 20 <<1 : 101000 = 10100 << 1
80= 40 <<1 : 1010000 = 101000 << 1
160= 80 <<1 : 10100000 = 1010000 << 1
320= 160 <<1 : 101000000 = 10100000 << 1
```

Links Shift : hinten wird eine 0 angefügt

```
-10= -5 <<1 :
111111111111111111111111111110110 = 11111111111111111111111111111011 << 1

-20= -10 <<1 :
1111111111111111111111111111101100 = 111111111111111111111111111110110 << 1

-40= -20 <<1 :
11111111111111111111111111111011000 = 1111111111111111111111111111101100 << 1

-80= -40 <<1 :
111111111111111111111111111110110000 = 11111111111111111111111111111011000 << 1

-160= -80 <<1 :
1111111111111111111111111111101100000 = 111111111111111111111111111110110000 << 1

-320= -160 <<1 :
11111111111111111111111111111011000000 = 1111111111111111111111111111101100000 << 1
```

Sie finden das Programm, welches die obige Ausgabe generiert, auf dem Server.

Hier ein Programmfragment, welches die wesentliche Umwandlung und Berechnung zeigt:

```
System.out.println(" "+(iVar>>1)+ " = "+iVar+" >>1 :
"+iInt.toBinaryString(iVar>>1)+" = "+iInt.toBinaryString(iVar)+" >> 1" );
```

Der *logische* und *vorzeichenlose* Rechts-Shift-Operator ">>>" kümmert sich nicht darum, ob es sich um eine positive oder negative Zahl handelt. Es wird immer eine 0 oder bei mehrfach Verschiebung entsprechend viele 0en vorne angefügt.

Der Effekt dieser Operation ist harmlos bei positiven Zahlen; bei negativen Zahlen sieht die Situation etwas anders aus, da auf einen Schlag, durch die führende 0, aus der negativen Zahl eine (sehr grosse) positive Zahl resultiert.

```
2147483645= -5 >>>1 :
0111111111111111111111111111101 = 111111111111111111111111111111011 >>> 1
```

Also Vorsicht!

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

Hier noch einmal ein Beispiel für die beiden Operatoren:

```
1010 ... >> 2 gibt 111010 ...
1010 ... >>> 2 gibt 001010 ...
```

Das Programmfragment ist identisch mit jenem für den vorzeichenbehafteten Shift-Operator.

```
do {
    Integer iInt = new Integer(iVar);
    System.out.println(" "+(iVar>>>1)+ " = "+iVar+" >>>1 :
        "+iInt.toBinaryString(iVar>>>1)+" = "+iInt.toBinaryString(iVar)+"
        >>> 1" );
    iVar = iVar >>>1;}
while(iVar >0);
```

Die Umwandlung in ein `Integer` Objekt geschieht lediglich, damit die Ausgabe nett formatiert werden kann, da die `Integer` Klasse eine Methode `toBinaryString(...)` zur Verfügung stellt.

Selbsttestaufgaben

- 1) Überlegen Sie sich die Lösung für folgende Aufgaben bevor Sie sich die Programme anschauen.
 - a) was passiert, wenn Sie eine negative Zahl, die einige Nullen enthält, soweit nach links verschoben wird (`<<`) bis eine der Nullen ganz vorne steht, also eigentlich eine positive Zahl dargestellt wird?
 - b) was passiert, wenn eine positive Zahl, die einige Einsen enthält, soweit nach links verschoben wird (`<<`) bis eine der Einsen ganz vorne steht, also eigentlich eine negative Zahl dargestellt wird?
- 2) Schauen Sie sich das Programm genau an und versuchen Sie durch Eingabe einiger Zahlenbeispiele zu verstehen, wie diese Shift Operatoren funktionieren.

1.5.6. Promotion und Typumwandlung (Casting)

In Java kann man zwar viele Datentypen ineinander umwandeln; allerdings geschieht nur wenig automatisch!

Etwas was immer funktioniert, ist die Umwandlung irgend eines Datentyps in eine Zeichenkette. Wir haben dies in den obigen Programmen mehrfach verwendet, um irgendeinen Datentyp in eine Zeichenkette umzuwandeln und auszugeben.

In Java funktioniert eine Konversion immer dann, wenn das Ziel *weiter* ist als als der ursprüngliche Ausdruck oder die ursprüngliche Variable. Weiter heisst, dass das Ziel mehr Bits enthält. Eine Reduktion der Länge führt zu einer Fehlermeldung, ausser die Umwandlung wird explizit verlangt (casting).

Beispiele

```
long bigval = 6;           // 6 ist ein int Typ, OK
int smallval = 99L;       // 99L ist ein long, illegal

float z = 12.414F;        // 12.414F ist float, OK
float z1 = 12.414;        // 12.414 ist double, illegal
```

Falls wir den Datentypen explizit umwandeln, können wir auch längere in kürzere Datentypen umwandeln, wobei aber unter Umständen signifikante Stellen verloren gehen, wie wir schon gesehen haben.

```
long bigValue = 99L;
int squashed = (int)(bigValue); // explizites Abschneiden
```

Wir müssen also unterscheiden zwischen der *Konversion* und dem *Casting*, dem bewussten Umwandeln, unter in Kauf nehmen von Genauigkeitsverlusten oder sogar massiven Änderungen des Werts der Variablen.

Zudem müssen wir den Unterschied verstehen, zwischen Umwandlungen primitiver Datentypen (Basisdatentypen) und jener von Objekten, welche ebenfalls möglich ist, sofern es sich um Polymorphismus handelt.

1.5.6.1. Konversion primitiver Datentypen - Anweisungen

Anweisungskonversionen geschehen, wenn ein Wert einer Variable eines andern Datentyps zugewiesen wird.

Beispiel

```
int i=10;
double d;
d=i;          // korrekte Konversion von i (=10) in double
```

Die Anweisung geschieht in mehreren Schritten:

- 1) als erstes wird der Integer Wert 10 in einen Double Wert umgewandelt:
dies geschieht, indem die 10 durch Nullen ergänzt wird
10 wird 10.0000....
- 2) der Double Wert wird der Double Variable zugewiesen

Den Vorgang bezeichnet man im englischen als *widening*, als Erweiterung, Ausdehnung.

Aufpassen muss man bei Literalen, weil in diesem Fall der Standarddatentyp bekannt sein muss. Ist beispielsweise 1.34 eine Float oder eine Double Zahl?

Beispiel

```
double d=1.34;          // Gleitkommazahlen sind double
short s;
s = d;                  // das geht offensichtlich nicht
                        // wegen Verlust von Stellen
d = (short)d;          // hier wird eine Umwandlung erzwungen
```

Generelle Regeln:

- `boolean` können in keine andere Datentypen konvertiert werden
- nicht `boolean` **können** in einen andern nicht `boolean` Datentyp umgewandelt werden, falls es sich um eine *erweiternde Konversion*, engl. *widening conversion* handelt.
- nicht `boolean` können **nicht** in einen andern nicht `boolean` Datentyp umgewandelt werden, falls es sich um eine *einschränkende Konversion* engl. *narrowing conversion* handelt.

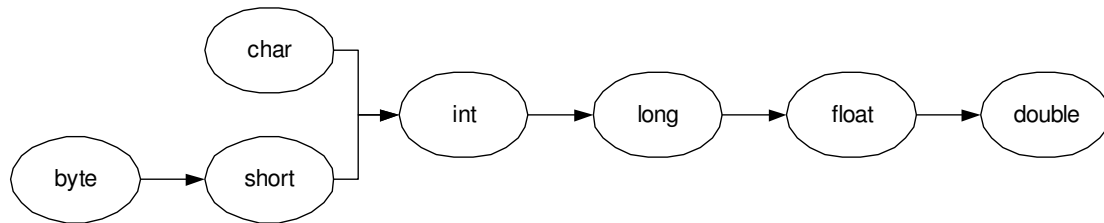
Bei einer erweiternden Konversion wird der Wert des Datentyps so geändert, dass zwar der Wert erhalten bleibt, aber im einfachsten Fall entweder vorne oder nach dem Komma Nullen hinzugefügt werden. Damit passt die Variable dann in den Speicherbereich des Zieldatentyps.

Dies entspricht den Beispielen oben.

Warum diese Regeln gelten, liegt daran, dass bei Einhaltung dieser Regeln keine Verluste der Genauigkeit und signifikanter Bits geschieht.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

Anschaulich finden Sie dies im Begleittext, hier der Einfachheit eins zu eins reproduziert:



Die Konversion von Links nach rechts ist *gestattet*, jene von rechts nach links nicht.

1.5.6.2. Konversion bei Literalen

Einen Spezialfall gibt es bei der Zuweisung von Literalen:

- Standarddatentypen numerischer Variablen sind entweder `int` oder `double`.
- Literale vom Typ `int` werden automatisch in `byte`, `short` oder `char` umgewandelt.

Beispiel

```
byte b = 1;           // legale Zuweisung eines Literals
short s = 2;         // dito
char c = 3;          // auch das geht: '\u0003' (siehe Debugger)
double d=4;          // siehe Beispielprogramm (d=4.0)
...
b = s;               // illegal, weil s kein Literal ist
```

Sie finden auf dem Server / dem Web ein einfaches Programm, welches die obigen Zuweisungen testet und verifiziert. Falls Sie das Programm mit dem Debugger starten und die Werte anzeigen lassen, sehen Sie, dass das Zeichen als Unicode zugewiesen wird, wie oben im Kommentar beschrieben.

1.5.6.3. Konversion der Basisdatentypen bei Methodenaufrufen

Die obigen Konversionsregeln gelten auch, falls Sie Methoden aufrufen und der Parameter nicht mit dem Datentyp in der Deklaration übereinstimmt.

Beispiel

```
float winkel = 2.6789f;
double d = Math.cos(winkel); //cos() erwartet double
```

In diesem Methodenaufruf (Cosinus Funktion) wird beim obigen Aufruf das Argument zuerst in eine Double Zahl konvertiert und dann an die mathematische Funktion weitergeleitet.

1.5.6.4. Arithmetische Promotion bei Basisdatentypen

Die relativ einfache Umwandlung von Basisdatentypen in andere Basisdatentypen, wie wir sie kennen gelernt haben, ist nur ein Teil der Konversionsregeln.

Wichtig ist die Konversion auch bei allen arithmetischen Operationen, bei denen unterschiedliche Basisdatentypen miteinander verknüpft werden.

Dabei müssen wir unterscheiden, zwischen unären (+, -, ++, -- : ein Operand) und binären (+, -, &, *, ^, /, |, %, >>, >>>, << : zwei Operanden) Operatoren.

Regeln

- unäre Operatoren
 - falls der Operand `byte`, `short` oder `char` ist, wird in `int` konvertiert, ausser der Operator ist `++` oder `--`, dann geschieht nichts.
 - sonst geschieht keine Konversion
- binäre Operatoren
 - falls ein `double` Operand vorkommt, wird in `double` konvertiert
 - sonst falls ein `float` Operand vorkommt, wird in `float` konvertiert
 - sonst falls ein `long` Operand vorkommt, wird in `long` konvertiert
 - sonst werden beide Operanden in `int` konvertiert.

Neben diesen Regeln müssen wir die Präzedenzregeln (siehe Abschnitt über Operatoren weiter vorne) beachten. Im Wesentlichen muss man von links nach rechts auflösen, mit Ausnahme der `++`, `+=`, ... Operatoren.

Beispiel

```
sei s short, i int, f float und d double
if (-s*i >= f/d) :
1. -s ist unär; s wird in int umgewandelt
2. (-s)*i ist eine Multiplikation zweier int also int
3. f ist float und wird wegen d in eine double umgewandelt. Die
   Division der beiden double liefert wieder eine double.
4. (-s*i) als int wird in einen double umgewandelt und mit dem
   double (f/d) verglichen.
5. der Vergleich liefert einen boolean.
```

1.5.6.5. Einfache Datentypen und Typenumwandlung (Casting)

Die bewusste Datentypumwandlung lebt mit dem Risiko, dass Daten verändert werden. Wir haben dies bereits an einigen Beispielen gesehen. Die Veränderung der Datenwerte geschieht immer dann, wenn der ursprüngliche Datentyp im Zieldatentyp nicht Platz hat und Einsen abgeschnitten werden.

Beispiel

```
short sc=259;
byte bc = (byte)sc;    // 256 werden abgeschnitten
// bc = 3
```

Sie finden auf dem Server / Web ein Beispiel zur Typenkonversion, in dem auch dieses Beispiel berechnet und ausgegeben wird (binär und dezimal).

Regeln

- jeder nicht `boolean` Datentyp kann in einen beliebigen nicht `boolean` Datentyp umgewandelt werden.
- `boolean` Daten können in keinen andern Datentyp konvertiert werden; beliebige Datentypen ausser `boolean` können nicht in `boolean` umgewandelt werden.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.5.7. Quiz - Ausdrücke und Kontrollflussteuerung II

Zur Abwechslung noch etwas zur Festigung Ihres Wissens. Sie sollten jetzt Grundkenntnisse über Ausdrücke und Kontrollflussteuerung haben. Wie gut Ihr Wissen ist, können Sie gleich testen.



Welche der folgenden Aussagen zu den Shift Operatoren (>>) und (>>>) ist korrekt?⁶

- a) >> gilt für Integer, >>> für Float
- b) >> shiftet, >>> rotiert
- c) >> rotiert, >>> shiftet
- d) >> shiftet ohne Beachtung des Vorzeichens, >>> berücksichtigt das Vorzeichen
- e) >> shiftet unter Beachtung des Vorzeichens, >>> berücksichtigt das Vorzeichen nicht

Von welchem Datentyp ist das Ergebnis der folgenden Berechnung, falls d double, i int, f float und b byte ist?

$$3.14*2*d*i*f*b^7$$

- a) Compilerfehler
- b) double
- c) float
- d) long
- e) int
- f) short
- g) byte

⁶ d)

⁷ double also b)

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.5.8. Zusammenfassung - Ausdrücke und Kontrollflusssteuerung II

In diesem Modul haben Sie mehrere Konzepte kennen gelernt, welche Ihnen Hintergrundwissen über Java vermitteln.

In diesem Modul haben Sie gelernt:

- zu unterscheiden zwischen Member und automatischen Variablen
- wie die Initialisierung der Member Variablen anläuft
- wie Sie den Compiler Fehler "Possible reference before assignment" schon erkennen können, bevor Sie das Programm übersetzen, oder leicht beheben können.
- legale und illegale Zuweisungen zu unterscheiden und zu erkennen und allfällig nötige Konversionen durchzuführen.
- wie die Shift Operatoren und die logischen Operatoren im Detail funktionieren.

1.6. Modul 4 : Arrays II

Arrays sind einfache Sammlungen von Variablen des selben Typs. Der Zugriff auf die einzelnen Komponenten eines Arrays geschieht mit Hilfe eines Integer Indices.

1.6.1. Einleitung

Wir haben einfache Fälle von Arrays bereits kennen gelernt. Jetzt wollen wir kurz einige Punkte wiederholen und Neues dazu lernen.

1.6.1.1. Lernziele

Nach dem Durcharbeiten sollten Sie in der Lage sein

- Arrays für Basisdatentypen, Klassen (Instanzen von Klassen) oder Array Datentypen zu deklarieren und zu kreieren
- Arrayelemente zu initialisieren
- die Anzahl Elemente in einem Array zu bestimmen
- die von Java zur Verfügung gestellte Methoden zum Kopieren von Arrays zu benutzen

Wir haben uns bereits einmal mit Arrays befasst. Allerdings hat es sich in der Übung gezeigt, dass viele Punkte unklar geblieben sind.

Aus diesem Grund wiederholen wir hier einige Punkte bewusst noch einmal.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.6.2. Deklarieren von Arrays

Arrays können sowohl für Basisdatentypen als auch für Objekte deklariert und kreiert werden.

Die zwei folgenden Beispiele zeigen dies. Das eine Array besteht aus Zeichen, das andere aus Punkten, also aus Instanzen einer Klasse.

