24 Glossar

Erläuterungen zu den wichtigsten in diesem Buch eingeführten und verwendeten Fachbegriffen werden hier noch einmal in alphabetischer Reihenfolge wiederholt. Viele der Begriffe, aber nicht alle, sind allgemein üblich und verbreitet. Die folgenden Kennzeichnungen sollen einen Hinweis darauf geben, wie weit der betreffende Begriff verbreitet ist:

(a) Der Begriff ist **a**llgemein üblich und verbreitet.
(o) Der Begriff stammt aus einem **o**ffiziellen Standard oder einem ähnlichen Dokument, ist aber nicht unbedingt weit verbreitet.
(s) Ein **s**pezieller Begriff, der in diesem Buch eingeführt wird, aber wenig oder gar nicht verbreitet ist.
(a, s) Ein **a**llgemein üblicher Begriff, der in diesem Buch mit einer **s**peziellen Bedeutung eingeführt und verwendet wird.

Zu kursiv gesetzten Worten oder Wortfolgen in den Erläuterungen, z. B. *Profil* oder *einfache Anweisung*, gibt es in diesem Glossar weitere Erläuterungen (in einem HTML-Dokument würde man aus diesen Worten Hyperlinks machen).

+0/-0 (a): Zum Typ `float` gehören zwei Zahlen namens `+0` (oder einfach `0`) und `-0`. In den meisten Zusammenhängen haben diese beiden Zahlen die gleiche Wirkung (z. B. beim Vergleichen zweier `float`-Wert und beim Addieren und Subtrahieren), nur beim Dividieren und Multiplizieren unterscheiden sie sich. Z. B. ist `17/+0` gleich *infinity*, `17/-0` ist gleich *-infinity*, `+0*17` ist gleich `+0` und `-0*17` ist gleich `-0`. Für den Typ `double` gilt Entsprechendes.

abstrakte Klasse (a): Eine Klasse, die man beerben aber nicht instanziieren darf. Eine abstrakte Klasse kann alle Elemente einer konkreten Klasse enthalten und zusätzlich auch noch abstrakte Methoden.

abstrakte Methode (a): Eine abstrakte Methode hat ein *Profil*, aber keinen *Rumpf*. Ihre Vereinbarung beginnt mit dem Schlüsselwort `abstract` und anstelle eines Rumpfes wird nur ein Semikolon notiert. Eine abstrakte Methode ist immer eine *Objektmethode* (es gibt keine abstrakten *Klassenmethoden*). Abstrakte Methoden dürfen nur in *abstrakten Klassen* und in *Schnittstellen* vereinbart werden. Wenn eine konkrete Klasse K abstrakte Methoden erbt oder mit `implements` zu implementieren verspricht, muss K diese abstrakten Methoden mit konkreten Methoden („Methoden mit Rumpf") *überschreiben*.

actual parameter (a): siehe *aktueller Parameter*.

Adapter-Klasse (zu einer Schnittstelle) (a): Eine Klasse, die die abstrakten Methoden der Schnittstelle mit konkreten Methoden mit leeren Rümpfen (*Strohpuppen*) implementiert.

aktueller Parameter (engl. actual parameter, argument) (a): siehe *Parameter*.

anonymous classes (a): siehe *namenlose Klassen*.

anonymous constructor (o): siehe *namenloser Konstruktor*.

annoymous objects (a): siehe *namenlose Objekte*.

anonymous package (a): siehe *namenloses Paket*.

Anweisung (engl. statement) (a): Ein Befehl des Programmierers an den Ausführer, bestimmte *Werte* in bestimmte *Wertebehälter* zu tun. Man unterscheidet *einfache Anweisungen* (die keine anderen Anweisungen enthalten, z. B. die Zuweisung, die `return`-Anweisung und die `break`-Anweisung) und *zusammengesetzte Anweisungen* (die andere Anweisungen enthalten, z. B. die `if`-Anweisung und die `while`-Anweisung etc.).

argument (a): siehe *aktueller Parameter*.

arity (a): siehe *Stelligkeit (einer Operation)*.

array (a): siehe Reihung.

array type (a): siehe Reihungstyp.

ASCII-Code (a): Der ASCII-Code legt für 128 Zeichen (`a` bis `z`, `A` bis `Z`, `0` bis `9` und einige Sonderzeichen wie `. , ; ! ? ( ) [ ] { }` etc.) Codezahlen zwischen 0 und 127 fest. Zahlreiche Erweiterungen des ASCII-Codes legen zusätzlich für weitere 128 Zeichen Codezahlen zwischen 128 und 255 fest. Welche zusätzlichen Zeichen das sind und welche Codezahlen ihnen zugeordnet werden, ist von Land zu Land und manchmal von Programm zu Programm verschieden (deshalb gibt es so viele Varianten des ASCII-Codes). Beispiel: Die Zeichen `ä ö ü Ä Ö Ü` und `ß` werden in einer zeichenorientierten DOS-Eingabeaufforderung unter Windows häufig anders codiert als in einem grafischen Fenster (z. B. eines Editors).

assertion (assert-Anweisung) (a): siehe Zusicherung.

Assoziativität (eines zweistelligen Operators) (a): Die Infixnotation zweistelliger Operatoren ist mehrdeutig und muss durch zusätzliche Konventionen eindeutig gemacht werden. Weit verbreitet und allgemein bekannt ist z. B. die Konvention, dass der Subtraktionsoperator `-` linksassoziativ ist, d. h. in einem Ausdruck wie `8 - 5 - 3` soll der Operand zwischen den beiden Minusoperatoren (d. h. die `5`) „nach links assoziiert" werden, etwa so: `(8 - 5) - 3`. In Java ist die Zuweisung `=` ein rechtsassoziativer Operator, d. h. in einem Ausdruck wie `a = b = c` soll der Operand `b` „nach rechts assoziiert" werden, etwa so: `a = (b = c)`.

Attribut (einer Klasse, engl. field) (a): Eine Variable, die innerhalb einer Klasse vereinbart wurde.

aufrufen (engl. to call) (a): Aufrufen kann man nur *Unterprogramme* (oder *Methoden*, *Funktionen*, *Prozeduren*). Dabei muss man dem Unterprogramm für jeden seiner *formalen Parameter* einen entsprechenden *Wert* als *aktuellen Parameter* übergeben. Diese aktuellen Parameterwerte beschreibt der Programmierer durch *Ausdrücke*. *Klassen*, *Objekte*, andere Module und *Variablen* kann man *benutzen*, aber nicht aufrufen.