```
public class Point {
    int xValue, yValue;
    char s[];      // Zeichenarray
    Point p[];    // Objektarray
}
```

Soweit stimmt das Format mit jenem aus C/C++ überein. Java gestattet allerdings auch noch eine alternative Form, nämlich jene, bei der die Klammern nicht bei der Instanz, sondern beim Datentyp stehen:

```
Point p[ ]; // ist äquivalent zu
Point[] p;
```

1.6.3. Kreieren von Arrays

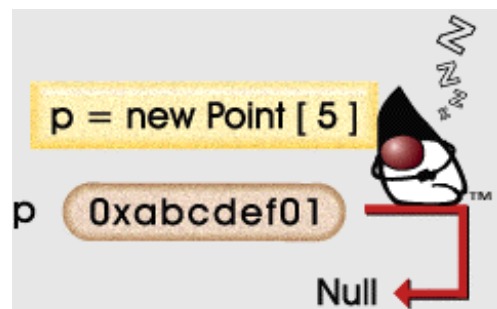
In Java ist ein Array ein Objekt, selbst wenn das Array lauter Elemente eines Basistyps enthält. Und wie bei Objekten üblich, kreiert die Deklaration des Objekts das Objekt selbst noch nicht, sondern lediglich eine Referenzvariable.

Die Figur unten soll dies veranschaulichen.

Das neue Array wird mit dem `new` Schlüsselwort folgendermassen kreiert:

```
char s[] = new char[20];
Point p[] = new Point[5];
```

Die erste Zeile kreiert ein **Array** bestehend aus 20 Zeichen. Die zweite Zeile kreiert ein Array (Objekt) bestehend aus 5 Variablen des Typs Point. Die Punkte selber werden aber damit noch *nicht* kreiert.



Falls Sie Punkt **Elemente** in Ihrem Array haben wollen, müssen Sie diese jetzt noch kreieren:

```
p[0] = new Point();
p[1] = new Point();
...
```

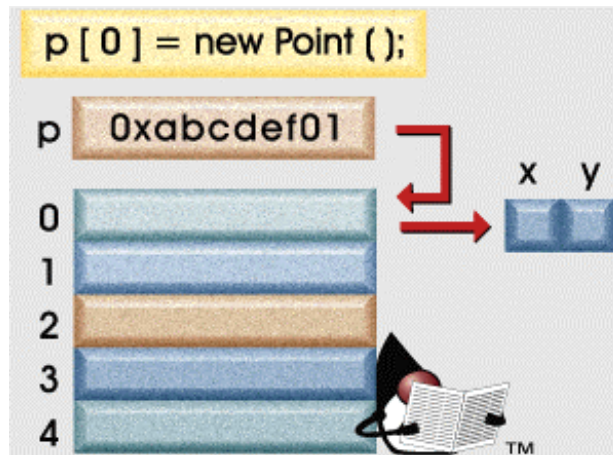
JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.6.4. Initialisierung von Arrays

Falls Sie ein Array kreieren, wird jedes Element initialisiert.

Im Fall des Arrays bestehend aus Zeichen, wird jedes Element auf `null` (`\u0000` - `null`) gesetzt.

Im Fall des Arrays bestehend aus Objekten, wird jedem Element ein `null` Objekt zugewiesen. Damit wird ausgedrückt, dass die Objekte, welche im Array Objekt abgespeichert werden sollen, noch nicht existiert.



Nach der Anweisung

```
p[0] = new Point();
```

weist das erste Element des Arrays auf ein reales Punktobjekt. Die Initialisierung gehört zum Teil ins Kapitel Security. Nicht jeder kann beliebige Objekte anlegen und jede Variable muss initialisiert werden.

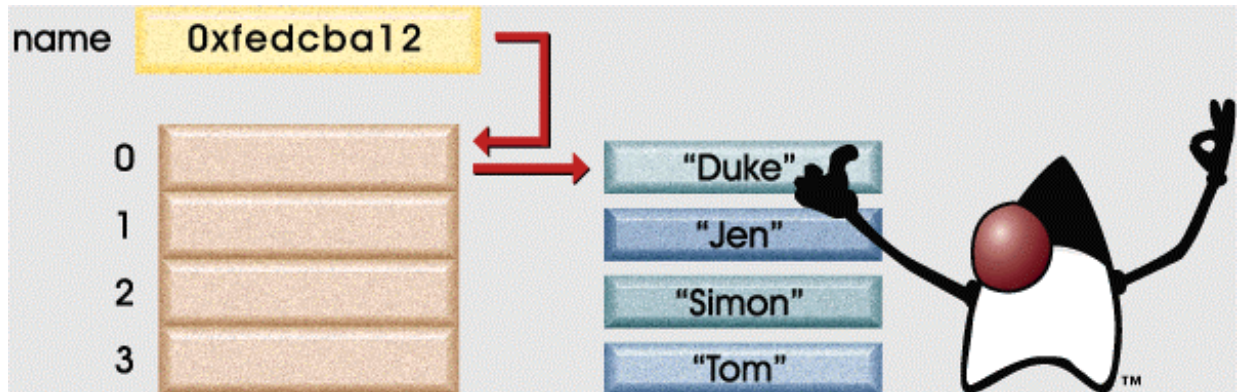
Wie wir bereits gesehen haben, kann man das mehrstufige Verfahren, zum Definieren, Anlegen und Initialisieren eines Arrays auch abkürzen.

```
String names[] = {  
    "Duke",  
    "Jen",  
    "Simon",  
    "Tom"  
};
```


JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

Dieses abgekürzte Verfahren ist äquivalent zu folgenden einzelnen Anweisungen:

```
String names[];           // Deklaration
names = new String[4];    // Referenzvariable
names[0] = "Duke";        // Initialisierungen
names[1] = "Jen";
names[2] = "Simon";
names[3] = "Tom";
```



JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.6.5. Arrays aus Arrays

Mehrdimensionale Arrays werden in Java dadurch ermöglicht, dass ein Array ein Objekt ist und es Ihnen frei gestellt ist, ein Array bestehend aus Array Objekten zu kreieren.

```
1 int twoDim [][] = new int [4] [];  
2 twoDim[0] = new int[5];  
3 twoDim[1] = new int[5];  
4 twoDim[2] = new int[5];  
5 twoDim[3] = new int[5];
```

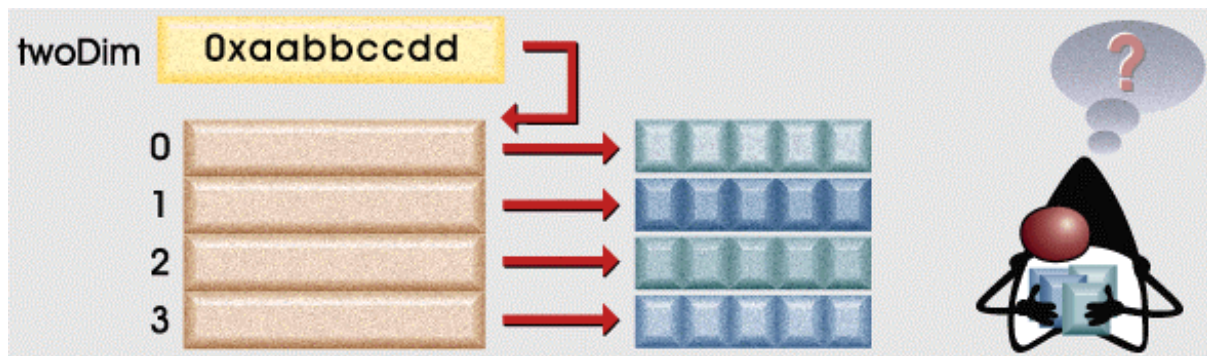
Als erstes wird ein Array Objekt kreiert.

Das Array enthält vier Elemente. Jedes dieser Elemente verweist auf ein `null` Element.

Die vier Elemente müssen wir nun separat kreieren. Dies geschieht in den Zeilen 1-5.

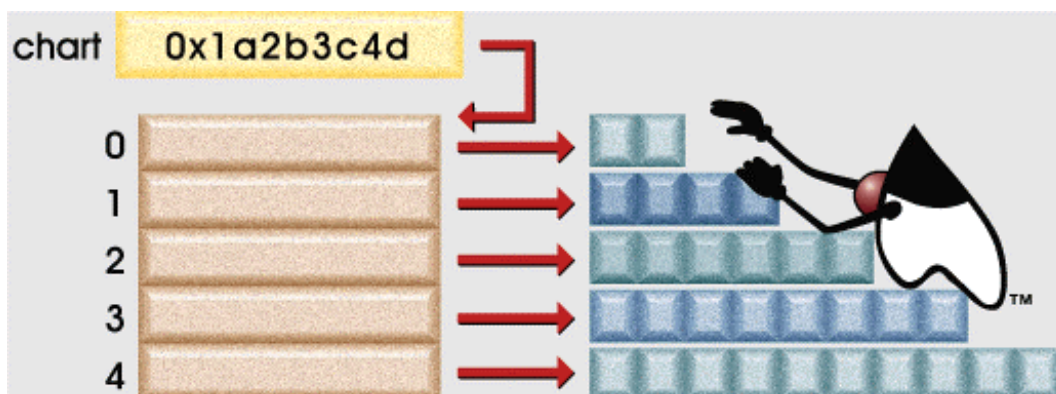
Übrigens: bei aller Flexibilität in Java, folgende Anweisung ist *illegal*.

```
int twoDim [][] = new int [] [4]; //erste Dimension muss vorhanden sein
```



Da jede Zeile separat kreiert werden muss, könnten wir auch eine dreieckige Matrix konstruieren.

```
int chart [][] = new int [5] [];  
chart[0] = new int[2];  
chart[1] = new int[4];  
chart[2] = new int[6];  
chart[3] = new int[8];  
chart[4] = new int[10];
```



1.6.6. Array Grenzen

Die obige Situation (Dreieck) ist eher selten. Daher werden Sie in der Regel gleich beide Dimensionen angeben:

```
int zweiDim[ ][ ] = new int[4][5];
```

In vielen Anwendungen müssen Sie die Grenzen eines Arrays bestimmen, also wissen, wieviele Elemente in einem Array maximal angelegt werden können. Sie können nicht so einfach bestimmen, welche Zellen auch tatsächlich belegt sind.

Die Grösse eines Arrays wird bestimmt mit Hilfe der Member Variable `length`.

Beispiel

```
int list[] = new int [10];
for (int i = 0; i < list.length; i++)
    // irgend etwas.
```

Die Länge des Arrays wird hier mit Hilfe des Attributs `length` abgefragt. Dies ist besser, als mit Hilfe einer fixen Länge, hier 10. Allfällige Änderungen beim Array lassen sich auch leichter anpassen, wenn Sie generelle Grössen, an Stelle der fixen Länge 10 verwenden.

Damit ist auch klar, wie die Länge einer Matrix abgefragt werden muss:

```
l1 = matrix.length; l2=matrix[0].length; // x, y Länge der Matrix
```

1.6.7. Kopieren von Arrays - `System.arraycopy`

Java bietet Ihnen die Möglichkeit ein Array (fast dynamisch) zu vergrössern:

```
int elements[] = new int[6];           // kreiert ein int[6] Array
elements = new int[10];                // überschreibt das Array
```

Das erste Array geht dabei verloren, ausser Sie haben es irgendwie gerettet. Um die Werte des alten Arrays in das neue Array zu kopieren, stellt Ihnen Java eine System Methode zur Verfügung: **`System.arraycopy()`**.

Beispiel

```
// Original Array
int elemente[] = new int [] { 1, 2, 3, 4, 5, 6 };

// neues grösseres Array
int neu[] = new int [] { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };

// kopieren aller alten Elemente in das neue Array
// Start beim Element 0; die initialisierten neuen Elemente
// werden überschrieben
System.arraycopy(elemente, 0, neu, 0, elemente.length);

// neu enthält : 1,2,3,4,5,6,4,3,2,1
```

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

Bei gleicher Länge und gleichem Typ können Sie natürlich auch einfach Referenzen kopieren: `neu1 = neu2`; Sie finden im Programm ein Beispiel dafür (selbes Array mit zwei unterschiedlichen Namen).

1.6.8. Quiz - Arrays II

Zur Abwechslung noch etwas zur Festigung Ihres Wissens. Sie sollten jetzt etwas verteilte Kenntnisse über Arrays haben. Wie gut Ihr Wissen ist, können Sie gleich testen.



Gegeben sei folgendes Programmfragment

```
public class BasicArrays {
    public static void main(String args[ ]) {
        int[ ] thisArray;
        int[ ] thirdArray = new int[ ] {1,2,3,4,5,6,7,8,9,10};

    }
}
```

Welches der folgende Fragmente kopiert `thirdArray` auf `thisArray` und ergänzt das Programm korrekt?⁸

- a) `arraycopy(thirdArray,0,thisArray,length);`
- b) `System.arraycopy(thirdArray,0,thisArray,thirdArray.length);`
- c) `System.arraycopy(thisArray,0,thirdArray,thisArray.length);`

⁸ b)

1.6.9. Zusammenfassung - Arrays II

In diesem Modul haben Sie Ihr Wissen über Arrays vertieft und vermutlich einige offene Fragen klären können.

In diesem Modul haben Sie gelernt:

- Arrays zu deklarieren und Arrays unterschiedlicher Datentypen zu kreieren.
- wie der Initialisierungsprozess abläuft und wie initialisiert wird
- den Inhalt eines Arrays zu bestimmen
- wie man Arrays kopiert.

1.7. Module 5 : Objekte und Klassen

1.7.1. Einleitung

In einem früheren Modul haben wir die grundlegenden Konzepte der Java Objekte und Klassen besprochen. In diesem Modul wollen wir nun dieses Wissen vertiefen und unser Verständnis über Objekte und Klassen festigen.

1.7.1.1. Lernziele

Nach dem Durcharbeiten dieses Moduls sollten Sie in der Lage sein:



- den Begriff "Abstrakter Datentyp" zu definieren.
- Methoden, inklusive Parametern, zu schreiben.
- das Java Schlüsselwort `this` einzusetzen, um auf Datenfelder (Data Members) einer Methode zuzugreifen.
- Methoden eines Objekts aufzurufen.
- mit Hilfe des `this` Schlüsselworts auf das aktuelle Objekt zuzugreifen.
- Ihre Kenntnisse über private Zugriffe auffrischen.
- Konstruktoren, mit und ohne Parameter, zu schreiben
- Unterklassen zu definieren
- Methoden in Unterklassen zu schreiben, welche Oberklassenmethoden überschreiben.
- überladene Methoden zu definieren und auszuführen.
- überschriebene Methoden auszuführen.
- die `import` Anweisung zu verstehen.

1.7.1.2. Orientieren Sie sich selbst



Als Vorbereitung für diesen Modul sollten Sie sich folgende Fragen stellen:

- wie kann ich eine Gruppe von Objekten des selben Typs zusammenfassen?
- wie kann ich mehrere Objekte des selben Typs effizient manipulieren?
- wie kann ich Informationen kapseln?

Falls Sie diesen Modul durcharbeiten, sollten Sie diese Fragen im Kopf behalten. Sie sollten Ihnen helfen den Text besser zu verstehen.

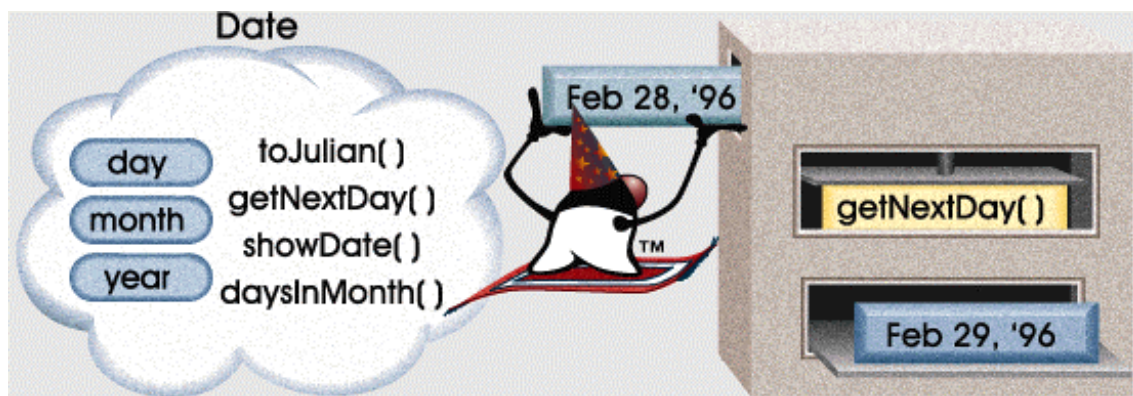
JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.7.2. Abstrakte Datentypen

Das Zusammenfassen von Datentypen, wie beispielsweise dem Datum, ist der erste Schritt zur Definition eines abstrakten Datentyps. Zusätzlich werden noch Zugriffsmethoden benötigt. Das Konzept ist nicht neu. Sie kennen es auch von den Basisdatentypen: neben den Datentypen wurden auch noch die üblichen Operationen wie Multiplikation, Division usw. definiert.

In vielen Programmiersprachen kann man zu aggregierten Datentypen Zugriffsmethoden und Hilfsmethoden definieren und dadurch eine enge Verknüpfung zwischen den Daten und den Methoden herstellen.

Nachdem wir eine Datum Datentyp definiert haben, wäre die Definition einer `getNextDay()` Methode naheliegend. Diese würde einfach das nächste akzeptable Datum liefern, Datum gemäss der Syntax für Datum Datentypen.



In einigen Programmiersprachen, inklusive Java, kann man die Deklaration der abstrakten Datentypen und Methoden eng zusammenknüpfen. Diese enge Bindung zwischen den Daten und den Methoden bezeichnet man als *abstract data type*, also abstrakten Datentyp.

In Java kann man diese enge Bindung zwischen dem `Date` Datentyp und der `getNextDay` Operation in einer Klassendeklaration zusammenfassen. Die Klasse fasst also die *Methoden* wie `getNextDay` mit der Definition des abstrakten Datentyps (aggregierte Daten) zusammen. Am Stelle von Methoden spricht man auch oft von *member function* und *function*.

Schauen wir uns den Aufbau genauer an:

```
public class Date {
    private int day, month, year;
    public void getNextDay() {
        // Ihr Programmcode, um den nächsten Tag zu berechnen
    }
}
```

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.7.2.1. Assoziation von Daten mit Code

Der Unterschied zwischen klassischen Programmiersprachen und OO Programmiersprachen besteht darin, dass im klassischen Fall eine Methode definiert würde, beispielsweise:

```
void gotoNextDay(Date d);
```

welche dann das nächste Datum bestimmt.

In OO gehen wir anders vor.

Wir beschreiben nicht Programmcode, welcher auf den Daten operiert, wie oben, wir beschreiben Daten, die mit Hilfe von Methoden sich selbst verändern:

```
Date d = ?????;           // initialisieren der Daten
d.gotoNextDay();         // gotoNextDay() operiert auf d
```

Diese Notation soll die andere Denkweise andeuten. Die Daten und Objekte führen Aktivitäten aus, nicht umgekehrt.

1.7.2.2. Methoden als Eigenschaften der Daten

Sie können auf die Datenfelder des Datentyps `Date` mittels der `.` (dot) Notation zugreifen:

```
d.day
```

Diese Notation wird interpretiert als "das `day` Feld des `Date` Objekt auf das die Variable `d` referenziert".

Der Methodenaufruf `d.getNextDay()`; wird gelesen als: "die `getNextDay` Methode des `Date` Objekts referenziert von der Variable `d`". In andern Worten, "führe die `getNextDay` Operation auf das `d` Objekt aus".

Damit wird die Methode eine Eigenschaft der Daten, keine losgelöste Prozedur.

Falls wir von *message passing* sprechen, wird damit ausgedrückt, dass Daten beauftragt werden, etwas (eine Methode) auf sich selbst anzuwenden. Diese Terminologie wird speziell von Smalltalk verwendet.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.7.2.3. Methodensyntax

In Java werden Methoden analog zu C / C++ definiert.

Syntax

```
<modifiers> <return_type> <name> (<argument_list>) <block>
```

Als Namen **<name>** können Sie einen beliebigen legalen Java Identifier wählen, sofern dieser noch nicht verwendet wird.

Der **<return_type>** beschreibt den Datentyp, der von der Methode zurück gegeben wird (`void`, ein Basisdatentyp, ein Array, ein Objekt). Da Java strikte Datentypenprüfung kennt wird die Korrektheit des Rückgabedatentyps überprüft, insbesondere darf `void` nicht fehlen!

Die Zugriffsmodifizier **<modifiers>** sind `public`, `protected`, `static`, `final` und `private`. Der `public` Zugriffsmodifizier zeigt an, dass von überall auf die Methode zugegriffen werden kann. `private` beschreibt die Situation, bei der die Methode lediglich von Methoden aus der selben Klasse aufgerufen werden darf. `protected`, `static` und `final` werden später besprochen (`protected` regelt den Zugriff innerhalb der Packages, aus Subklassen).

Die **<argument_list>** gestattet es, Argumentwerte an eine Methode zu übergeben. Elemente der Liste werden durch ein Komma getrennt. Jedes Element besteht jeweils aus dem Datentyp plus einer Variable. Beispielsweise:

```
public void addDays(int days) {  
}
```

Diese Methode besitzt einen (Eingabe) Parameter vom Typ `int`. Dieser Parameter heisst `days`. Die Methode liefert keinen Wert zurück (`void`). Der Zugriff auf die Methode ist von überall her möglich (`public`).

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.7.2.4. Pass-By-Value - Parameter werden wertmässig übergeben

Java kennt lediglich die Übergabe der Parameter "by-value"; das heisst, dass das Argument innerhalb der Methode *nicht verändert* werden kann. Anders ausgedrückt: alle Änderungen an den Parametern sind sinnlos. Und als Konsequenz: Parameterwerte sollten möglichst sofort auf lokale Variablen in der Methode kopiert werden. Mit diesen wird dann gerechnet.

Sie können beispielsweise den Parameter halbieren. Nach dem Verlassen der Methode hat der Parameter wieder den ursprünglichen Wert. Sie können ein Parameterobjekt auf `null` setzen, den Garbage Collector starten und trotzdem bleibt das ursprüngliche Objekt bestehen.

Falls Sie als Parameter ein Objekt verwenden, verhält es sich also analog: Sie können zwar die Werte der Attribute verändern, nicht aber die Referenz.

```
1 public class PassTest2 {
2
3     float ptValue;
4
5     public static void main (String args[]) {
6
7         // neue Instanz der Klasse bilden
8         PassTest2 pt = new PassTest2 ();
9
10        // ptValue setzen
11        pt.ptValue = 101f;
12
13        // Initialwert?
14        System.out.println ("Initialwert von ptValue ist: " +
15            pt.ptValue);
16
17        // ändern des Werts mittels
18        // Objektreferenz
19        pt.changeObjValue (pt);
20
21        // aktueller Wert?
22        System.out.println ("Aktueller Wert von ptValue ist: " +
23            pt.ptValue);
24    }
25
26    public void changeObjValue (PassTest2 ref) {
27        ref.ptValue = 99f;
28    }
29 }
```

Sie finden eine erweiterte Version des Programms auf dem Server. Wichtig ist, dass Sie sich bewusst sind, dass es schlecht bis unsinnig ist, mit dem Parameter weiter zu arbeiten. Der Parameter übergibt lediglich einen Wert und hat nachher keine Bedeutung mehr.

Selbsttestaufgabe

Schreiben Sie ein Programm, analog zum obigen, bei dem der Parameter intern modifiziert wird.

Wie können Sie vermeiden, dass der oben beschriebene 'Fehler' passiert?

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.7.2.5. Die `this` Referenz

Das Schlüsselwort `this` kann im Rumpf von Methoden von Instanzen, im Konstruktor oder bei der Initialisierung eingesetzt werden. Was besagt das Schlüsselwort? Im einführenden Beispiel sehen Sie, dass mit Hilfe von `this` ein eindeutiger Bezug zu einer Objektvariablen hergestellt werden kann.

Beispiel


```
public class thisDemo {
    float fKontostand=10f;
    public static void main(String[] args) {
        thisDemo demo = new thisDemo();
        System.out.println("Kontostand main : "+100);
        demo.kontoEroeffnen(100);
    }
    public void kontoEroeffnen(float fKontostand) {
        // welches fKontostand?
        System.out.println("Kontostand Parameter: "+fKontostand);
        // Objektkontostand: this bezieht sich auf diese Objekt
        System.out.println("this.Kontostand : "+this.fKontostand);
    }
}
```

Ausgabe

```
Kontostand main : 100
Kontostand Parameter: 100.0
this.Kontostand : 10.0
```

Es gibt Fälle, bei denen ist das Schlüsselwort als Präfix zwingend, wie im Demo Beispiel, um auf das richtige Datenfeld zugreifen zu können. In anderen Fällen kann man darauf verzichten.

```
public class Date {
    private int day,month,year;
    public void getNextDay(){
        this.day=this.day + 1;
        // check month end, year end, leapyear, etc...
    }
}
```



```
public class Date {
    private int day, month, year;
    public void getNextDay() {
        this.day = this.day + 1; // keyword "this" ist redundant
        // ... check month end, year end, leap year, etc ...
    }
}
```

```
public class Date {
    private int day, month, year;
    public void getNextDay() {
        day = day + 1; // kein `this.` vor `day`
        // ... check month end, year end, leap year, etc ...
    }
}
```

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.7.3. Überladen von Methoden

Oft benötigt man Methoden, welche das Gleiche erledigen, wie andere Methoden, aber mehr oder weniger oder andere Parameter besitzt. Sie kennen bereits ein typisches Beispiel für das Überladen von Methoden, die `print()` Funktion / Methode. Ein anderes Beispiel wäre die Umformattierung des Datums: Sie können das Datum auf unterschiedliche Art und Weise eingeben: `dd-mm-yy`, `dd-mmm-yyyy`, `dd/mm/yy` ...

Bei der `print` Methode existieren eigentlich unterschiedliche Versionen, eine für `int`, eine für `float`, ... Wir könnten also zum Beispiel mehrere Methoden mit individuellen Namen verwenden: `printInt()`, `printString()`, ..

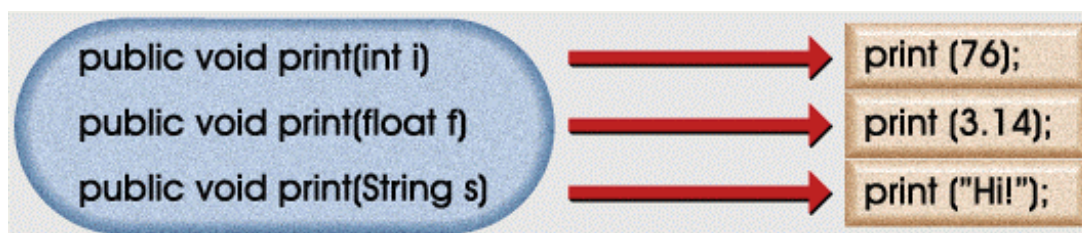
Das würde das Drucken aber auch nicht vereinfachen! Die bessere Lösung ist sicher das Überladen einer und nur einer `print` Methode, die in der Lage ist, die unterschiedlichen Parametertypen zu verarbeiten.

Java und einige andere Programmiersprachen, gestatten es dem Benutzer, den Namen einer Methode mehrfach zu verwenden. Die Unterscheidung, welche Methode konkret eingesetzt werden muss, ergibt sich aus dem Datentyp des Arguments.

In unserem Beispiel mit der `print` Methode, müssten wir beispielsweise drei Methoden definieren:

1. eine zum Drucken von `int`'s
2. eine zum Drucken von `float`'s
3. eine zum Drucken von Zeichenketten, `String`'s

```
public void print(int i)
public void print(float f)
public void print(String s)
```



Beim Überladen der Methoden müssen einige Regeln beachtet werden:

- die Argumentliste muss eindeutig sein, damit unterschieden werden kann, welche Methode konkret zum Einsatz kommt.
- die Rückgabe der unterschiedlichen Methoden kann unterschiedlich sein. Unterschiede bei der Rückgabe sind aber nicht ausreichend. Die Argumenteliste muss unterschiedlich sein.

Selbsttestaufgabe

schreiben Sie eine eigene Klasse `Datum`. Diese Klasse kann beispielsweise bestimmen, ob es sich um ein Schaltjahr handelt, welcher Wochentag ist, der wievielte Tag im Jahr ein bestimmtes Datum ist, zu welcher Woche ein bestimmter Tag gehört.

Datum : 1 String `dd/mm/yy`, `dd-mm-yy`, `dd-mmm-yyyy`, `dd/mmm/yyyy`; `dd mm yy` (3 Zahlen); `dd mmm yyyy` (3 Strings) (`12-01-98`, `12/01/98`, `12-Jan-1998`, `12/Jan/1998`).

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.7.4. Vererbung und das Überschreiben von Methoden

Zur Erinnerung:

Objekte werden schrittweise angelegt

- als erstes wird Speicherplatz für das neue Objekt reserviert und eine Defaultinitialisierung vorgenommen
- dann folgen die expliziten Initialisierungen
- und schliesslich wird der Konstruktor ausgeführt

Wir haben auch gesehen, dass mit der "is-a" Frage festgestellt werden kann, ob eine Vererbung sinnvoll ist oder nicht.

Schauen wir uns die folgenden zwei Klassenbeschreibungen an:

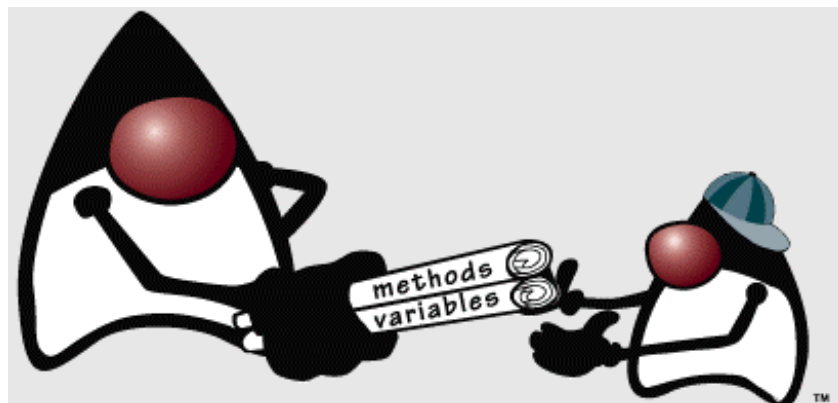
```
public class Employee {
    private String name;
    private Date hireDate;
    private Date dateOfBirth;
    private String jobTitle;
    private int grade;
    ...
}

public class Manager {
    private String name;
    private Date hireDate;
    private Date dateOfBirth;
    private String jobTitle;
    private int grade;
    private String department;
    private Employee [] subordinates;
    ...
}
```

Dann ist klar, dass wir eine neue Klasse und eine Erweiterung dieser Klasse definieren könnten. Dies könnte etwa folgendermassen aussehen:

```
public class Employee {
    private String name;
    private Date hireDate;
    private Date dateOfBirth;
    private String jobTitle;
    private int grade;
    ...
}

public class Manager extends Employee {
    private String department;
    private Employee [] subordinates;
    ...
}
```



JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

Der Manager wird damit zu einem speziellen Angestellten, wie dies in der Realität auch der Fall ist.

Der Manager erbt alle Datenfelder und Methoden, aber nicht die Konstruktoren!

Den Polymorphismus sehen wir an obigem Beispiel deutlich: ein Objekt kann mehrere Formen haben, hier beispielsweise `Manager` oder `Employee`.

In Java, wie in den meisten objektorientierten Sprachen, kann man anstelle eines Objekts auch eine Variable des Oberklassentyps verwenden.

Betrachten wir ein Beispiel:

```
Employee e = new Manager();
```

Diese Anweisung ist korrekt. Die Frage ist, was kreiert wird. Nach der Deklaration von `e` als `Employee` könnte man mit `e` lediglich `Employee` Felder und Methoden verwenden. Im obigen Fall wird aber der Konstruktor der `Manager` Klasse aufgerufen. Es wird ein Objekt kreiert, welches sowohl `Manager` als auch `Angestellter` ist. Beim Methodenaufruf werden aber die `Manager` Methoden verwendet.

Selbsttestaufgabe

Implementieren Sie das besprochene Beispiel und verifizieren Sie die obige Aussage:
kreieren Sie eine Klasse für `Angestellte`, `Manager` als Erweiterung und ein Testprogramm, in dem Sie eine Instanz wie oben bilden und versuchen auf ein Datenfeld oder eine Methode der Unterklasse zuzugreifen.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.7.4.1. Der `instanceof` Operator

Wenn wir komplexere Methodenaufrufe oder Arrays von beliebigen Objekten bilden ist es oft nötig festzustellen, von welchem Objekttyp eine konkrete Instanz ist.

Java stellt dafür den `instanceof` Operator zur Verfügung.