Aufzählungstyp (engl. enumeration type) (a): In Java: Ein Klassentyp, dessen Vereinbarung mit dem Schlüsselwort `enum` und einer Aufzählung von Namen für die Werte des Typs beginnt.

Ausdruck (engl. expression) (a): Ein Befehl des Programmierers an den Ausführer, einen Wert zu berechnen. Ausdrücke bestehen aus Namen von Variablen, Funktionsaufrufen, Operatoren und runden Klammern. Man unterscheidet einfache Ausdrücke (die keine anderen Ausdrücke enthalten, z. B. `123` oder `x`) und zusammengesetzte Ausdrücke (die andere Ausdrücke enthalten, z. B. `123 + x`).

Ausführer (engl. exer) (s): Eine Rolle im Rollenspiel des Programmierens. Der Ausführer prüft Programme, lehnt sie ab oder akzeptiert sie und führt sie aus. Der Begriff des Ausführers soll alles zusammenfassen, was man zum Ausführen eines Programms benötigt, z. B. eine bestimmte Hardware, ein Betriebssystem, einen Compiler, eine Standardbibliothek, einen Interpreter und evtl. weitere Werkzeuge.

Ausnahme (engl. exception) (a): Ein Objekt einer *Ausnahmeklasse*. Wird (mit der *throw*-Anweisung) geworfen, wenn eine Ausnahmesituation auftritt (z. B. wenn eine Ganzzahl durch 0 dividiert

wird). Wird eine Ausnahme nicht gefangen (mit einem `try-catch`-Befehl) bewirkt sie einen Abbruch der Programmaussführung und die Ausgabe einer Fehlermeldung.

Ausnahmeklasse (engl. exception class) (o): Eine (direkte oder indirekte) Unterklasse der Klasse `java.lang.Throwable`. Nur Objekte von Ausnahmeklassen können mit der *throw*-Anweisung als Ausnahmen geworfen werden. Siehe auch *geprüfte Ausnahmen* und *ungeprüfte Ausnahmen*.

auto boxing (a): siehe *Autohüllen*.

Autohüllen (s): Das automatische Verhüllen bzw. Enthüllen eines primitiven Wertes. Beim Verhüllen wird der primitive Wert in ein *Hüllobjekt* eingehüllt, beim Enthüllen wird er aus dem Objekt wieder zurückgewonnen.

Bauplanaspekt (einer Klasse) (s): Dazu gehören alle nicht mit `static` gekennzeichneten *Elemente* der Klasse (engl. instance members). Diese Elemente werden in jedes Objekt der Klasse eingebaut.

Bedingung (engl. condition) (a): Ein *Ausdruck* des Typs `boolean`.

Bedingungsoperation (a): Eine Art `if`-Befehl, der aber keine *Anweisung* ist, sondern zum Bilden von *Ausdrücken* dient (ein „Ausdrucks-if-Befehl“). Diese Operation ist dreistellig, ihr Name (der Bedingungsoperator) besteht aus zwei Teilen (einem Fragezeichen und einem Doppelpunkt) und wird *mixfix* notiert. Beispiel: `a < b ? a+b : 2*c + 17` Für diesen Ausdruck gilt: Wenn `a<b` ist, hat er den Wert `a+b` und sonst den Wert `2*c + 17`.

beerben (eine Oberklasse, engl. to inherit from) (a, s): In Java darf/muss jede *Klasse* `U` genau eine andere Klasse `O` beerben, etwa so: `class U extends O { ... }`. Dadurch übernimmt `U` alle *Elemente* von `O`. Wenn der Programmierer eine Klasse `U` vereinbart und nicht ausdrücklich etwas anderes festlegt, beerbt seine Klasse `U` die Klasse `Object`. Die einzige Ausnahme von dieser Regel ist die Klasse `Object`, die keine andere Klasse beerbt. Weil jede Klasse nur **eine** Klasse beerben darf, spricht man auch von einfacher Beerbung. Mit dem Wort *erweitern* kann man das Gleiche ausdrücken wie mit beerben (jede *Klasse* U darf/muss genau eine andere Klasse O *erweitern* etc.).

Befehl (s): Ein Befehl ist entweder eine *Vereinbarung* oder ein *Ausdruck* oder eine *Anweisung* (oder einer der wenigen Befehle, die sich nur schwer in eine dieser drei Gruppen einordnen lässt).

Behälter (engl. container) (o): Ein Objekt einer *Behälterklasse*. In ein Behälter-Objekt kann man *Grabo-Komponenten* einfügen.

Behälterklasse (engl. containerclass) (o): In Java: Eine Unterklasse der Klasse `java.awt.Container`.

Behandler-Methode (engl. handler method) (a, s): Eine Methode, die als Reaktion auf *Ereignisse* einer bestimmten Art ausgeführt wird (in einem *Grabo-Programm*).

Behandler-Objekt (a, s): Ein Objekt, welches *Behandler-Methoden* enthält und bei einem *Grabo-Objekt* angemeldet werden kann.

benutzen (eine Klasse) (a, s): Eine Klasse kann man als *Bauplan* benutzen (instanziieren) oder als *Modul* benutzen. Beispiele: Mit einem Ausdruck wie `new StringBuilder("Hallo!")` instanziiert man die Klasse `StringBuilder`. Mit einem Ausdruck wie `Math.max(a, b)` und mit einer Anweisung wie `System.out.println()` benutzt man die Klasse `Math` bzw. `System` als Modul.

benutzen (eine Methode) (a, s): Ein Unterprogramm (eine Methode, eine Funktion, eine Prozedur) benutzt man, indem man sie *aufruft*.

benutzen (eine Variable) (a, s): Eine Variable benutzt man, indem man sie als Ausdruck oder als Teil eines Ausdrucks erwähnt oder indem man ihr einen Wert zuweist. Beispiele: In den Ausdrücken `x` und `x + 1` und in der Zuweisung `x = 17;` wird die Variable `x` benutzt.

benutzen (einen Modul): Einen Modul `m` benutzt man, indem man ein Element `e` aus dem ungeschützten (öffentlichen, sichtbaren) Teil von `m` benutzt. Falls dieses Element `m.e` ein Attribut (d. h. eine Variable) ist, kann man es als Variable benutzen. Falls das Element `m.e` eine Methode ist, kann man sie aufrufen etc. In Java kann der Modul `m` eine Klasse (z. B. `Math`) oder ein Objekt (z. B. `System.out`) sein.

Benutzer (engl. user) (s): Eine Rolle im Rollenspiel des Programmierens. Der Benutzer fordert den *Ausführer* dazu auf, Programme auszuführen und ist für Ein- und Ausgabedaten zuständig.

Beobachter (engl. observer, listener) (a): siehe *Behandler-Objekt*.

Bindungsstärke (eines zweistelligen Operators, engl. priority) (a): Die Infixnotation *zweistelliger Operatoren* ist mehrdeutig und muss durch zusätzliche Konventionen eindeutig gemacht werden. Weit verbreitet ist z. B. die Konvention „Punktrechnung geht vor Strichrechnung“, d. h. ein Ausdruck wie `3 + 2 * 5` hat den Wert `3 + (2 * 5)` gleich `30`, und nicht den Wert `(3 + 2) * 5` gleich `25`. Weniger verbreitet ist die Konvention, dass ein Ausdruck wie `true || false && false` den Wert `true`, und nicht den Wert `false` hat. In Java hat jeder *Operator* eine Bindungsstärke zwischen 1 und 13. *Operationen* mit höherer Bindungsstärke werden vor solchen mit niedrigerer Bindungsstärke ausgeführt. Nur wenige *Kollegen* kennen die Bindungsstärken aller Operatoren auswendig. In Zweifelsfällen sollte man deshalb die Bedeutung von Ausdrücken durch runde Klammern deutlich machen.

Blank (engl. blank, space) (a): siehe *SPACE*.

Block: eine zusammengesetzte Anweisung. Dient dazu, mehrere Vereinbarungen und Anweisungen zu einer Anweisung zusammenzufassen.

Boje (a, s): Eine grafische Darstellung einer Variablen. Stellt die einzelnen Teile einer Variablen (die Referenz, den Wert, den Namen und den Zielwert) durch Kästchen verschiedener Formen dar. Betont den Unterschied zwischen „normalen Werten“ (in viereckigen Kästchen) und Referenzen (in sechseckigen Kästchen). Wurde im Zusammenhang mit der Sprache Algol68 erfunden.

boolean (a): In Java ein *primitiver*, *nicht-numerischer* Typ, zu dem nur zwei Werte gehören, `true` und `false`. Diese Werte können **nicht** mit Cast-Befehlen in Werte eines anderen Typs (z. B. eines *numerischen* Typs) umgewandelt werden.

break: eine *einfache Anweisung*, mit der man eine *Schleife* oder eine *switch*-Anweisung beenden kann. Siehe auch *continue*.

button (a, s): siehe *Knopf*.

by value (a): siehe *Parameterübergabe per Wert*.

Bytecode-Interpreter (a): Ein Programm, welches Bytecode-Dateien (einlesen und) ausführen kann. Beispiele: Das Programm `java` der Firma Sun und das Programm `gij` aus dem Gnu-Projekt sind Bytecode-Interpreter.

byteorientierter Strom (engl. byte stream) (a): Dient zum Einlesen und Ausgeben von binären Daten, z. B. von Objekten, von Bildern und Audiodaten. Wenn es um Zeichen oder Texte geht, wird meistens ein *zeichenorientierter Strom* mit einem byteorientierten Strom kombiniert (d. h. auch Zeichen und Texte werden letztlich als binäre Daten ein- und ausgegeben).

call (a subprogram or a method): siehe *aufrufen*.

call by reference (a): Sprachlich missglückte Bezeichnung für *pass by reference*. Siehe *Parameterübergabe per Referenz*.

call by value (a): Sprachlich missglückte Bezeichnung für *pass by value*. Siehe *Parameterübergabe per Wert*.

Cast-Befehl (a): Ähnelt einem Funktionsaufruf (dient dazu, einen Wert zu berechnen). Bewirkt eine *Typumwandlung* zwischen nah miteinander verwandten Typen (zwischen zwei *Referenztypen*, von denen der eine ein Untertyp des anderen ist, oder zwischen zwei *numerischen Typen*). Beispiel: Der Cast-Befehl `(int) 17.85` berechnet aus dem `double`-Wert `17.85` den `int`-Wert `17`.

Kein Typ ist mit dem Typ `String` oder mit dem Typ `boolean` nah verwandt (Umwandlungen in diese Typen können also nicht mit Cast-Befehlen bewirkt werden).

character stream (a): siehe *zeichenorientierter Strom*.

checked exception (o): siehe *geprüfte Ausnahme*.

collection (o): siehe *Sammlung*.

collection class (o): siehe *Sammlungsklasse*.

compound expression (a): siehe *zusammengesetzter Ausdruck*.

compound statement (a): siehe *zusammengesetzte Anweisung*.

concurrent (a): siehe *nebenläufig*, *Prozess*, *Faden*

container (o): siehe *Behälter*.

container class (o): siehe *Behälterklasse*.

continue (a): eine *einfache Anweisung*, mit der man eine Ausführung eines Schleifenrumpfes (aber nicht „die ganze Schleife“) beenden kann. Siehe auch *break*.

conversion (a): siehe *Typumwandlung*.

CR-Zeichen (a): Das Unicode-Zeichen CARRIAGE RETURN („Wagenrücklauf“, gemeint ist der „Wagen“ einer mechanischen Schreibmaschine) mit dem Code `u000D` (kann auch durch das `char`-Literal `'\r'` bezeichnet werden). Siehe auch *Zeilenende-Markierung*.

deadlock (o): siehe *Verklemmung*.

declaration (a): siehe *Vereinbarung*.

direkte Oberklasse (o): Durch `class DU extends DO {...}` vereinbart man die Klasse `DU` als Unterklasse der direkten Oberklasse `DO`. Jede Klasse `DU` hat genau eine direkte Oberklasse `DO` (nur die Klasse `Object` hat keine direkte Oberklasse und auch keine anderen Oberklassen).

direkte Unterklasse (o): Durch `class DU extends DO {...}` wird die Klasse `DU` als eine direkte Unterklasse der Klasse `DO` vereinbart. Man kann beliebig viele Klassen `DU1`, `DU2`, ... als direkte Unterklassen einer Klasse `DO` vereinbaren.

do-while (a): eine *zusammengesetzte Anweisung*, eine *Wiederholungsanweisung* (*Schleife*).