```
public class Manager extends Employee
public class Contractor extends Employee
```

Syntax

```
<objekt> instanceof <klasse>
```

Semantik

Falls das Objekt `<objekt>` eine Instanz der Klasse `<klasse>` ist, liefert der Ausdruck den Wert `true`, sonst `false`.
Typischerweise wird der obige Ausdruck in Abfragen oder Verzweigungsanweisungen eingesetzt.

Damit können wir mit generischen Parametern arbeiten (der Parameter einer Methode kann dann einfach vom Oberklassentyp sein, also `Employee` oder sogar `Object`). In der Methode können wir dann immer noch feststellen, von welchem Typ der Parameter ist:

```
public void checkStatus (Employee e) {
    if (e instanceof Manager) {
        // ...
    } else
    if (e instanceof Contractor) {
        // ...
    } else {
        // ...
    }
}
```

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.7.4.2. Casting von Objekten - Objektumwandlung

Unter Umständen möchten Sie Zugriff auf die volle Funktionalität eines Objekts gewinnen, welches generisch an eine Methode übergeben wurde.

Sie stellen beispielsweise in einer Methode fest, dass Sie ein Managerobjekt vor sich haben. Nun müssen Sie, damit Sie die volle Funktionalität des Managers nutzen können, das Objekt, welches als Parameter benutzt wurde, in eine Instanz der Klasse Manager umwandeln. **Erst dann können Sie alle Methoden des Managers nutzen.**

Klassendefinitionen

```
public class Employee {
    private String name;
    private Date hireDate;
    private Date dateOfBirth;
    private String jobTitle;
    ...
}

public class Manager extends Employee
{
    private String department;
    private Employee [] subordinates;
    ...
}
```

Programmfragment

```
1 public void method(Employee e) {
2     if (e instanceof Manager) {
3         Manager m = (Manager)e; // wie beim normalen Casting: (Zieltyp)
4         System.out.println("Objektumwandlung");
5     }
6     // ...
7 }
```

Natürlich können Sie nicht x-beliebige Objekte ineinander umwandeln. Die Regeln für das Casting von Objekten lauten:

- Casts "up" in der Klassenhierarchie sind immer legal. Sie brauchen kein explizites Casting, eine Zuweisung reicht.
- bei "downward" Casts prüft der Compiler, ob die Umwandlung überhaupt möglich ist. Im obigen Beispiel ist die Umwandlung eines Managers zu einem Contractor nicht möglich, da zusätzliche Funktionalität zur Verfügung gestellt wird.
- falls Casting nicht möglich ist, wird eine Ausnahme ausgelöst.

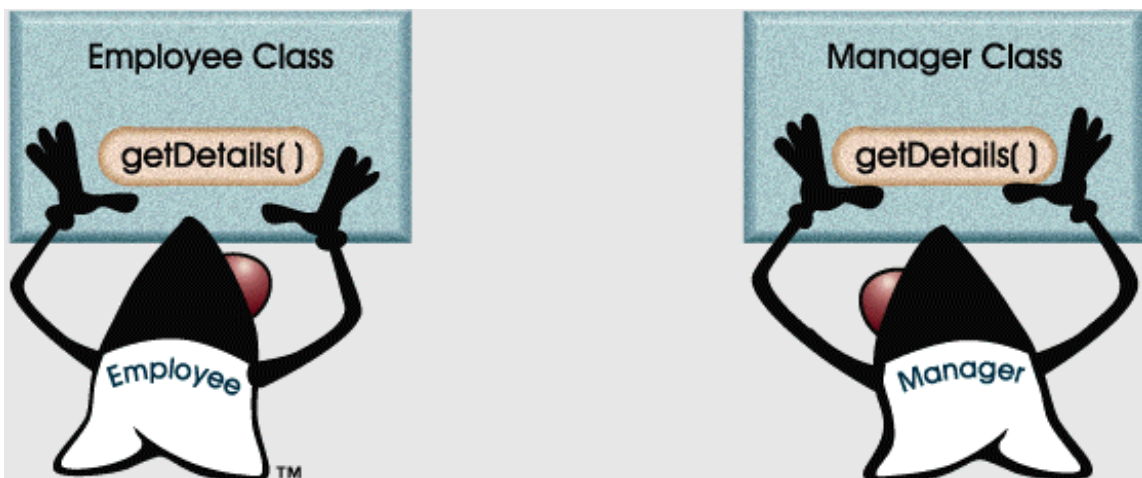
JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.7.4.3. Überschreiben von Methoden

Falls Sie in einer Unterklasse eine Methode mit dem selben Namen wie in einer Oberklasse definieren, aber eventuell mit angepasster Funktionalität, dann sagt man, dass die Methode der Oberklasse in der Unterklasse überschrieben wird.

Das folgende Beispiel zeigt eine Methode, welche die Details zu den Objekten einer Klasse liefert. Die Details der Klasse und der Unterklasse sind unterschiedlich. Es macht also sehr viel Sinn, die Methode in der Unterklasse neu zu definieren, zu *überschreiben*.

Verwechseln Sie zwei Begriffe nicht: *überladen* beschreibt den Vorgang, bei dem Sie eine Methode mit unterschiedlichen Parametern definieren (also unterschiedliche Definitionen der Methode liefern, wie zum Beispiel `print(int...)`, `print(float...)`). *Überschreiben* ist etwas völlig anderes. Sie können nur in einer Klassenhierarchie Methoden überschreiben, überladen können Sie immer.



```
1 public class Employee {
2     String name;
3     int salary;
4
5     public String getDetails() {
6         return "Name: " + name + "\n" +
7             "Gehalt: " + salary;
8     }
9 }
10
11 public class Manager extends Employee {
12     String department;
13
14     public String getDetails() {
15         return "Name: " + name + "\n" +
16             "Leiter von " + department; //für emp nicht definiert
17     }
18 }
```

Falls Sie zwei Instanzen bilden:

```
Employee e = new Employee();
Manager m = new Manager();
```

dann führt der Methodenaufruf zur Ausgabe der Objektdetails in den beiden Fällen zu unterschiedlichen Ausgaben:

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

```
e.getDetails() und  
m.getDetails()
```

zeigen unterschiedliches Verhalten

Weniger offensichtlich ist, was im Fall von

```
Employee e = new Manager();  
e.getDetails();
```

geschieht.

In diesem Fall wird die Methode der Managerklasse aufgerufen. Der Konstruktor zeigt, dass es sich eigentlich um eine Instanz der Managerklasse handelt, man spricht auch von einem virtuellen Methodenaufruf, *virtual method invocation*.

Selbsttestaufgabe

Überprüfen Sie die obige Aussage mit Hilfe Ihrer Programme aus der vorherigen Selbsttestaufgabe.

Falls Sie diese nicht gelöst haben/lösen konnten:

bilden Sie eine Klasse Angestellter mit mindestens einer Methode

bilden Sie eine Unterklasse Manager (erweitert Angestellter)

überschreiben Sie die Methode der Klasse Angestellter

Schreiben Sie ein Testprogramm für Ihre Klassenhierarchie:

bilden Sie eine Instanz der Klasse Angestellter

rufen Sie die Methode auf (sie soll Details über den Angestellten ausgeben)

bilden Sie eine Instanz der Klasse Manager

rufen Sie die Methode auf

bilden Sie jetzt eine Instanz der Klasse Angestellter aber vom Typ

Manager: Angestellter heinz = new Manager();

rufen Sie jetzt die Methode auf mit heinz als Objekt.

Welche Methode wird aufgerufen?

Sie finden eine Lösung dieser Aufgabe auf dem Web / Server unter Programme. Versuchen Sie's aber selbst.

1.7.5. Konstruktion und Destruktion von Objekten

Wir haben bereits gesehen, dass Objekte mit einem Konstruktor kreiert werden.

Syntax

```
Klasse objekt = new Konstruktor(...);
```

Semantik

im Einzelnen geschieht die Konstruktion in mehreren Schritten.

- als erstes wird der Speicherplatz reserviert und alle Felder werden mit Standardwerten initialisiert.
Dieser Schritt geschieht atomar, wird also entweder ganz oder überhaupt nicht ausgeführt. Damit ist sichergestellt, dass die Initialisierung nicht mit zufälligen Werten geschieht.
- als zweites werden die Werte echt initialisiert gemäss speziellen Vorgaben
- als drittes wird der Konstruktor ausgeführt

Selbsttestaufgabe

Verwenden Sie den Debugger im JBuilder, um in die `java` und `javax` Klassen hineinzusteppen.

Sie müssen dafür eine kleine Änderung beim JBuilder vornehmen. Die Anleitung bezieht sich auf JBuilder 4.

Im JBuilder Menü können Sie unter "Run" ein-und ausschalten, in welche Klassen und Packages hineingezoomed werden soll.

"View Classes with Tracing Disabled" heisst die Option, mit der Sie mehr oder weniger Tracing einschalten können.

Suchen Sie `java` und `javax` und klicken Sie darauf. Die Farbe ändert sich ganz leicht. Jetzt können Sie im Debugger (mit F7) in jede der Java Klassen hineinzoomen.

Insbesondere erkennen Sie mit dem Debugger, dass bei der Konstruktion eines neuen Objekts auch ein Objekt zur Klasse `Object`, der Wurzel aller Klassen, angelegt wird, also Objekte schrittweise angelegt werden. Falls Sie in einzelne Anweisungen nicht hineinzoomen möchten, verwenden Sie einfach F8.

Was sehen Sie noch bei dieser Art zu Debuggen?

Sicher sehen Sie, dass der **Security Manager** x-fach aufgerufen wird, um sicherstellen zu können, dass keine unberechtigte Aktionen stattfinden.

Soweit so gut. Aber was passiert beim Vernichten der Objekte?

Wünschenswert wäre beispielsweise eine Methode, welche beim "Vernichten" des Objekts ausgeführt wird. Damit könnte man beispielsweise sicherstellen, dass Dateien, Dateiobjekte, geschlossen werden, bevor sie aus dem System geschmissen werden.

Java bietet für solche Aufgaben den **Finalizer**, die Methode `finalize()` zu einer Klasse. Analoge Konstrukte kennt man auch in C/C++. Im Gegensatz zu C/C++ bei dem der Finalizer wichtig ist, um Speicher frei zu geben, wird bei Java diese Arbeit durch den Garbage Collector erledigt. Der Finalizer ist daher nicht zwingend nötig.

Sie finden ein Beispiel auf dem Web / Server. Wir werden später auf sinnvollere Einsätze des Finalizers eingehen.

1.7.6. Packages

Java bietet mit dem *Package* Konstrukt die Möglichkeit, zusammengehörende Klassen zu gruppieren. Falls Sie die Package Anweisung, die **ganz am Anfang**, vor allen anderen Anweisungen stehen muss, nicht benutzen, wird ein Standard Package, das sogenannte `unnamed package` als Ersatz genommen.

Syntax

```
package <name>;
```

Semantik

die nach der Package Anweisung folgende Definition einer oder mehrerer Klassen gehört / gehören zu einem Package mit dem Namen `<name>`.

`<name>` sollte dabei falls das Package für den internationalen bzw. öffentlichen Einsatz entwickelt wird, mit dem umgekehrten Domain Namen beginnen. Falls wir für die Firma HergiswilSoft mit Domain Name `hergiswilsoft.ch` ein öffentlich nutzbares Package entwickeln, um beispielsweise das FTP Protokoll zu nutzen, könnte der Package Name lauten:
`ch.hergiswilsoft.ftp`

Beispiel

```
package ch.fhz.hta.inf.keyboard

public class keyboardInput {
    ...
}
```

Sie können nicht mehrere Package Anweisungen gleichzeitig angeben.

Package Angaben können lediglich Kommentare vor sich haben, also kein `import`, keine Klassendefinitionen oder ähnliches.

Package Namen werden in der Regel hierarchisch aufgebaut. In der Regel besteht der Package Name ausschliesslich aus **Kleinbuchstaben** (im Gegensatz zu Klassennamen).

Für unsere Beispielfirma (HergiswilSoft mit Domain Name `hergiswilsoft.ch`) würde also über die Jahre eine Hierarchie von Packages aufbauen:

Packages der HergiswilSoft:

```
ch.hergiswilsoft
  ch.hergiswilsoft.keyboard
    ch.hergiswilsoft.keyboard.char
ch.hergiswilsoft.ftp
  ch.hergiswilsoft.ftp.protocolhandler
  ch.hergiswilsoft.ftp.contenthandler
...
```

1.7.7. Die `import` Anweisung

In Java haben Sie durch Angabe der Packages in der Import Anweisung die Möglichkeit anzugeben, wo sich die fehlenden Klassen Ihres Programms befinden.

Syntax

```
import <Package oder Klassen-Namen>;  
[import <weiteres Package oder weitere Klasse>;]
```

Semantik

Falls Sie in Ihrem Programm Klassen aus anderen Paketen benötigen, müssen Sie diese mit einer Import Anweisung angeben.

Sie können mehrere Import Anweisungen verwenden, die einfach hinter einander stehen. Verschiedene Packages werden immer geladen, wie `java.lang`, in dem die Sprachdefinition von Java und Basisklassen definiert werden.

Die Import Anweisung muss vor allen Klassendefinitionen sein, jedoch nach einer allfälligen Package Anweisung.

Beispiele

Zur Illustration des oben gesagten hier einige Beispiele.

Dazu definieren wir zwei Packages, die zu `ch.fhz.hta` gehören

```
package ch.fhz.hta.meineklassen;  
public class MeineKlasse {  
    private float f=10f;  
    static String strBsp = "MeineKlasse";  
    public MeineKlasse() {  
    }  
    public void updateFloat(float f) {  
        this.f = f;  
    }  
    public float readFloat() {  
        return f;  
    }  
}
```

Und hier eine Anwendung:

```
import ch.fhz.hta.meineklassen.*;  
class MeineAnwendung {  
    public static void main(String[] args) {  
        float f=3.14f;  
        String strM = "MeineKlasse";  
        MeineKlasse mko = new MeineKlasse();  
        System.out.println("Float aus meiner Klasse:  
                            "+mko.readFloat());  
        System.out.println("Float aus meiner Klasse mutieren");  
        mko.updateFloat(f);  
        System.out.println("neuer Float aus meiner Klasse:  
                            "+mko.readFloat());  
    }  
}
```

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

Die Import Anweisung vereinfacht auch die Notation:

anstelle von `<package_name>.<klassen_name>` können wir einfach `<klassen_name>` verwenden:

```
package ch.fhz.hta.meineanwendung;
// hier könnte auch ein anderes Package stehen

import ch.fhz.hta.meineklassen.*;

class MeineAnwendung {
    public static void main(String[] args) {
        float f=3.14f;
        String strM = "MeineKlasse";
        MeineKlasse mko = new MeineKlasse();
        System.out.println("Float aus meiner Klasse: "+mko.readFloat());
        System.out.println("Float aus meiner Klasse mutieren");
        mko.updateFloat(f);
        System.out.println("neuer Float aus meiner Klasse: "+mko.readFloat());
        // jetzt spielen wir etwas
        ch.fhz.hta.meineklassen.MeineKlasse mkox = new MeineKlasse();
        // mkox ist jetzt ein ganz normales Objekt,
        // eine Instanz der Klasse MeineKlasse
    }
}
```

Selbsttestaufgabe

Schreiben Sie eine einfache Klassenbeschreibung, beispielsweise jene aus der früheren Selbsttestaufgabe zum Überladen von Methoden (Datum Formate) so um, dass Sie sie in einer andern Anwendung importieren und wiederverwenden können. Wählen Sie für Ihre Testklasse, in die Sie Ihre Datumsklasse importieren wollen, bewusst einen andern Namen.

Wie unterstützt Sie JBuilder beim Importieren? JBuilder müsste Ihnen die verfügbaren Packages anzeigen und Sie hineinzoomen lassen.

1.7.8. Packages und Verzeichnisse

Packages werden in Verzeichnissen abgespeichert. Der Package Name wird dabei zum Verzeichnisnamen.

Die Beispiele aus dem vorherigen Abschnitt wurden beispielsweise im Verzeichnis `ch.fhz.hta.*` abgespeichert.

Damit die Klassen automatisch gefunden werden können, müsste Ihre Umgebungsvariable (`CLASSPATH`) die Angabe `ch.fhz.hta` enthalten.

Wenn Sie eine Classdatei starten möchten, die zu einem Package gehört, müssen Sie auch in diesem Fall die Datei im DOS Fenster vollqualifiziert aufrufen.

Selbsttestaufgabe

Versuchen Sie die Classdatei aus den vorherigen Beispielen direkt in einem DOS Fenster zu starten, mit `java ...`. Beachten Sie, dass die Klassen zu einem Package gehören.

Alternativ, falls Sie damit Schwierigkeiten haben:

Schreiben Sie ein kleines Programm (`MeineAnwendung`), mit `main(...)`, welches zu einem einfachen Package (zum Beispiel `meinpackage`) gehören. Falls sich die Classdatei im Verzeichnis `meinpackage` befinden, können Sie beispielsweise im übergeordneten Verzeichnis die Klasse starten mit