DOS-Eingabeaufforderung (unter Windows) (a): siehe *Konsole*.

dreistellig (a): siehe *Stelligkeit*.

einfache Anweisung (engl. simple statement) (a): Eine *Anweisung*, die (möglicherweise *Ausdrücke* aber) keine Anweisungen enthält.

einfacher Ausdruck (engl. simple expression) (a): Entweder ein *Literal* oder der Name einer *Variablen*. Solche Ausdrücke enthalten keine anderen Ausdrücke.

einstellig (a): siehe *Stelligkeit*.

Element (einer Klasse, engl. member) (a): Eine *Variable*, ein *Unterprogramm*, eine *Klasse* oder *Schnittstelle*, die innerhalb einer Klasse vereinbart wurde (solche Variablen werden häufig als *Attribute* und solche Unterprogramme als *Methoden* bezeichnet). In Java zählen *Konstruktoren* offiziell nicht zu den Elementen ihrer Klasse. Die Elemente einer Klasse kann man auf verschiedene Weisen in Gruppen einteilen: Nach ihrer Erreichbarkeit (`public`, `protected`, paketweit und `private`), nach ihrer „Aspektzugehörigkeit“ (Klassenelemente, Objektelemente) und nach ihrer Art (Attribut, Methode, Klasse oder Schnittstelle).

empty array (a): siehe *leere Reihung*.

empty body (a): siehe *leerer Rumpf*.

empty collection (a): siehe *leere Sammlung*.

empty statement (a): siehe *leere Anweisung*.

empty string (a): siehe *leerer String*.

enum type (a): siehe *Aufzählungstyp*.

Ereignis (engl. event) (a): Wenn ein Benutzer mit einer grafischen Benutzeroberfläche interagiert, indem er z. B. einen Knopf (engl. button) anklickt, löst er damit ein Ereignis und (wenn der

Programmierer entsprechende Vorkehrungen getroffen hat) die Ausführung bestimmter Behandlermethoden aus.

Ereignisfaden (engl. event thread) (o, s): siehe *Faden*.

Ergebnistyp (einer Methode, engl. result type, return type) (a): In Java muss man beim Vereinbaren einer *Methode* unmittelbar vor ihrem Namen immer einen Ergebnistyp festlegen, bei *Prozeduren* (die nichts zurückgeben) den Pseudotyp *void* und bei *Funktionen* (die etwas zurückgeben) einen „richtigen Rückgabetyp“ wie `int` oder `String` etc.

ersetzen (geerbte Elemente) (s): In einer Klasse `K` kann man ein geerbtes Element durch ein *homonymes*, in `K` vereinbartes Element ersetzen. Es gibt zwei Arten von ersetzen: *verdecken* und *überschreiben*. Objektmethoden werden „endgültig“ überschrieben, alle anderen Elemente (Klassenmethoden, Klassenattribute und Objektattribute) werden „nur“ verdeckt.

erweitern (eine Oberklasse, engl. to extend) (a): In Java darf/muss jede *Klasse* `U` genau eine andere Klasse `O` erweitern, etwa so: `class U extends O { ... }`. Dadurch übernimmt `U` alle *Elemente* von `O`. Wenn der Programmierer eine Klasse `U` vereinbart und nicht ausdrücklich etwas anderes festlegt, erweitert seine Klasse U die Klasse `Object`. Die einzige Ausnahme von dieser Regel ist die Klasse `Object`, die keine andere Klasse erweitert. Mit dem Wort *beerben* kann man das Gleiche ausdrücken wie mit erweitern (jede *Klasse* U darf/muss genau eine andere Klasse O *beerben* etc.).

Etikett (engl. label) (a, s): Ein *Grabo-Objekt*, welches zu einer der Klassen `Label`, `JLabel`, `BasicComboBoxRenderer`, `DefaultListCellRenderer`, `DefaultTableCellRenderer`, `DefaultTreeCellRenderer` gehört.

event (a): siehe *Ereignis*.

event thread (a): siehe *Faden*.

exception (o): siehe *Ausnahme*.

exer (s): siehe *Ausführer*.

expression (a): siehe *Ausdruck*.

Faden (engl. thread of control) (o, s): Beim Ausführen eines Programms besteht der Java-Ausführer aus mindestens einem Faden, dem Hauptfaden (main thread). Er kann sich aber (auf Befehl des Programmierers) „in mehrere Fäden aufspalten“. Jeden Faden kann man sich als einen selbständigen Ausführer vorstellen, der nebenläufig zu (concurrently, zeitlich unabhängig von) den anderen Fäden, bestimmte Befehle des Programms ausführt. Dabei kann es passieren, dass mehrere Fäden z. B. dieselbe Methode (nebenläufig zueinander) ausführen. Ein Programm mit `awt-` oder `swing`-*Grabo* wird automatisch von mindestens zwei Fäden ausgeführt, dem Hauptfaden (main thread), der die *main*-Methode ausführt, und einem Ereignisfaden (event thread), der auf die vom Benutzer ausgelösten Ereignisse (Maus-Klicks, Tastendrücke etc.) reagiert. Auf heute noch üblichen Computern mit nur einem Hauptprozessor werden Fäden „stückchenweise abwechselnd“ ausgeführt. Um dabei von einem Faden zu einem anderen zu wechseln müssen relativ wenig Maschinenbefehle ausgeführt werden („ein Fadenwechsel ist billig“). Siehe auch *Prozess*. In Java ist ein Faden ein Objekt der Klasse `java.lang.Thread`.

Fallunterscheidungsanweisung (a, s): Eine *zusammengesetzte Anweisung* mit der man bewirken kann, dass die darin enthaltenen Anweisungen weniger als einmal (also nicht) ausgeführt werden. Siehe auch *if* und *switch*.

field (of a class) (a): siehe *Attribut (einer Klasse)*.

final (a): Eine mit `final` vereinbarte Variable ist unveränderbar (d. h. ihr Wert kann nicht verändert werden). Eine mit `final` vereinbarte Methode kann nicht *überschrieben* oder *verdeckt* werden. Eine mit `final` vereinbarte Klasse kann nicht erweitert (beerbt) werden.

floatingpoint arithmetic (a): siehe *Gleitpunktarithmetik*.

for: eine *zusammengesetzte Anweisung*, eine *Wiederholungsanweisung* (*Schleife*).

formaler Parameter (engl. formal parameter) (a): siehe *Parameter*.

Funktion (engl. function) (a): Ein *Unterprogramm*, welches einen Wert liefert, wenn man es aufruft. Gegensatz: *Prozedur*. Funktionen dienen dazu, einen Wert zu berechnen. Der Aufruf einer Funktion ist ein *Ausdruck*.

funktionale Programmiersprache (engl. functional programming language) (a): Eine Programmiersprache, bei der alle *Unterprogramme Funktionen* sind, die keine veränderbaren *Variablen* und keine *Anweisungen* (insbesondere keine *Zuweisung*) enthält, nur *Vereinbarungen* und *Ausdrücke*. Nur wenige verbreitete Programmiersprachen sind funktional (z. B. SQL und XSLT), es gibt aber zahlreiche weniger verbreitete funktionale Sprachen (z. B. Lisp, Scheme, Miranda und Opal). Gegensatz: *prozedurale Programmiersprache*.

Ganzzahlarithmetik (integer arithmetic) (a): Dazu gehören die Typen `byte`, `char`, `short`, `int` und `long` und alle Regeln, nach denen der Java-Ausführer mit Werten dieser Typen rechnet.

generische Einheit (engl. generic unit) (a): Eine *Klasse*, *Schnittstelle* oder *Methode*, in deren Vereinbarung anstelle von konkreten Typen (wie `String`, `Integer` etc.) Typ-Parameter (wie `S`, `T`, `K` etc.) vorkommen (Beispiele: `ArrayList<K>` ist eine generische Klasse und `Set<K>` ist eine generische Schnittstelle. Bei beiden ist K ein Typ-Parameter).

geprüfte Ausnahme (engl. checked exception) (o): Ein Objekt der Klasse `Exception` oder einer Unterklasse. Wenn in einem Unterprogramm eine geprüfte Ausnahme auftreten kann, muss sie entweder gefangen-und-behandelt werden (mit einem `try-catch`-Befehl) oder sie muss am Anfang der Methodenvereinbarung nach dem Schlüsselwort `throws` erwähnt (und damit dokumentiert) werden.

getypte/ungetypte Variable (engl. typed/untyped variable) (a): Auf eine ungetypte Variable darf man beliebige Befehle anwenden und sie darf beliebige Werte (Ganzzahlen, Bruchzahlen, Zeichenketten, Wahrheitswerte etc.) enthalten. Auf eine getypte Variable eines Typs T darf man nur die Befehle von T anwenden und sie darf nur Werte des Typs T enthalten.

Gleitpunktarithmetik oder Gleitkommarithmetik (floatingpoint arithmetic) (a): Dazu gehören die Typen `float` und `double` und alle Regeln, nach denen der Java-Ausführer mit Werten dieser Typen rechnet.

Grabo (engl. GUI, graphical user interface) (s): Eine Grafische Benutzeroberfläche. Eine Grabo besteht aus Fenstern, Knöpfen, Menüs etc., mit denen der Benutzer interagieren kann (indem er z. B. mit einer Maus einen Knopf anklickt oder einen Eintrag in einem Menü wählt).

Grabo-Klasse (engl. GUI class) (s): In Java: Eine Unterklasse der Klasse `java.awt.Component`.

Grabo-Komponente (engl. GUI component) (s): In Java: Alternative Bezeichnung für ein *Grabo-Objekt*. (ein Objekt einer Unterklasse der Klasse `java.awt.Component`).

Grabo-Objekt (engl. GUI object) (s): Objekt einer *Grabo-Klasse*. Ein Grabo-Objekt besitzt eine bestimmte grafische Darstellung die (mehr oder weniger automatisch) auf dem Bildschirm erscheint, wenn das Objekt erzeugt wird.

GUI (a): graphical user interface, siehe *Grabo*.

Hauptfaden (engl. main thread) (o, s): siehe *Faden*.

Hauptklasse (eines Java-Programms, engl. main class) (s): Ein Java-Programm namens `Otto` besteht im Kern aus einer einzigen Klasse namens `Otto` (Datei: `Otto.class`), die hier als Hauptklasse des Programms bezeichnet wird. Die Hauptklasse muss eine `main`-Methode enthalten. Alle Klassen, die der Ausführer (zusätzlich zur Klasse `Otto`) benötigt, um diese `main`-Methode auszuführen, gehören als Nebenklassen zum Programm. Eine Nebenklasse darf eine `main`-Methode enthalten, muss aber nicht.

hide (an object field) (a): siehe *verdecken*.

homonym (bei Attributen oder Klassen) (s): Zwei Attribute bzw. zwei Klassen sind homonym, wenn sie gleiche Namen haben. In einer Klasse `K` wird ein geerbtes Attribut durch ein in `K` vereinbartes homonymes Attribut *verdeckt*.

homonym (bei Methoden) (s): Zwei Methoden sind homonym, wenn sie gleiche *Signaturen* haben. In einer Klasse `K` wird eine geerbte Objektmethode durch eine in `K` vereinbarte homonyme Objektmethode *überschrieben*.

Hüllklasse (engl. wrapper class) (a): Zu jedem der acht *primitiven Typen* (`byte`, `char`, `short`, `int`, `long`, `float`, `double`, `boolean`) gibt es eine entsprechende Hüllklasse (`Byte`, `Character`, `Short`, `Integer`, `Long`, `Float`, `Double`, `Boolean`). Will man einen primitiven *Wert* wie ein *Objekt* behandeln, muss man ihn in ein Objekt der entsprechenden Hüllklasse einhüllen und kann ihn später wieder enthüllen (aus dem Hüllobjekt zurückgewinnen). In vielen Situation erledigt der Ausführer das Ver- und Enthüllen automatisch (siehe *Autohüllen*).