```
java -cp .;... meinpackage.MeineAnwendung
```

Hinweis

Packages sind eine einfache Möglichkeit Ihre Projekte zu strukturieren. Die Wichtigkeit der Packages sollte Sie aber nicht irritieren. In vielen Fällen können Sie vollständig darauf verzichten.

Sie sollten auch beachten, dass Sie durch die Einführung von Packages Ihre Lösung komplizieren können, speziell, wenn Sie den Package Aufbau ungeschickt machen.

1.8. Module 6 : Exceptions

1.8.1. Einleitung

In einem früheren Modul haben wir die grundlegenden Konzepte des Programmablaufs besprochen. Nun müssen wir ein wichtiges Thema besprechen: alternative Ausführungspfade. Was passiert im Fehlerfall? Wie werden Ausnahmestände abgefangen?

1.8.1.1. Lernziele

Nach dem Durcharbeiten dieses Moduls sollten Sie in der Lage sein:



- Exceptions zu definieren.
- erklären können warum Exception Handling wichtig ist.
- Code schreiben können, mit dem Sie Exceptions abfangen können.
- eigene Exceptions definieren und auslösen können.

1.8.1.2. Orientieren Sie sich selbst



Als Vorbereitung für diesen Modul sollten Sie sich folgende Fragen stellen:

- wie könnte man Fehler in einem Programm abfangen?
- welche Fehler könnten wann auftreten?
- macht es überhaupt Sinn alternative Ausführungspfade zu definieren?

Falls Sie diesen Modul durcharbeiten, sollten Sie diese Fragen im Kopf behalten. Sie sollten Ihnen helfen den Text besser zu verstehen.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.8.2. Exceptions - Ausnahmen

Was ist eine Ausnahme?

In Java werden Exceptions als milde Fehler definiert. Anstatt das das Programm abgebrochen wird, kann man eine Ausführungsalternative vorschlagen. Dafür muss man die Bedingungen definieren, unter denen fortgefahren wird, und es muss festgehalten werden, wie fortgefahren werden soll.

Was ist ein Fehler?

Im Diagramm unten sehen Sie, dass die `Throwable` Klasse die Superklasse von `Error` und `Exception` ist, die ihrerseits die Superklassen für alle weiteren systemseitig oder benutzerseitig definierten *Ausnahmen* sind.

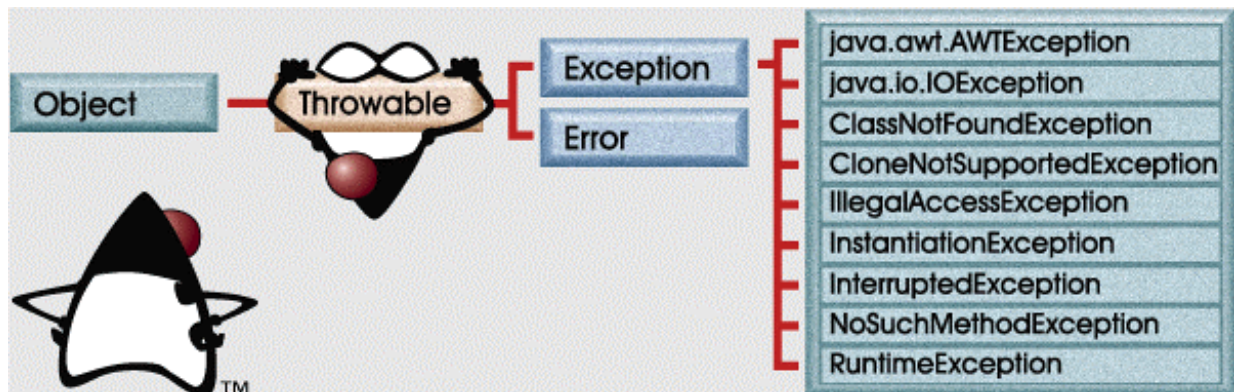
Die `Error` Klasse definiert das, was man als sehr *seriösen* Fehler bezeichnet, also Fehler, die man nicht einfach so beseitigen kann.

Die `Exception` Klasse ist die Superklasse, welche zwei Fehlertypen definiert:

- Runtime Exceptions werden vom Java Laufzeitsystem "geworfen", ausgelöst. Diese gehören zur Klasse `RuntimeException` und sollten nicht abgefangen werden.
- alle anderen Ausnahmen, `Exception`, die abgefangen werden können.

Der Ablauf sieht folgendermassen aus:

falls eine Ausnahme auftritt wird eine Ausnahme geworfen (Ausnahmen sind Objekte). Falls Sie diesen Ausnahmefall abfangen, wird der entsprechende Programmcode ausgeführt.



Achtung: die Reihenfolge der Exceptions ist wichtig.

Falls Exception als erste Ausnahme abgefangen wird, werden die anderen Exceptions nicht mehr ausgewertet!

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.8.2.1. Exception - Beispiele

Betrachten wir ein einfaches Beispiel:

Sie möchten ein Hello World Programm schreiben, bei dem Sie mehrere Begriffe, Zeichenketten eingeben können. Das Programm heiße `HelloWorld` und werde folgendermassen aufgerufen:

```
$ java HelloWorld Hello world! No, I mean it! HELLO WORLD!!
java.lang.ArrayIndexOutOfBoundsException: 3 at
HelloWorld.main(HelloWorld.java:12)
```

So ein Pech! Das ging aber gründlich schief. Warum?

Schauen wir uns das Programm genau an. Zuerst eine Version, die eine Fehlermeldung liefert:

```
1 public class HelloWorld {
2     public static void main (String args[]) {
3         int i = 0;
4         String greetings [] = {
5             "Hello world!",
6             "No, I mean it!",
7             "HELLO WORLD!!"
8         };
9         while (i < 4) { // Fehler: greetings[.] hat nur 0,1,2
10            System.out.println (greetings[i]);
11            i++;
12        }
13    }
14 }
```

Nun möchten wir das Programm so abändern, dass die Sprüche als Parameter eingegeben werden können. Unser Programm lässt sich leicht anpassen:

```
1 public class HelloWorld {
2     public static void main (String args[]) {
3         int i = 0;
4         /*String greetings [] = {
5             "Hello world!",
6             "No, I mean it!",
7             "HELLO WORLD!!"
8         }; */
9         while (i < 4)
10            System.out.print(args[i]+" ");
11            i++;
12        }
13    }
14 }
```

Die Angabe einer fixen Anzahl Parameter ist ungeschickt, weil Sie eventuell unterschiedlich viele Zeichenketten eingeben möchten.

Das Programm lässt sich jedoch leicht so formulieren, dass auch dieser Fall abgefangen wird.

Das folgende Programm zeigt eine korrekte und eine falsche Version:

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

```
public class EinfuehrendesBeispiel {
    public static void main(String[] args) {
        // korrekte Version
        for (int i=0; i< args.length; i++) // okay
            System.out.print(" "+args[i]);
        System.out.println();
        // fehlerhafte Version
        int i=0;
        while (i<4) {                               // Schrott
            System.out.print(args[i]+" ");
            i++;
        }
    }
}
```

```
Hallo Du da
Hallo Du da java.lang.ArrayIndexOutOfBoundsException
at
exception.EinfuehrendesBeispiel.main(EinfuehrendesBeispiel.java:21)
Exception in thread "main"
```

Im zweiten Fall tritt eine Exception auf, weil args[] nun nur noch aus drei Elementen besteht, somit die Schleife zu weit zählt.

In diesem Fall verlangt Java nicht schon zum voraus, dass Sie eine Exception definieren. In anderen Fällen, wie beispielsweise der Eingabe und Ausgabe, wird dies bereits vom Compiler überprüft.

1.8.2.2. Exception Handling - try und catch Anweisung

Java führt zur Behandlung solcher Softerrors das try...catch...finally Konzept ein.

Im Programm werden die kritischen Bereiche mit try...catch gesichert. Es wird versucht die Anweisungen im try Block auszuführen.

Falls keine Ausnahme auftritt, wird der try Block ausgeführt und anschliessend nach dem catch Block fortgefahren.

Falls während des try Blocks eine Ausnahme geworfen wird, prüft das Programm, ob ein zur Ausnahme passender catch Block definiert ist. Falls ja, dann wird dieser Block ausgeführt. Falls nicht, wird abgebrochen.

Syntax

```
try {
    // Code, in dem eine Ausnahme auftreten kann
} catch (BestimmteAusnahme e) { ....}
// Code der ausgeführt werden kann,
catch(WeitereAusnahme we) {...}
}
```

Semantik

Sie müssen angeben, welche Ausnahme Sie abfangen möchten. Falls Sie mehrere Ausnahmen abfangen möchten, müssen Sie einfach mehrere catch Blöcke spezifizieren.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

Beispiel

Wir schauen uns das obige Beispiel an und möchten die geworfene Exception abfangen. Um welche Exception es sich handelt, sehen wir aus der Fehlermeldung. In der Java Dokumentation (API) finden Sie jeweils die Ausnahmen aufgelistet, die eine Anweisung werfen kann.

```
int i=0;
  try {
    while (i<4) {
      System.out.print(args[i]+" ");
      i++;
    }
  } catch (ArrayIndexOutOfBoundsException ie) {
    System.out.println("Sie haben versucht, args["+i+"] auszugeben.
Es gibt aber nur "+args.length+" Argumente");
  }
}
```

mit der Ausgabe

```
...
Sie haben versucht, args[3] auszugeben. Es gibt aber nur 3 Argumente
...
```

1.8.2.3. Die *finally* Anweisung

Wir haben nun also eine etwas verzwickte Situation: ein Teil des `try` Blockes wird ausgeführt, einer oder mehrere `catch` Blöcke werden ausgeführt. Aber eventuell möchten Sie, dass ein bestimmter Block **immer** ausgeführt wird. Diesen stecken Sie in den `finally` Block. Dieser Block wird *immer* ausgeführt

Falls Sie im `try` oder im `catch` Block die `System.exit()` Anweisung ausführen, wird das System allerdings sofort verlassen. Man bezeichnet den Programmcode im `try` Block auch als *protected code*.

```
try {
  while (i<4) {
    System.out.print(args[i]+" ");
    i++;
  }
} catch (ArrayIndexOutOfBoundsException ie) {
  System.out.println();
  System.out.println("Sie haben versucht, args["+i+"] auszugeben.
  Es gibt aber nur "+args.length+" Argumente");
} finally {
  System.out.println("nun naht das Ende...");
}
```

```
Sie haben versucht, args[3] auszugeben. Es gibt aber nur 3 Argumente
nun naht das Ende...
```

Hinweis

Exceptions müssen Sie immer abfangen. Die `finally` Anweisung setzt man allerdings sehr selten ein.

1.8.2.4. Einige typische Exceptions

Java definiert einige Exceptions. Hier eine Liste der am meisten gebrauchten. Wie Sie eigene Exceptions definieren können, sehen Sie weiter unten.

- `ArithmeticException` - Beispielsweise eine Division durch 0

```
int i = 12 / 0;
```
- `NullPointerException` - Zugriff auf ein Objekt oder eine Methode, bevor das Objekt instanziiert wurde:

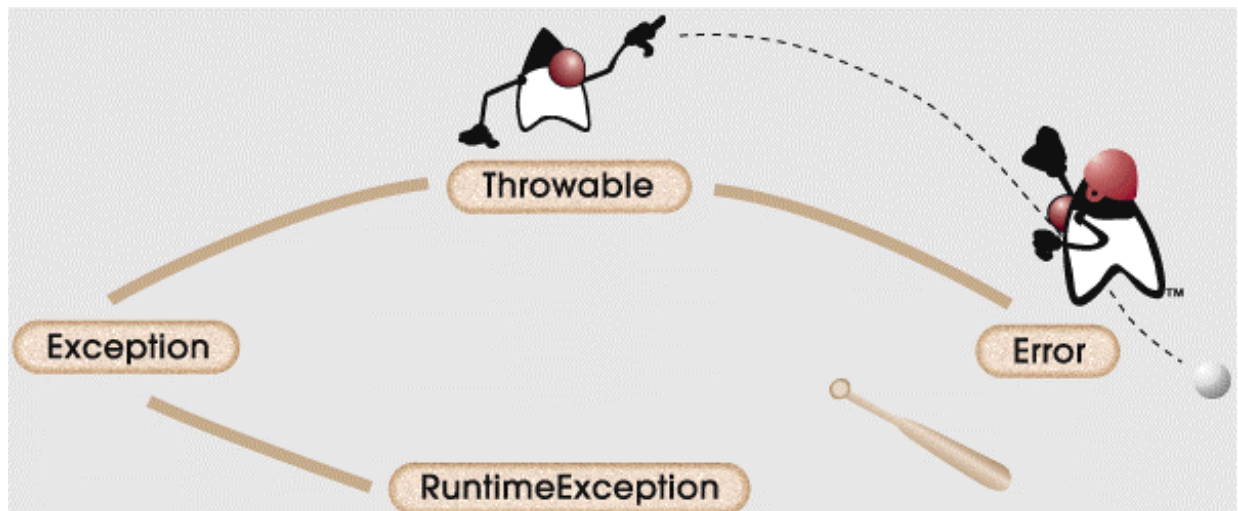
```
Image im [] = new Image [4];  
System.out.println(im[0].toString());
```
- `NegativeArraySizeException` - Versuch ein Array mit negativer Anzahl Elemente zu kreieren.
- `ArrayIndexOutOfBoundsException` - Diese Exception kennen Sie aus dem Beispiel: Sie versuchen auf Elemente eines Arrays zuzugreifen, die es nicht gibt..
- `SecurityException` - In einem Browser kann der `SecurityManager` diese Ausnahme werfen, falls Sie versuchen:
 - auf lokale Dateien zuzugreifen
 - falls Sie auf andere Rechner zugreifen möchten, andere als jener, von dem das Applet stammt.
 - Programme auszuführen, für die Sie keine Berechtigung haben.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.8.2.5. Unterschiedliche Exceptions

In Java werden die Exceptions als Unterklasse von `java.lang.Throwable` definiert. Java kennt vereinfacht gesagt, drei Kategorien von Exception.

- `Error` - dabei handelt es sich um schwere Fehler, die in der Regel nicht behoben werden können. Ein Beispiel wäre der Laufzeitfehler, der eintritt, wenn das System keinen Speicher mehr hat. Da können Sie auch nicht mehr viel machen.
- `RuntimeException` - diese deuten auf Design oder Implementationsprobleme. Ein typisches Beispiel ist der Fehler mit den Array Index Grenzen.
- andere Laufzeitausnahmen - Ausnahmen die zur Laufzeit auftreten können und die auch leicht behoben werden können. Beispielsweise kann der Zugriff auf eine Datei zur Zeit nicht möglich sein. Oder Sie haben einen falschen Dateinamen angegeben. Solche Fehler können Sie abfangen und gegebenenfalls korrigieren.



1.8.2.6. Behandeln von Ausnahmen

Sie können als Programmierer den Ablauf kontrollieren, indem Sie auf der einen Seite die `try... catch` Anweisung benutzen. Auf der andern Seite können Sie auch angeben, welche Ausnahme in einem bestimmten Fall geworfen werden kann. Hier eine Methoden Definition

```
public void versucheZuLesen() throws IOException
```

Hier tritt ein weiteres Schlüsselwort auf: `throws`.

Syntax

<Methode> throws <Exceptionliste>

Semantik

Sie können eine oder mehrere Ausnahmen angeben, die geworfen werden können.

Beispiel

wir werden einige Beispiele konstruieren, sobald wir eigene Exceptions definieren können.

1.8.2.7. Definieren eigener Ausnahmen

Sie können eigene Exception auf die billige Art und Weise implementieren. Dies geschieht so, dass Sie einfach die Klasse `Throwable` erweitern. Die `Throwable` Klasse enthält eine Zeichenkette, mit der Sie beispielsweise angeben könnten, welcher Fehler aufgetreten ist.

Dafür wird jedoch `Exception`, eine Unterklasse von `Throwable` erweitert. Falls Sie mehr Daten manipulieren möchten, können Sie dies durch eine eigene Klassendefinition, welche `Exception` erweitert, jederzeit tun.

Sie sollten aber unterscheiden zwischen der Beschreibung der (eigenen) `Exception` und der Behandlung der `Exception` im `catch` Block.

Um eine `Exception` zu werfen können Sie beispielsweise folgende Anweisung verwenden.

Syntax

```
throw new <meine_Exception>;
```

Semantik

`throw` zeigt an, dass ein Ausnahmezustand auftritt.
Mit `new` wird ein Objekt, eine Instanz Ihrer `Exception` kreiert.

Definition einer `Exception`:

```
public class DefinitionDerException extends Exception{  
  
    public DefinitionDerException(String hinweis) {  
        // wir verwenden einfach den Konstruktor der Exception Klasse  
        // super(...) konstruiert ein Objekt der Oberklasse, Exception  
        super(hinweis);  
    }  
}
```

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.8.2.8. Beispiel

Der Einfachheit halber reproduzieren wir das obige Beispiel, inklusive dem Testprogramm:

```
package exceptions;

public class DefinitionDerException extends Exception{

    public DefinitionDerException(String hinweis) {
        // wir verwenden einfach den Konstruktor der Exception Klasse
        // super(...) konstruiert ein Objekt der Oberklasse, Exception
        super(hinweis);
    }
}

package exceptions;

public class ExceptionTester {
    public static void main(String[] args) {
        System.out.println("Test der Exception 'DefinitionDerException'");
        try {
            System.out.println("im try Block");
            throw new DefinitionDerException("Meine Güte - das ging aber
schief");
        } catch(DefinitionDerException ex) {
            System.out.println("im catch Block");
            // Version 2: etwas mehr Ausgabe
            System.out.println("localizedMessage "+ex.getLocalizedMessage());
            System.out.println("Message "+ex.getMessage());
            ex.printStackTrace();

        }
    }
}
```

In der `catch` Anweisung wird die Exception, die abgefangen wird, deklariert (die `catch` Anweisung besitzt einen Parameter, den man deklariert. Er ist vom Typ `Exception` oder einer Unterklasse; in unserem Fall ist er vom Typ `DefinitionEinerException` und die Referenz darauf heisst `ex`).

Das `super` Schlüsselwort werden wir noch genauer kennen lernen. Es ist aber sehr anschaulich:

- mit `super.<Datenfeld>` wird auf ein Datenfeld der Oberklasse zugegriffen;
- mit `super.<Methode>` wird eine Methode der Oberklasse aufgerufen;
- mit `super ()` wird der Konstruktor der Oberklasse aufgerufen.

An Stelle des einfachen Konstruktors könnten wir jederzeit einen weiteren Block einsetzen, in dem dann noch verschiedene Anweisungen ausgeführt werden.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

Selbsttestaufgabe

Schreiben Sie eine Klasse, die ein Bankkonto implementiert.

Das Bankkonto hat die privaten Datenfelder:

den Kontostand, (double)

die Kontonummer, (long)

den Inhaber (String)

Zusätzlich werden Zugriffsmethoden definiert:

```
public double abfragenKontostand();  
public void einzahlen(double betrag);  
public void abheben(double betrag) throws Ungedeckt;
```

Ihre Aufgabe:

definieren Sie eine Klasse Bankkonto mit obigen Methoden und Datenfeldern, sowie die Exception `Ungedeckt`, welche beim Versuch Geld abzuheben geworfen wird, falls mehr Geld abgehoben werden soll als auf dem Konto vorhanden ist.

Testen Sie Ihre Klassendefinition und die Exception mit einem einfachen Testprogramm.

1.9. Modul 7 - Einführung in Java Applets

In dieser Kurseinheit besprechen wir im Sinne einer Einführung, was Applets sind und wie man Applets programmiert. Einige Grundlagen fehlen noch. Sie werden grafische Elemente einsetzen, obschon wir die grafischen Elemente erst in einer späteren Einheit besprechen.

1.9.1. Einführung

Um Applets einsetzen zu können, benötigt man einen Web Browser. Der Browser enthält eine Virtual Machine, die in der Lage ist, den Bytecode des Applets zu interpretieren.

1.9.1.1. Lernziele

Nach dem Durcharbeiten diese Moduls sollten Sie in der Lage sein:

- den Unterschied zwischen einer Java Applikation und einem Java Applet erklären zu können.
- die HTML Tags schreiben zu können, mit denen ein Applet gestartet wird
- die Klassenhierarchie eines Applets und des AWTs zu skizzieren
- ein HelloWorld Applet zu schreiben
- die wichtigsten Applet Methoden zu beschreiben
- das Painting Modell des AWT zu erklären
- mit Applets Bilder und Sound zu lesen und auszugeben
- den <param> Tag so zu konfigurieren, dass die Parameter von einem Applet interpretiert werden kann.

1.9.2. Applets verstehen

1.9.2.1. Ein Applet ist ...

Ein Applet ist eine Java Klasse, welche in einer Browser Umgebung läuft. Die Java Klasse unterscheidet sich von einer Applikation durch die Art, wie sie ausgeführt wird.

Eine Applikation wird durch die main() Methode gestartet. Im Gegensatz dazu ist der Lebenszyklus komplexer. Im Folgenden werden Sie sehen, inwiefern diese Unterschiede wichtig oder weniger wichtig sind, und wie Sie Applet Programme schreiben können.

Da das Applet im Browser läuft, kann man es nicht direkt starten. Sie müssen im Browser eine HTML Seite laden, in der das Applet angegeben wird, inklusive "Codebase" und Parametern. Das folgende Beispiel zeigt eine HTML Seite mit drei Applets (TitleApplet, TextApplet, and CopyrightApplet) mit der die übliche Copyright Information von Sun angezeigt werden kann, sofern Sie diese auf Ihrem Rechner haben.

```
<html>
<head>
  <title>An Applet Is ...</title>
</head>
<body BGCOLOR="#D3D9E4">
<center>
<applet code=sun.suntutor.applets.TitleApplet width=600 height=40>
  <param name=text value="An Applet Is ...">
</applet><br>
<applet code=sun.suntutor.applets.TextApplet width=600 height=195>
  <param name=TextURL value="top.html">
```

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

```
</applet>
<p>
<applet code=sun.suntutor.applets.TextApplet width=600 height=195>
  <param name=TextURL value="bottom.html">
</applet>
<br>
<applet code=sun.suntutor.applets.CopyrightApplet width=600 height=18>
</applet>
</center>

</body>
</html>
```

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.9.3. Applet Sicherheitseinschränkungen

Applets werden in der Regel über das Internet geladen. Das klingt gefährlich. Kann da jeder Hacker sein Lieblingsapplet über das Internet an unbedachte Benutzer verteilen, Passwörter lesen und sensible Daten lesen?

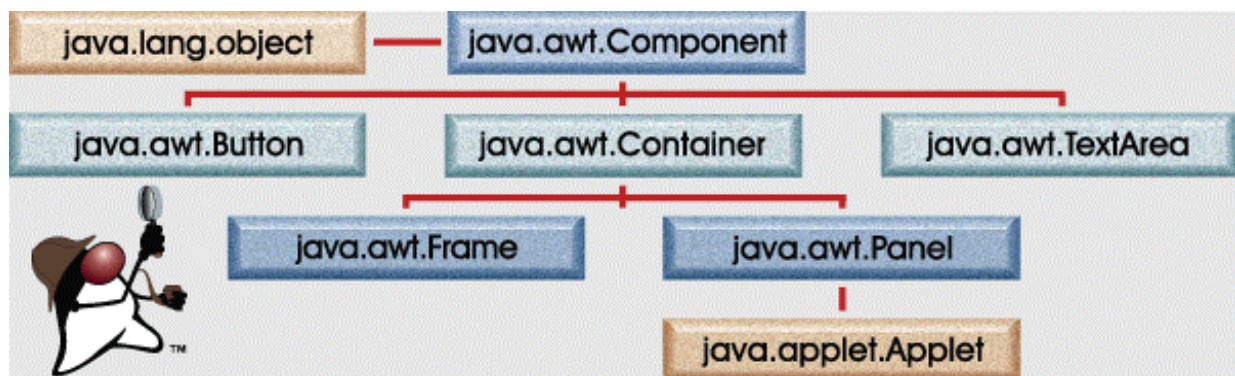
Java regelt die Zugriffsrechte mit Hilfe des SecurityManagers, einer Java Klasse, mit deren Hilfe die Zugriffsrechte für fast alle Zugriffe aus der JVM geregelt und überprüft werden. Dieses Modell bezeichnet man als "Sandbox Modell". Wir werden dieses Modell in der Einheit "Sicherheit" speziell anschauen. Einfach ausgedrückt funktioniert das Sandbox Modell so, dass ein Applet nur auf seine Codebase zugreifen kann, also auf die URL, auf den Host und das Verzeichnis, von dem es stammt. Es kann also nicht auf lokale Daten zugreifen, falls es nicht lokal geladen wurde.

Insbesondere werden folgende Aktivitäten verboten:

- Ausführen von andern Programmen, beispielsweise starten eines Servers
- Datei Eingabe und Ausgabe
- Aufruf von Systemroutinen
- Verbindungsaufbau über Sockets mit andern als dem Codebase Servern

1.9.4. Die Applet Klassenhierarchie

Bevor wir ein erstes Applet anschauen, betrachten wir die Klassenhierarchie, in die die Applet Klasse eingebettet ist. AWT steht für Abstract Windowing Toolkit und ist der Versuch / war der Versuch, ein plattformunabhängiges Windowing System auf die Beine zu stellen.



AWT ist ein Package, welches Klassen enthält, mit deren Hilfe (einfache und komplexere) Oberflächen entwickelt werden können.

Die einzelnen Elemente der Benutzeroberfläche, wie beispielsweise ein Knopf oder ein Rollbalken, werden als Komponenten, *component*, bezeichnet. AWT enthält eine ganze Reihe Standardkomponenten, welche auf allen Plattformen verfügbar sind.

Eine *Component* generiert Ereignisse, *events*, wenn der Benutzer mit Ihnen wechselwirkt. Ein *Container* ist eine Komponente, welche Komponenten und andere Container enthält. Ein Container kann auch einen **Layout Manager** enthalten, mit dessen Hilfe die visuelle Platzierung der Komponenten im Container geregelt wird.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

Der Layout Manager legt fest, wie die Komponenten angeordnet werden. Es gibt verschiedene Standard Layout Manager, wie beispielsweise das Anordnen in "Nord-Süd-Ost-West-Zentrum" und viele weitere Möglichkeiten. Sie können auch eigene Layout Manager definieren.

Im obigen Diagramm sehen Sie, dass `java.applet.Applet` eine Unterklasse von `java.awt.Container` ist. Sie können also mit einem Applet gleich AWT Layouts benutzen und sofort grafische Oberflächen bauen. Der Standardlayout für Applets ist der sogenannte `FlowLayout`, den wir gleich kennen lernen.

1.9.5. Ein "HelloWorld" Applet

Ein Applet startet normalerweise folgendermassen:

```
import java.applet.*;
public class HelloWorld extends Applet {
```

Die Appletklasse muss `public` sein. Damit muss, auf Grund der Spezifikation der JVM, die Datei gleich heissen, wie das Applet. In unserem Fall: `HelloWorld.java`.

Die Klasse muss zudem eine Unterklasse von `java.applet.Applet` sein.

Ein Hinweis:

seien Sie vorsichtig mit Packages bei Applets. Das kann kompliziert werden! Sie sehen oben, dass kein Package Namen angegeben wurde. Das war Absicht!

```
1 // HelloWorld Applet
2 //
3 import java.awt.Graphics;
4 import java.applet.*;
5
6 public class HelloWorld extends Applet {
7     String hw_text;
8     public void init () {
9         hw_text = "Hello World";
10    }
11
12    public void paint(Graphics g){
13        g.drawString(hw_text, 25, 25);
14    }
15 }
```

Selbsttestaufgabe

Schreiben Sie ein HelloWorld Applet und versuchen Sie es zu testen. Im JBuilder wird automatisch eine HTML Seite generiert, mit der das Applet gestartet werden kann.

Beachten Sie den obigen Hinweise in Bezug auf Packages und löschen Sie die Package Angabe beim Definieren des Applets.

Applets können in JBuilder direkt generiert werden. JBuilder übernimmt fast alles für Sie! Leider zuviel...

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

Der Wizzard fragt Sie auch, ob Sie das Applet als standalone Applikation starten möchten. Falls Sie diese Option anklicken, wird auch noch eine `main` Methode generiert. Sie haben dann ein Applet und eine Applikation in einem.

Parameter geben wir keine an. Der Wizzard bietet Ihnen die Möglichkeit auch dazu Angaben zu machen. Im HelloWorld Beispiel benötigen wir diese Option nicht.

Die HTML Details können Sie unverändert übernehmen.

Den generierten Code können Sie eigentlich vollständig löschen und durch den einfacheren Code oben ersetzen.

Sie finden das Beispiel auch auf dem Server

1.9.6. Wichtige Applet Methoden

Im vorigen Beispiel verwendeten wir keine `main` Methode. Applets werden mit Hilfe der `init` Methode gestartet. Diese entspricht der `main` Methode für Applikationen.

Nach der Initialisierung würde der Konstruktor ausgeführt. Konstruktoren werden bei Applets sehr selten eingesetzt. Nach der Initialisierung mit `init()` wird die `start()` Methode ausgeführt.

`start()` werden wir noch genauer besprechen. Mit `start()` wird das Applet zum Leben erweckt.

1.9.7. Der Applet Lebenszyklus

Im Leben des Applets gibt es drei wesentliche Methoden. Dies sind `init()`, `start()` und `stop()`.

- `init()` - Diese Member Funktion wird aufgerufen, wenn das Applet kreiert und das erste Mal geladen wird (in den Browser, in den Appletviewer). Das Applet kann mit Hilfe dieser Methode Datenwerte initialisieren. Aber Sie müssen beachten: diese Methode wird nur beim ersten Laden des Applets ausgeführt; bei jedem weiteren Laden der gleichen Seite wird diese Methode nicht mehr ausgeführt.
- `start()` - Damit wird das Applet aufgefordert aktiv zu werden, zu "leben". Gestartet wird erst, nachdem das Applet initialisiert wurde. Aber gestartet wird auch, falls die Seite neu geladen oder die Grösse des Browser Fensters geändert wird. Es handelt sich also um die eigentliche Starter Routine.

```
public void start() {  
    musicClip.play();  
}
```

- `stop()` - Mit dieser Methode wird das Leben des Applets beendet. Gründe können sein: das Browser Fenster wurde zum Ikon verkleinert oder eine neue Seite wurde geladen. Das Applet kann damit auch Animationen stoppen oder das Laden von Sound.

```
public void stop() {  
    musicClip.stop();  
}
```

Die `start()` und `stop()` Methoden gehören wie Zwillinge zusammen.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.9.8. Applet Anzeige

Applets sind von Grund auf grafische Gebilde. Sie werden in einem Applet kaum `System.out` finden. Für die Anzeige wird alles in grafische Objekte umgewandelt und angezeigt.

Die grafischen Elemente werden als grafische Objekte an die `paint()` Methode übergeben.