Hüllobjekt (engl. wrapper object) (a): Objekt einer *Hüllklasse*. Dient im Wesentlichen dazu, einen primitiven Wert „als Objekt zu verkleiden“.

if: eine *zusammengesetzte Anweisung*, eine *Fallunterscheidungsanweisung*.

implementieren (a): Allgemein: Eine Problemlösung konkret als Programm realisieren (z. B. einen Sortieralgorithmus implementieren). Speziell in Java: Eine *Klasse* kann beliebig viele *Schnittstellen* implementieren, d. h. ihre *abstrakten Methoden* mit konkreten *Methoden überschreiben*. Die implementierten Schnittstellen sollten am Anfang der Klassenvereinbarung nach `implements` angegeben werden.

import (a): Ein Java-Befehl, mit dem man Abkürzungen einführen kann. `import`-Befehle müssen immer ganz am Anfang einer Quelldatei stehen (nur ein *package*-Befehl darf/muss noch davor stehen). Beispiel: Nach dem Befehl `import java.util.ArrayList;` darf man anstelle des vollen Klassennamens `java.util.ArrayList` auch den einfachen Klassennamen `ArrayList` benutzen. Nach dem Befehl `import.java.util.*;` darf man alle Klassen aus dem Paket `java.util` mit ihren einfachen Namen statt mit ihren vollen Namen bezeichnen. Siehe auch *static import*. Das Wort „import“ suggeriert etwas anderes, als der Befehl tatsächlich leistet.

infinity/-infinity (a): Zum Typ `float` gehören zwei spezielle Zahlen namens `infinity` und `-infinity`. Diese Zahlen repräsentieren alle reellen Zahlen, die zu groß (bzw. zu klein) sind, um als endliche `float`-Zahl dargestellt zu werden. Für den Typ `double` gilt Entsprechendes.

Infixnotation (a): Ein mittelalterlicher Brauch, zweistellige Operatoren **zwischen** ihre Operanden zu schreiben (statt davor oder dahinter). Die Infixnotation ist mehrdeutig und muss durch zusätzliche Konventionen eindeutig gemacht werden. Siehe auch *Assoziativität* und *Bindungsstärke*.

Instanz (einer Klasse, engl. instance) (a): siehe *Objekt (einer Klasse)*.

instanziieren (eine Klasse, engl. to instantiate) (a): Eine Instanz einer Klasse (ein Objekt) erzeugen lassen. Mit dem Ausdruck `new StringBuilder("Hallo!")` befiehlt der Programmierer dem Ausführer, eine Instanz der Klasse `StringBuilder` zu erzeugen.

integer arithmetic (a): siehe Ganzzahlarithmetik.

interface (o): siehe *Schnittstelle*.

Interpreter (a): Ein Programm, welches Programme (einlesen und) ausführen kann. Siehe auch *Quellcode-Interpreter* und *Bytecode-Interpreter*.

Klasse (engl. class) (o, s): Ein *Modul* und ein Bauplan für Module. Die nach einem solchen Bauplan gebauten Module werden allgemein als *Objekte* oder als *Instanzen* der Klasse bezeichnet. Eine Klasse enthält Elemente und Konstruktoren. Jedes Element ist entweder ein Klassenelement (mit `static` gekennzeichnet) oder ein Objektelement (nicht mit `static` gekennzeichnet). Die *Klassenelemente* und Konstruktoren gehören zum *Modulaspekt* einer Klasse, die *Objektelemente* zum *Bauplanaspekt*.

Klassenattribut (engl. static field, class variable) (a, s): Ein mit *static* gekennzeichnetes *Attribut* einer Klasse. Gehört zum *Modulaspekt* der Klasse. Wird erzeugt, wenn die Klasse erzeugt wird.

Klassenelement (einer Klasse, engl. static member): Ein mit `static` gekennzeichnetes *Element* einer *Klasse*. Gehört zum Modulaspekt der Klasse. Wird erzeugt, wenn die Klasse erzeugt wird. Beispiel: Die `main`-Methode eines Java-Programms muss als Klassenelement der *Hauptklasse* vereinbart werden.

Klasseninitialisierer (engl. static initialiser) (o): Eine Blockanweisung, die direkt in einer Klasse steht und mit `static` gekennzeichnet ist. Wird genau einmal ausgeführt wenn die Klasse geladen (d. h. erzeugt) wird. Der Programmierer kann z. B. komplizierte Befehle (*Schleifen* etc.) zum Initialisieren von Klassenattributen in solche Klasseninitialisierer schreiben.

Klassenmethode (engl. static method) (a, s): Eine mit *static* gekennzeichnete *Methode* einer *Klasse*. Gehört zum *Modulaspekt* der Klasse. Wird erzeugt, wenn die Klasse erzeugt wird. Beispiel: Die `main`-Methode ist eine Klassenmethode ihrer Klasse.

Knopf (engl. button) (a, s): Ein *Grabo-Objekt*, welches zu einer der Klassen `Button`, `JButton`, `JCheckBox`, `JRadioButton` etc. gehört.

Kollegen (s): Der *Warter* und der *Verwender* werden hier gemeinsam als die Kollegen des *Programmierers* bezeichnet.

Kommentar (engl. comment) (a): Eine Erläuterung zu einem Programm, die der *Programmierer* für seine *Kollegen* in das Programm geschrieben hat. Der *Ausführer* ignoriert Kommentare.

Komponente (einer Reihung, engl. component of an array) (a): siehe *Reihung*.

Komponente (einer Sammlung, engl. element of a collection) (a): Ein Objekt, welches in eine *Sammlung* eingefügt wurde.

Konsole (engl. console) (a): Bestand früher aus einer Tastatur und einem zeichenorientierten Bildschirm. Besteht heute meist aus einer Tastatur und einem speziellen Fenster auf einem grafischen Bildschirm, in das der Benutzer Kommandos eingeben kann. Eine Konsole wird unter Linux auch als Shell (-Fenster) und unter Windows als DOS-Eingabeaufforderung bezeichnet.

Konstante (engl. constant) (a, s): Ein Name für einen *Wert*. Welchen Wert eine Konstante bezeichnet, wird vom Programmierer (in der *Vereinbarung* der Konstanten) festgelegt. Eine Konstante hat (anders als eine *unveränderbare Variable*) keine Referenz oder Adresse. Siehe auch *Literal* und *Wert*.

Konstruktor (einer Klasse, engl. constructor) (a): Ähnelt einer *Methode*, unterliegt aber speziellen Regeln. Ein Konstruktor heißt so wie seine Klasse, wird immer unmittelbar nach dem `new`-Befehl aufgerufen (z. B. `new StringBuilder("Hallo!")`) und dient dazu, das neu erzeugte Objekt zu initialisieren (die Bezeichnung „Initialisator“ wäre intuitiver, aber „Konstruktor“ hat sich allgemein durchgesetzt). Wenn man innerhalb einer Klasse mehrere Konstruktoren vereinbart, müssen sie sich durch die Anzahl und/oder die Typen ihrer Parameter unterscheiden. In Java zählen Konstruktoren offiziell nicht zu den *Elementen* einer Klasse und werden auch nicht an Unterklassen vererbt. Um in einem Konstruktor einen Konstruktor der selben Klasse (bzw. der direkten Oberklasse) aufzurufen, gibt man nicht dessen Namen an, sondern das Schlüsselwort `this` (bzw. `super`).

label (a): siehe *Etikett*.

leere Anweisung (engl. empty statement) (a): Eine Anweisung die nichts bewirkt (nichts Nützliches und nichts Schädliches). Wird in Java durch ein Semikolon dargestellt. In Java ist der Programmierer nie gezwungen, eine leere Anweisung in sein Programm zu schreiben.

leere Reihung (engl. empty array) (a): Eine *Reihung* der Länge 0.

leerer Rumpf (engl. empty body) (a): Der Rumpf einer Schleife oder einer Methode kann leer sein.

leere Sammlung (engl. empty collection) (a): Eine *Sammlung* (z. B. ein `Vector`-Objekt oder ein `Set`-Objekt) der Länge 0.

leerer String (engl. empty string) (a): Ein String der Länge 0.

leere Variable (s): Gibt es nicht. Eine Variable enthält immer genau einen Wert und kann nicht leer sein.

Leerzeichen (a): siehe *SPACE*.

LF-Zeichen (a): Das Unicode-Zeichen LINE FEED („Zeilenvorschub“) mit dem Code `u000A`. Kann auch durch das `char`-Literal `'\n'` bezeichnet werden. Siehe auch *Zeilenende-Markierung*.

listener (a): siehe *Behandler-Objekt*.

Literal (engl. literal) (a, s): Ein Name für einen *Wert*, z. B. `123`, `3.7`, `'A'`, `"ABC"` , `true`, `false`. Welchen Wert ein Literal bezeichnet, wird vom Ausführer festgelegt und kann vom Programmierer nicht verändert werden. Siehe auch *Konstante* und *Wert*.

main thread (a): siehe *Faden*.

mehrdimensionale Reihung (a): Eine *Reihung*, deren Komponenten Reihungen sind (und nicht nur Referenzen auf Reihungen). Die in einer mehrdimensionalen Reihung enthaltenen Reihungen müssen alle gleich lang sein. Zum Kern der Sprache Java gehören *mehrstufige Reihungen*, aber keine mehrdimensionalen Reihungen. Die Firma IBM hat aber eine Java-Klasse entwickelt, deren Objekte sich genau wie mehrdimensionale Reihungen verhalten (siehe `www.alphaWorks.ibm.com`).

mehrstufige Reihung (s): In Java eine *Reihung* von Reihungen, d. h. eine Reihung, deren *Komponenten* Referenzen auf Reihungen enthalten. Die in einer mehrstufigen Reihung enthaltenen Reihungen können unterschiedlich lang sein. Gegensatz: *mehrdimensionale Reihungen*.

member (of a class) (a): siehe *Element (einer Klasse)*.

Methode (engl. method) (a): Eine Methode ist ein *Unterprogramm*, welches innerhalb einer Klasse vereinbart wurde. Siehe auch *Funktion*, *Prozedur*.

Mixfixnotation (a): Einige *Operatoren* (Namen für Operationen) bestehen aus mehreren Teilen, z. B. der Cast-Operator aus einer öffnenden und einer schließenden runden Klammer und der Bedingungsoperator aus einem Fragezeichen und einem Doppelpunkt. Wenn man einen solchen mehrteiligen Operator verwendet, muss man eine „gemischte Folge von Operatorteilen und Operanden“ notieren und bezeichnet das als Mixfixnotation.

Modul (a): Ein Behälter für *Variablen*, *Unterprogramme*, weitere *Module* etc., der aus mindestens zwei Teilen besteht, einem geschützten (privaten, unsichtbaren) Teil und einem ungeschützten (öffentlichen, sichtbaren) Teil. Von Stellen ausserhalb eines Moduls kann man nur auf die Elemente zugreifen, die sich im ungeschützten Teil befinden.

Modulaspekt (einer Klasse) (s): Dazu gehören alle mit `static` gekennzeichneten *Elemente* der Klasse und alle *Konstruktoren*. Diese Elemente und Konstruktoren werden erzeugt, wenn die Klasse erzeugt (geladen) wird.

namenlose Klassen (anonymous classes) (o): In Java kann man nach `new` anstelle eines Typnamens auch die Vereinbarung einer namenlosen Klasse angeben, z. B. so: `new ActionListener() {public void actionPerformed(ActionEvent ae) { ... } }`

namenloser Konstruktor (anonymous constructor) (o): Eine *namenlose Klasse* hat genau einen namenlosen Konstruktor und keine weiteren Konstruktoren.

namenlose Objekte (anonymous objects) (o): In Java kann man Objekte erzeugen lassen (mit `new`), ohne ihnen Namen zu geben, z. B. als Parameter im Aufruf einer Methode.

namenloses Paket (anonymous package) (o): Dieses *Paket* hat, wie sein Name schon andeutet, keinen Namen (zumindest keinen, den man in einem `package`-Befehl angeben dürfte). Wenn man beim Vereinbaren einer Klasse `K` nicht ausdrücklich etwas anderes festlegt (mit einem `package`-Befehl), gehört `K` zum namenlosen Paket.

NaN (a): Abkürzung für `Not a Number`, siehe *Unzahlen*.

Nebenklasse (eines Java-Programms) (s): Eine Klasse, welche zur Ausführung der `main`-Methode des Programms benötigt wird. Siehe auch *Hauptklasse*. Eine Klasse kann gleichzeitig die Hauptklasse eines Programms und eine Nebenklasse beliebig vieler anderer Programme sein.

nebenläufig (engl. concurrent) (a): *Prozesse* werden von einem Betriebssystem nebenläufig zueinander (concurrently, zeitlich unabhängig voneinander) ausgeführt. *Fäden* (threads) werden ebenfalls nebenläufig zueinander ausgeführt.

nicht-numerischer Typ (a): In Java ist der Typ *boolean* der einzige nicht-numerische, *primitive Typ*.

null (a): Ein Literal. Sein Wert darf in jeder Referenzvariablen stehen (unabhängig vom Typ der Variablen). Eine Variable mit diesem Wert hat keinen Zielwert (d. h. die Variable referenziert kein Objekt, sie zeigt auf kein Objekt und erst recht nicht auf null).

numerischer Typ (o): Einer der sieben primitiven Typen `byte`, `char`, `int`, `long`, `float`, `double`. Die Klassentypen `BigInteger` und `BigDecimal` zählen offiziell nicht zu den numerischen Typen (obwohl ihre Objekte Zahlen repräsentieren und man damit rechnen kann).