```
public void paint(Graphics g){
    g.drawString(hw_text, 25, 25);
}
```

Die `paint()` Methode wird nicht von Ihnen aufgerufen sondern vom Browser! Wenn immer sich die Browser Umgebung ändert, muss der Bildschirm aufgefrischt werden. Dies geschieht beispielsweise wenn Sie die Grösse des Browser Fensters ändern, minimieren oder verschieben.

Es ist Ihre Aufgabe dafür zu sorgen, dass in der `paint()` Methode alles drin steht, so dass der Refresh auch korrekt abläuft. Dieser Ablauf hängt damit zusammen, dass die Darstellung von der Umgebung, nicht vom Programm abhängt.

1.9.9. Die `paint()` Methode

Das Argument der `paint()` Methode ist eine Instanz von `java.awt.Graphics`. Das Argument der `paint()` Methode ist immer das Kontext Panel, aus dem das Applet besteht.

Das einfachste mögliche Applet zeigt einen Text als Zeichnung an. Die `paint()` Methode wird benutzt, um den Text zu schreiben. Sie haben bereits das entsprechende Applet kennen gelernt.



Die numerischen Werte in der `drawString` Methode sind `x` und `y` Pixel Koordinaten für den Start des Textes. Die Koordinaten beziehen sich auf die Basislinie des Textes. Falls Sie an Position 0 schreiben, wird ein Teil des Buchstaben abgeschnitten.

Selbsttestaufgabe

Bestimmen Sie eine sinnvolle Startposition, beispielsweise mit Hilfe des Zeichens 'y' als Ausgabe.

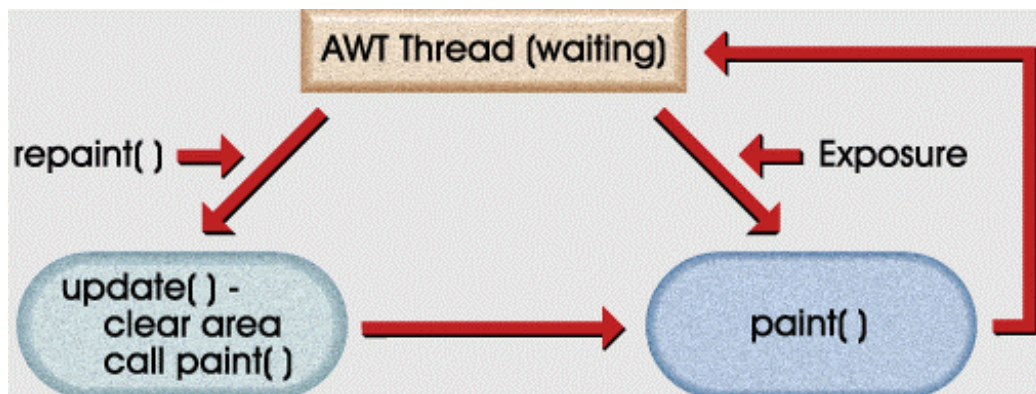
JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.9.10. AWT Painting

Neben den Lebenszyklus Methoden des Applets müssen Sie auch genau wissen, wie ein Applet die AWT Komponenten nutzt. Viele Aufgaben werden durch die AWT Komponenten automatisch erledigt.

Das Updaten des Bildschirms geschieht mit einem eigenen Thread, dem AWT Thread, also mit einem Teilprozess, der nichts anderes als dies zu tun hat. Der Teilprozess wird automatisch aufgerufen, falls:

1. ein Teil der Anzeige aktualisiert werden muss, weil beispielsweise ein anderes Fenster darüber gelegt worden war und damit das grafische Objekt "beschädigt" wurde.
2. das Programm selbst entscheidet, dass die grafische Anzeige aufgefrischt werden soll. Unter Umständen muss dazu ein altes Bild durch ein neues ersetzt werden.



Die Illustration zeigt das Zusammenspiel zwischen `paint()`, `repaint()` und `update()`.

paint() : Alle Änderungen der grafischen Darstellung, die nicht programmbedingt sind, geschehen durch einen Aufruf der `paint()` Methode. Ein spezielles Konstrukt der `Graphics` Klasse, ein sogenanntes Clip Rechteck, wird benutzt, um die `paint()` Methode zu optimieren. Dadurch kann vermieden werden, dass jeweils der ganze grafische Bereich aufgefrischt werden muss. Die Clip Rechtecke "merken" sich die extern veränderten Bereiche, also jene, die neu gezeichnet werden müssen.

repaint() : mit dieser Methode wird das System angewiesen, das grafische Objekt neu zu zeichnen. Die `repaint()` Methode verursacht den AWT Thread die `update()` Methode aufzurufen. Normalerweise löscht diese den entsprechenden grafischen Bereich und ruft dann `paint()` auf.

update() : die Methode gehört zu einer AWT Komponente. Sie wird von den beiden oben beschriebenen Methoden eingesetzt, um grafische Komponenten neu anzuzeigen. Im Einzelnen geschieht folgendes:

- zuerst wird die neu zu zeichnende Komponente gelöscht, und zwar so, dass sie aussieht wie der Hintergrund
- dann wird die Farbe des Kontexts so wie der Vordergrund gefärbt.
- und schliesslich wird die grafische Komponente neu gezeichnet.

Damit das Applet Ihre grafischen Elemente auch korrekt anzeigen kann, müssen Sie sich an bestimmte Regeln halten:

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

- Sie können die grafische Darstellung weitestgehend selbst verwalten, speichern also das Bild in ein Array ab und verwalten die gesamten Pixel.
- die `paint()` *rendered* die Anzeige, basierend *lediglich* auf dem Inhalt des Modells. Damit haben Sie die Gewähr, dass die `paint()` Methode für eine korrekte Anzeige sorgen kann.
- wann immer Ihr Programm das Display ändern will, müssen Sie zuerst Ihr Modell updaten und dann die `repaint()` Methode aufrufen, die ihrerseits die `update()` Methode aufruft, ein Ablauf, der durch den AWT Thread erledigt wird.

1.9.11. Was ist der AppletViewer?

Der `appletviewer` ist eine Java Applikation, mit deren Hilfe Applets ohne Browser gestartet werden können. Jeder Java fähige Browser kann das Gleiche, wie der `appletviewer`. In der Regel werden die Applets auch in einem der Standard Browser laufen. Aber um die Entwicklungszeiten zu beschleunigen, wurde ein unabhängiges Werkzeug mit JDK mitgeliefert. Dieses Werkzeug ist der `appletviewer`.

Der `appletviewer` liest HTML von einer URL, welche auf der Kommandozeile angegeben wird. In dieser HTML Datei erwartet der `appletviewer` die Anweisungen zum Laden und Ausführen des Applets. Alle HTML Anweisungen im HTML werden ignoriert bis auf die Applet Anweisung.

1.9.12. Aufruf von Applets mit dem Appletviewer

Der `appletviewer` ist im Prinzip ein minimaler Browser. Als Argument erwartet er eine HTML Seite, die er lädt und in der die Instruktion steht, wie das Applet ausgeführt werden soll. Dies ist konkret der `<applet>` Tag, mit dem der Code spezifiziert wird, der vom `appletviewer` geladen wird.

Schauen wir uns ein Beispiel eines solchen Tags an, in der `HelloWorld.html` Datei.

Der `appletviewer` kreiert auf Grund dieser Anweisung einen Leerraum. Dieser ist eine 100 mal 100 Pixel Graphikfläche und ruft dann das Applet auf. In diesem Beispiel lädt der `appletviewer` eine Klasse `HelloWorld` und setzt deren grafischen Inhalt in diesen Leerraum:

```
<html>
  <applet code=HelloWorld.class width=100 height=100>
  </applet>
</html>
```

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.9.13. Starten des appletviewer

Der `appletviewer` öffnet entweder die HTML Seite, die als Parameter angegeben wird, oder sucht die URL, die als Parameter angegeben wurde. Beide müssen einen `<applet>` Tag oder den Namen einer URL einer HTML Seite enthalten.

```
appletviewer [-debug] urls ...
```

Falls die HTML Datei, welche vom `appletviewer` gelesen wird, keinen gültigen `<applet>` Tag enthält, tut der `appletviewer` nichts. Der `appletviewer` zeigt keine anderen HTML Tags an.

Der `appletviewer` besitzt nur eine Option, das `-debug` Flag, mit dem das Applet im Java Debugger, `jdb`, gestartet wird. Falls Sie das Applet mit der `-g` Option übersetzt haben, wird der Quellcode des Applets im Debugger sichtbar sein.

Der `appletviewer` im Beispiel startet den `appletviewer` und ein kleines Fenster, welches "Hello World!" ausgibt.



1.9.14. Die <Applet> Tag Syntax

Die Applet Tag Sytax wird unten zusammengefasst.

```
<applet
  code=appletFile.class
  width=pixels height=pixels
  [codebase=codebaseURL]
  [archive=archiveList]
  [alt=alternateText]
  [name=appletInstanceName]
  [align=alignment]
  [vspace=pixels] [hspace=pixels] >
  [<param name=appletAttribute1 value=value>]
  [<param name=appletAttribute2 value=value>]
  ...
  [alternateHTML]
</applet>
```

Der <applet> Tag besitzt eine ganze Anzahl vordefinierter Attribute, zusätzlich zur Fähigkeit Applet spezifische Parameter anzugeben.

- **code=**
Dies ist ein Mussfeld. Es gibt den Namen der Datei an, welche die Unterklasse der Applet Klasse angibt, in der das konkrete Applet implementiert wird. Die Angabe könnte auch ein Package enthalten: *package.appletFile.class*.
- **width=pixels height = pixels**
Auch dies ein Mussfeld. Es gibt die Grösse des Leerraumes an, in den die grafische Ausgabe des Applets geschrieben wird.

Falls Sie das Applet aus einer URL laden wollen, welche sich von der HTML URL unterscheidet, können Sie diese als **codebase** angeben:

- **codebase=codebaseURL**
dieses optionale Attribut gibt die URL an, ab der das Applet heruntergeladen werden kann. Falls diese Angabe fehlt, müssen die HTML und Applet Klasse im selben Verzeichnis stehen.
- **archive=archiveList**
beschreibt optionale Archive, die der Klassenlader bereits herunter laden soll. Mehrere Archive werden durch Kommata getrennt.
- **alt=alternateText**
Falls das Applet nicht gestartet werden kann, wird dieser optionale Text angezeigt.
- **name=appletInstanzName**
Mit diesem optionalen Attribut können Sie den Applets eine Identität geben und damit auch eine Applet zu Applet Kommunikation ermöglichen.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.9.15. Einige nützliche Applet Methoden

Da Sie mit einem Applet ein fast vollwertiges Java Programm vor sich haben (bis auf einige Sicherheitseinschränkungen) können Sie unter anderem auch Netzwerkklassen benutzen. Einige dieser Klassen und Methoden sind besonders nützlich. Wir nehmen deren Einsatz hier vorweg, werden diese aber später im Detail besprechen.

Die Klasse `java.net.URL` beschreibt URLs und kann eingesetzt werden, um mit diesen URL eine Verbindung aufzubauen, sofern man berechtigt ist.

In den Applets sind zwei URLs besonders wichtig, weil ab Ihnen in der Regel Sound und Videos herunter geladen werden muss:

- `getDocumentBase()` liefert ein URL Objekt, welches die aktuelle Browser Seite beschreibt.
- `getCodeBase()` liefert die URL, bei der sich die Quelle des Applets befindet.
- `getImage(URL base, String target)` liest ein Bild ab einer Datei, welche durch die URL angegeben wird. Das Ergebnis ist eine Instanz der Klasse `Image`.
- `getAudioClip(URL base, String target)` liest eine Musikdatei aus der URL. Das Ergebnis ist eine Instanz der Klasse `AudioClip`.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.9.16. Ein einfaches Applet zum Zeichnen von Vierecken

Das folgende Applet zeigt, wie man mehrere Vierecke ineinander verschachteln kann.

```
import java.awt.*;
import java.applet.*;

public class Squares extends Applet {

    int x,y,width,height,color;
    Color colors[] = { Color.blue,Color.red,Color.green,Color.yellow,
                     Color.cyan,Color.magenta,Color.pink };

    public void start() {
        x = 0;
        y = 0;
        width = 120;
        height = 120;
        color = 0;
    }

    public void paint(Graphics g) {
        if (width < 10 | height < 10) {
            start();
        }
        g.setColor(colors[color]);
        color = (++color % colors.length);
        g.drawRect(x,y,width,height);
        nextSquare();
    }

    public void update(Graphics g) {
        paint(g);
    }

    void nextSquare() {
        x += 10;
        y += 10;
        width -= 20;
        height -= 20;
        if (width > 10 & height > 10) {
            repaint();
        }
    }
}
```

Mit dem HTML Tag

```
<APPLET
  CODEBASE = "."
  CODE     = "Squares.class"
  NAME     = "TestApplet"
  WIDTH    = 400 HEIGHT = 300 HSPACE = 0 VSPACE = 0
  ALIGN    = top>
</APPLET>
```

1.9.17. Ein einfaches Bild Testbeispiel

Im Applet, welches Sie unten abgebildet sehen, wird ein Bild aus einer URL gelesen. The applet code shown below illustrates the use of the `getImage()` and `getDocumentBase()` methods. The applet gets the image file `graphics/joe.surf.yellow.small.gif` relative to the directory path returned by the `getDocumentBase` method and displays it in the appletviewer.

```
1 import java.awt.*;
2 import java.applet.Applet;
3
4 public class HwImage extends Applet {
5     Image duke;
6
7     public void init() {
8         duke = getImage(getDocumentBase(),
9             "graphics/joe.surf.yellow.small.gif");
10    }
11
12    public void paint(Graphics g) {
13        g.drawImage(duke, 25, 25, this);
14    }
15 }
```

Die `drawImage()` Methode besitzt verschiedene Parameter:

- das `Image` Objekt, welches angezeigt werden soll (Duke)
- die `x` Koordinaten für die Zeichnung
- die `y` Koordinaten für die Zeichnung
- der *image observer*.

Der `Image Observer` ist eine Klasse, welche eine Mitteilung erhält, falls sich das Bild ändert. Beispielsweise beim Laden eines oder mehrerer Bilder ist dies praktisch. Ein Bild, welches mit `getImage()` geladen wird, ändert sich im Laufe der Zeit. Dies geschieht, weil das Bild in der Regel im Hintergrund geladen wird. Jedes Mal, wenn wieder ein Teil des Bildes geladen ist, wird die `paint()` Methode aufgerufen. Dies geschieht, weil das Applet als Beobachter in der `drawImage()` Methode vermerkt wurde (`this`).

Selbsttestaufgabe

- a) Erweitern Sie das Applet so, dass mehrere Bilder geladen und angezeigt werden (primitive Bildanimation). Sie finden auf dem Server eine einfache Bildanimation, die nach diesem Prinzip funktioniert.
- b) Sie können auch einfach ein animiertes GIF verwenden. Alle Dukes der T Serie sind im `animierterDuke.gif` kombiniert. Was passiert beim Laden eines animierten Gifs im Applet?

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.9.18. Audio Clips

Nachdem wir Bilder geladen haben, Vierecke gezeichnet haben ... was liegt näher als Audio Clips.

Die Java Sprache kennt spezielle Methoden, um Audio Clips abzuspielen. Diese sind in der `java.applet.AudioClip` Klasse beschrieben. Natürlich müssten Sie dann auch noch die passende Hardware installiert haben, um Sound auszugeben.

Typischerweise werden Sie zum Abspielen eines Audio Clips die folgende Methode einsetzen:

```
play(URL soundDirectory, String soundfile);
```

oder einfacher:

```
play(URL soundURL);
```

Die Angabe des Verzeichnisses, in dem sich die Audio Datei befindet, geschieht am besten wieder als URL. Dazu verwenden wir die `getDocumentBase()` Methode:

```
play(getDocumentBase(), "bark.au");
```

In diesem Falle muss die Datei `bark.au` im selben Verzeichnis, wie die HTML Seite sein.

1.9.19. Ein einfaches Audio Test Beispiel

Das Applet unten illustriert den Einsatz der `getDocumentBase()` Methode in Verbindung mit der `play()` Methode. Das Applet gibt die Meldung "Audio Test" aus im appletviewer und spielt dann die Audio Datei `sounds/cuckoo.au`.

```
1 //
2 // HelloWorld mit Audio Sound
3 // "sounds/cuckoo.au" Datei (sagt einfach "Hi" oder "cuckoo")
4 //
5
6 import java.awt.Graphics;
7 import java.applet.Applet;
8
9 public class PlayAudioSound extends Applet {
10
11     public void paint(Graphics g) {
12         g.drawString("Audio Test ", 25, 25);
13         play(getDocumentBase(), "sounds/cuckoo.au");
14     }
15 }
```

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.9.20. Looping eines Audio Clip

Sie können auch Audio Clips analog zu den Bildern behandeln. Das heisst: Sie können Audio Clips laden und später spielen.

Die entsprechende Methode ist `getAudioClip` aus der Klasse `java.applet.Applet`:

```
AudioClip sound;  
sound = getAudioClip(getDocumentBase(), "bark.au");
```

Nachdem das Clip einmal geladen ist, kann man eine der Methoden: `play`, `loop`, oder `stop` einsetzen. Frage: wo wird festgelegt, wie lange geloopt wird, wann abgebrochen wird?

Mit der `play` Methode in `java.applet.AudioClip` kann man das Audio Clip abspielen:

```
sound.play();
```

Um das Abspielen des Clips zu starten und endlos loopen zu lassen, kann man die `loop` Methode in der `java.applet.AudioClip` Klasse einsetzen:

```
sound.loop();
```

Und um schliesslich die Schleife abubrechen, kann man die `stop` Methode in der `java.applet.AudioClip` Klasse einsetzen:

```
sound.stop();
```

1.9.21. Ein einfaches Audio Looping Test Beispiel

Das folgende Beispiel illustriert die eben besprochenen Methoden:

```
1 import java.awt.Graphics;  
2 import java.applet.*;  
3  
4 public class PlayAudioSoundLoop extends Applet {  
5     AudioClip sound;  
6  
7     public void init() {  
8         sound = getAudioClip(getDocumentBase(), "sounds/cuckoo.au");  
9     }  
10  
11     public void paint(Graphics g) {  
12         g.drawString("Audio Test", 25, 25);  
13     }  
14  
15     public void start() {  
16         sound.loop();  
17     }  
18  
19     public void stop() {  
20         sound.stop();  
21     }  
22 }
```


JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.9.22. Maus Eingabe und das Java Event Modell

Java unterstützt direkte Interaktion. Als Beispiel betrachten die Maus und die Bewegungen der Maus. Diese lösen jeweils ein Ereignis aus: die Maus bewegt sich, die Maustaste wird gedrückt, die Maustaste wird losgelassen.....

Wie das Event Modell im detail funktioniert werden wir noch kennen lernen. Hier benötigen wir nur einen Bruchteil des Know Hows.

Das etwas vom ursprünglichen Event Modell abweichende Java Event Modell (ab Java 1.1) unterstützt einen Ereignistyp für jede Art der Wechselwirkung. Wenn der Benutzer auf der Stufe Benutzerinterface eine Aktion ausübt, generiert dies eine *Event*, ein Ereignis. Events sind Objekte, welche beschreiben, was konkret geschieht. Es gibt verschiedene Standardereignisse, welche bestimmte Benutzeraktionen beschreiben.

Die Quelle für die Events (auf Stufe Benutzerinterface) sind Ergebnisse einer Aktivität des Benutzers mit oder an einer AWT Komponente. Dazu gehören beispielsweise das Klicken einer Maus. Dies generiert ein *ActionEvent*. Das *ActionEvent* ist ein Objekt (also eine Instanz einer Klasse), welches die Informationen über das Ereignis enthält:

- *ActionCommand* - den Befehl zur Aktion
- *Modifiers* - allfällige Modifiers für die Aktion

Die *Komponente* (Button, Slider, Textfeld), mit der ein Benutzer in Kontakt ist, erhält das *Event* (Objekt).

Ein Event Handler ist eine Methode, welche das Ereignisobjekt empfängt und so dafür sorgt, dass das Ereignis verarbeitet werden kann.

1.9.23. Mouse Input

Ab JDK 1.1 stehen Events für fast alles zur Verfügung. Wir betrachten hier lediglich ein spezielles, das Maus Event, weil wir später noch einmal auf Events im Allgemeinen zurück kommen. Maus Events (wie oben erwähnt sind dies Objekte) werden von Klassen abgearbeitet, welche das *MouseListener* Interface implementieren:

- *mouseClicked* - wann immer Sie die Maus anklicken (das Drücken der Maustaste und das Loslassen werden als nur eine Aktion betrachtet)
- *mouseEntered* - wann immer Sie mit der Maus auf eine Komponente fahren, zeigen
- *mouseExited* - wann immer Sie mit der Maus eine Komponente verlassen
- *mousePressed* - wann immer Sie die Maustaste drücken
- *mouseReleased* - wann immer Sie die Maustaste loslassen

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.9.24. Mouse Input Beispiel

Mit dem folgenden Beispiel fragen wir die Reaktionen der Maus ab. Zeile 3 importiert die Event Handler. In Zeile 6 erklären wir, dass es sich um ein Applet handelt und dieses Applet den Mouse Listener implementiert.

```
1 import java.awt.*;
2 import java.awt.Graphics;
3 import java.awt.event.*;
4 import java.applet.*;
5
6 public class HwMouse extends Applet implements MouseListener {
7
8     int mouseX=25, mouseY=25, index=0;
9     AudioClip beep;
10
11     public void init() {
12         beep = getAudioClip(getDocumentBase(),"beep.au");
13         addMouseListener(this);
14     }
15
16     public void paint(Graphics g) {
17         g.drawString("Hello World!", mouseX, mouseY);
18         beep.play();
19     }
20
21     public void mousePressed(MouseEvent e) {
22         mouseX = e.getX();
23         mouseY = e.getY();
24         repaint();
25     }
26
27     public void mouseReleased(MouseEvent e) { }
28
29     public void mouseExited(MouseEvent e) { }
30
31     public void mouseEntered(MouseEvent e) { }
32
33     public void mouseClicked(MouseEvent e) { }
34 }
```

Die Methode `mousePressed` auf den Zeilen 21 bis 25 im Beispiel wird immer dann aufgerufen, falls die Maus innerhalb der Appletfläche geklickt wird. Das `MouseEvent e` enthält eine Methode, mit der die aktuelle x und y Koordinaten des Mauszeigers abgefragt werden können. Die Methode ruft `repaint()` auf, um das Applet neu zu zeichnen.

```
21 public void mousePressed(MouseEvent e) {
22     mouseX = e.getX();
23     mouseY = e.getY();
24     repaint();
25 }
```

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

Der Aufruf neu zu zeichnen führt zum Ausführen der `paint` Methode auf den Zeilen 16 bis 19 und damit zur Ausgabe der Zeichenkette "Hello World!" ab den aktuellen Mauskoordinaten, die der Event Handler `mousePressed` geliefert hat. Zudem wird dann auch noch das Audio Clip `beep` gespielt.

```
16 public void paint(Graphics g) {  
17     g.drawString("Hello World!", mouseX, mouseY);  
18     beep.play();  
19 }
```

Selbsttestaufgabe

Schauen Sie in der Java Dokumentation nach, wie das `MouseEvent` definiert ist.

<javadoc/java.awt.event.MouseEvent.html>

1.9.25. Mouse Input - Live Applet

Sie können das Applet mit dem Appletviewer testen. Es befindet sich inklusive aller Batch Dateien für die Übersetzung und das Starten, auf dem Server.

Klicken Sie an verschiedenen Stellen mit der Maus... bis Ihnen das "Hello World" oder das gebiepse auf die Nerven geht.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.9.26. Lesen von Applet Parametern

Im der HTML Datei, im Applet Tag, kann man Parameter an das Applet weiterleiten, analog zu den `String args[]` im `main` bei Applikationen. Dies geschieht mit Hilfe des `<param>` Tag.

Beispielsweise:

```
<applet code=Zeichne.class width=100 height=100>
<param name=image value=duke.gif>
</applet>
```

Diese Parameter kann man im Applet lesen. Zeile 9 im folgenden Listing zeigt, wie man dies macht. `getParameter()` im Applet liest diese Werte.

```
1  import java.awt.*;
2  import java.applet.*;
3  import java.net.*;
4  public class DrawAny extends Applet {
5      Image im;
6
7      public void init() {
8          URL myPage = getDocumentBase();
9          String imageName = getParameter("image");
10     im = getImage(myPage, imageName);
11 }
12
13 public void paint(Graphics g) {
14     g.drawImage(im, 0, 0, this);
15 }
16 }
```

Falls Sie nun vergessen haben, die Datei mit dem Duke ins passende Verzeichnis zu kopieren, dann darf das Applet selbstverständlich nicht abstürzen. In diesem Fall gibt `getParameter()` einfach den Wert `null` zurück. Sie sollten dafür sorgen, dass in diesem Fall das Programm sanft reagiert.

Parameter sind typischerweise Zeichenketten, `String`. Falls Sie etwas anderes benötigen, liegt es an Ihnen die nötigen Konversionen durchzuführen.

Hier ein Beispiel wie eine Integer Zahl gelesen werden kann:

```
int speed = Integer.parseInt (getParameter ("speed"));
```

HTML unterscheidet Gross- und Kleinschreibung nicht. Sie sollten sich jedoch angewöhnen, alles gross oder klein zu schreiben.

1.9.27. Zusammenfassung

Unser Schnelldurchlauf durch das Thema Applet endet hier. Die Grundidee des Schnelldurchlaufs ist es, Ihnen einige Möglichkeiten anzudeuten. Erst dann gehen wir in die Tiefe.

1.10. Zusammenfassung

Sie haben nun bereits etwas vertiefte Kenntnisse von Java,

- können eigene Unterklassen bilden,
- Methoden überladen
- und überschreiben
- und eigene Ausnahmen definieren.

Es fehlt noch einiges aus dem Grundbaukasten von Java:

- Interfaces
- innere Klassen, anonyme Klassen,
- Eingabe Ausgabe,
- Threads,
- Basisdatentypen als Klassen (Wrapper Klassen),
- Systemprogrammierung.

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG	1
1.1. ÜBER DIESEN EINFÜHRENDE TEIL DES KURSES	1
1.2. KURS EINFÜHRUNG	2
1.2.1. <i>Klassen</i>	3
1.2.2. <i>Objekte</i>	3
1.2.3. <i>Methoden</i>	3
1.2.4. <i>Attribute</i>	3
1.2.5. <i>Kursziele</i>	4
1.3. MODUL 1 : INFORMATIONEN ZU JAVA.....	5
1.3.1. <i>Einleitung zu diesem Modul</i>	5
1.3.1.1. <i>Lernziele</i>	5
1.3.2. <i>Java ist</i> ...	6
1.3.3. <i>Ziele von Java</i>	7
1.3.3.1. <i>Die Java Virtual Machine</i>	8
1.3.3.2. <i>Garbage Collection</i>	9
1.3.3.3. <i>Sicherheit der Java Programmierumgebung</i>	10
1.3.3.3.1. <i>Interpreter</i>	11
1.3.3.3.2. <i>Klassenlader</i>	12
1.3.3.3.3. <i>Just In Time Code Generator</i>	13
1.3.3.3.4. <i>Code Verifikationsprozess</i>	13
1.3.4. <i>Die Java API Dokumentation</i>	14
1.3.5. <i>CASE_INSENSITIVE_ORDER</i>	21
1.3.6. <i>String</i>	21
1.3.7. <i>Quiz - Java Grundlagen</i>	22
1.3.8. <i>Zusammenfassung - Java Grundlagen</i>	23
1.4. MODUL 2 : JAVA BUILDING BLOCKS.....	24
1.4.1. <i>Einleitung</i>	24
1.4.1.1. <i>Lernziele</i>	24
1.4.2. <i>Primitive Datentypen besser verstehen</i>	25
1.4.2.1. <i>Konversion von Basisdatentypen</i>	25
1.4.3. <i>Objekte besser verstehen</i>	26
1.4.3.1. <i>Aggregierte Datentypen</i>	26
1.4.3.2. <i>Kreieren eines Objekts - Deklaration</i>	28
1.4.3.3. <i>Kreieren von Objekten - die new Anweisung</i>	28
1.4.3.4. <i>Arbeiten mit Referenzvariablen</i>	29
1.4.3.5. <i>Default Initialisierung und null als Referenzwert</i>	30
1.5. MODUL 3 : AUSDRÜCKE UND KONTROLLFLUSSSTEUERUNG II	31
1.5.1. <i>Einleitung</i>	31
1.5.1.1. <i>Lernziele</i>	31
1.5.2. <i>Variablendeklarationen</i>	32
1.5.2.1. <i>Automatische und Member Variablen</i>	33
1.5.2.2. <i>Variablen Initialisierung</i>	33
1.5.3. <i>Operatoren</i>	34
1.5.3.1. <i>Zeichenketten mit + verbinden</i>	34
1.5.4. <i>Evaluation Relationaler und logischer Operatoren</i>	35
1.5.4.1. <i>Short-Circuit logische Ausdrücke</i>	35
1.5.4.2. <i>Bitweise Evaluation</i>	35
1.5.5. <i>Shift Operatoren >>, << und >>></i>	36
1.5.6. <i>Promotion und Typumwandlung (Casting)</i>	39
1.5.6.1. <i>Konversion primitiver Datentypen - Anweisungen</i>	40
1.5.6.2. <i>Konversion bei Literalen</i>	41
1.5.6.3. <i>Konversion der Basisdatentypen bei Methodenaufrufen</i>	41
1.5.6.4. <i>Arithmetische Promotion bei Basisdatentypen</i>	42
1.5.6.5. <i>Einfache Datentypen und Typenumwandlung (Casting)</i>	43
1.5.7. <i>Quiz - Ausdrücke und Kontrollflusssteuerung II</i>	44
1.5.8. <i>Zusammenfassung - Ausdrücke und Kontrollflusssteuerung II</i>	45
1.6. MODUL 4 : ARRAYS II.....	46
1.6.1. <i>Einleitung</i>	46
1.6.1.1. <i>Lernziele</i>	46
1.6.2. <i>Deklarieren von Arrays</i>	47
1.6.3. <i>Kreieren von Arrays</i>	47

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.6.4.	Initialisierung von Arrays	48
1.6.5.	Arrays aus Arrays.....	50
1.6.6.	Array Grenzen.....	51
1.6.7.	Kopieren von Arrays - <code>System.arraycopy</code>	51
1.6.8.	Quiz - Arrays II.....	52
1.6.9.	Zusammenfassung - Arrays II.....	53
1.7.	MODULE 5 : OBJEKTE UND KLASSEN	54
1.7.1.	Einleitung	54
1.7.1.1.	Lernziele.....	54
1.7.1.2.	Orientieren Sie sich selbst	54
1.7.2.	Abstrakte Datentypen	55
1.7.2.1.	Assoziation von Daten mit Code	56
1.7.2.2.	Methoden als Eigenschaften der Daten	56
1.7.2.3.	Methodensyntax	57
1.7.2.4.	Pass-By-Value - Parameter werden wertmässig übergeben.....	58
1.7.2.5.	Die <code>this</code> Referenz	59
1.7.3.	Überladen von Methoden	60
1.7.4.	Vererbung und das Überschreiben von Methoden.....	61
1.7.4.1.	Der <code>instanceof</code> Operator.....	63
1.7.4.2.	Casting von Objekten - Objektumwandlung.....	64
1.7.4.3.	Überschreiben von Methoden.....	65
1.7.5.	Konstruktion und Destruktion von Objekten	67
1.7.6.	Packages	68
1.7.7.	Die <code>import</code> Anweisung	69
1.7.8.	Packages und Verzeichnisse.....	71
1.8.	MODULE 6 : EXCEPTIONS	72
1.8.1.	Einleitung	72
1.8.1.1.	Lernziele.....	72
1.8.1.2.	Orientieren Sie sich selbst	72
1.8.2.	Exceptions - Ausnahmen	73
1.8.2.1.	Exception - Beispiele.....	74
1.8.2.2.	Exception Handling - <code>try</code> und <code>catch</code> Anweisung.....	75
1.8.2.3.	Die <code>finally</code> Anweisung	76
1.8.2.4.	Einige typische Exceptions.....	77
1.8.2.5.	Unterschiedliche Exceptions	78
1.8.2.6.	Behandeln von Ausnahmen	79
1.8.2.7.	Definieren eigener Ausnahmen	79
1.8.2.8.	Beispiel.....	80
1.9.	MODUL 7 - EINFÜHRUNG IN JAVA APPLETS	82
1.9.1.	Einführung.....	82
1.9.1.1.	Lernziele.....	82
1.9.2.	Applets verstehen.....	82
1.9.2.1.	Ein Applet ist	82
1.9.3.	Applet Sicherheitseinschränkungen.....	84
1.9.4.	Die Applet Klassenhierarchie	84
1.9.5.	Ein "HelloWorld" Applet.....	85
1.9.6.	Wichtige Applet Methoden	86
1.9.7.	Der Applet Lebenszyklus	86
1.9.8.	Applet Anzeige.....	87
1.9.9.	Die <code>paint ()</code> Methode	87
1.9.10.	AWT Painting	88
1.9.11.	Was ist der <code>AppletViewer</code> ?	89
1.9.12.	Aufruf von Applets mit dem <code>Appletviewer</code>	89
1.9.13.	Starten des <code>appletviewer</code>	90
1.9.14.	Die <code><Applet></code> Tag Syntax	91
1.9.15.	Einige nützliche Applet Methoden.....	92
1.9.16.	Ein einfaches Applet zum Zeichnen von Vierecken.....	93
1.9.17.	Ein einfaches Bild Testbeispiel.....	94
1.9.18.	Audio Clips.....	95
1.9.19.	Ein einfaches Audio Test Beispiel.....	95
1.9.20.	Looping eines Audio Clip	96
1.9.21.	Ein einfaches Audio Looping Test Beispiel	96
1.9.22.	Maus Eingabe und das Java Event Modell.....	97

JAVA PROGRAMMIERUNG - EINE EINFÜHRUNG

1.9.23.	<i>Mouse Input</i>	97
1.9.24.	<i>Mouse Input Beispiel</i>	98
1.9.25.	<i>Mouse Input - Live Applet</i>	99
1.9.26.	<i>Lesen von Applet Parametern</i>	100
1.9.27.	<i>Zusammenfassung</i>	101
1.10.	ZUSAMMENFASSUNG	102

© Copyright Hinweis

Die Zeichnungen mit Duke sind von Sun Microsystem und geschützt. Bitte beachten Sie diesen Hinweis und verweisen Sie beim Kopieren von Teilen oder dem ganzen Skript auf dieses Copyright.

<http://www.sun.com>