Oberklasse (engl. superclass) (o): Sei `U` eine Klasse und `DO` ihre *direkte Oberklasse*. Dann sind `DO` und alle Oberklassen von `DO` Oberklassen von `U`.

Objekt (einer Klasse, engl. object, instance) (a): Ein *Modul*, der nach einem durch eine Klasse repräsentierten Bauplan gebaut wurde.

Objektattribut (non static field, instance variable) (a, s): Ein nicht mit `static` gekennzeichnetes *Attribut* einer *Klasse*. Gehört zum Bauplanaspekt der Klasse und wird in jedes *Objekt* der Klasse eingebaut.

Objektelement (einer Klasse, instance member): Ein nicht mit `static` gekennzeichnetes *Element* einer *Klasse*. Gehört zum *Bauplanaspekt* der Klasse und wird in jedes *Objekt* der Klasse eingebaut.

Objektinitialisierer (engl. instance initializer) (o): Eine *Block*anweisung, die direkt in einer Klasse steht und nicht mit `static` gekennzeichnet ist. Wird jedes Mal ausgeführt, wenn die Klasse instanziiert wird, und zwar noch vor dem betreffenden Konstruktor. Der Programmierer kann z. B. komplizierte Befehle zum Initialisieren von Objektattributen (*Schleifen*, Zugriffe auf Dateien und Datenbanken etc.) in solche Objektinitialisierer schreiben (statt in die Konstruktoren).

Objektmethode (engl. instance method) (a, s): Eine nicht mit `static` gekennzeichnete *Methode* einer *Klasse*. Gehört zum Bauplanaspekt der Klasse und wird in jedes *Objekt* der Klasse eingebaut.

observer (a): siehe *Behandler-Objekt*.

Operand (a): Im weiteren Sinne ist ein Operand jeweils „das, worauf ein Befehl angewendet wird". Im engeren Sinne sind damit die aktuellen Parameter einer *Operation* gemeint. Z. B. besteht der Ausdruck `x + y` aus dem *Operator* `+` und den zwei Operanden `x` und `y`. Der Ausdruck `-(x + y)` besteht aus dem Operator `-` und dem einen Operanden `(x + y)`.

Operation (a, s): Im weiteren Sinne ist damit „irgendein Befehl" gemeint. Im engeren Sinne ist eine Operation eine *Funktion*, mit einem *Operator* als Namen.

Operator (a, s): Ein Name von *Operationen*. In Java bestehen die meisten Operatoren aus ein bis vier Sonderzeichen (z. B. `+`, `!=`, `>>>` und `>>>=`). Die Operatoren der einstelligen Operationen werden entweder *präfix* oder *postfix* (vor bzw. hinter ihrem einzigen Operanden) notiert. Die Operatoren (Namen) der meisten zweistelligen Operationen werden *infix* (zwischen ihren beiden Operanden) notiert.

overload (a method name) (a): siehe *überladen*.

override (an object method) (a): siehe *überschreiben*.

package-Befehl (o): Ist nur als erster Befehl in einer Quelldatei erlaubt. Legt fest, zu welchem Paket die Klassen und Schnittstellen gehören, die in dieser Quelldatei vereinbart werden. Beispiel: Die Befehle `package petra.test.versuch1; class Otto {...}` bewirken, dass die Klasse `Otto` zum Paket mit dem *vollen Namen* `petra.test.versuch1` gehört.

Paket (engl. package) (o): In Java (nicht bei der Post): Ein Behälter für *Klassen*, *Schnittstellen* und *Pakete*. Ein Paket ist kein richtiger *Modul*, weil man nur die Klassen und Schnittstellen, aber nicht die Pakete in einem Paket vor Zugriffen schützen kann.

Parameter (einer generischen Einheit, a): Ein Name für einen Typ. Wird in Java in spitze Klammern eingeschlossen vereinbart, z. B. `class Paar<T> {...}`. Dieser Name `T` darf innerhalb der generischen Einheit wie der Name eines Typs verwendet werden (mit ein paar Einschränkungen).

Parameter (eines Unterprogramms, a): Dienen dazu, dem Unterprogramm Werte zu übergeben, wenn man es aufruft. In der Vereinbarung des Unterprogramms vereinbart man *formale Parameter* (ganz ähnlich wie Variablen). In einem Aufruf des Unterprogramms muss man für jeden formalen Parameter einen Ausdruck (als *aktuellen Parameter*) angeben. Siehe auch *Parameterübergabe*.

Parameterübergabe per Wert (a): In einem Aufruf eines Unterprogramms U muss man für jeden formalen *Parameter*, mit dem U vereinbart wurde, einen entsprechenden *Ausdruck* als aktuellen *Parameter* angeben. Die formalen Parameter von U werden als Variablen erzeugt und mit den Werten der aktuellen Parameter initialisiert.

Parameterübergabe per Referenz (a): Gibt es z. B. in Pascal und C#, aber nicht in Java. Die Übergabe einer Referenzvariablen per Wert hat aber praktisch den gleichen Effekt wie eine Übergabe des betreffenden Objekts per Referenz. Dass man primitive Werte und Referenzwerte in Java nicht per Referenzwerte übergeben kann, ist einerseits ein Mangel, macht aber andererseits die Sprache einfacher.

parametrisierter Typ (a): Eine generische Klasse oder Schnittstelle (z. B. `ArrayList` oder `Set`), bei der man für jeden Parameter einen konkreten Typ festgelegt hat (z. B. `ArrayList<String>`, `ArrayList<Integer>`, `Set<String>`, `Map<String, Integer>` etc.).

pass by reference (a): siehe *Parameterübergabe per Referenz*.

pass by value (a): siehe *Parameterübergabe per Wert*.

per Referenz (a): siehe *Parameterübergabe per Referenz*.

per Wert (a): siehe *Parameterübergabe per Wert*.

Plattform (a): Eine Kombination aus einer bestimmten Computer-Hardware und einem Betriebssystem, z. B. ein PC unter Linux oder ein PC unter Windows oder ein Mac unter dem MacOS oder ein Mac unter Linux etc.

Postfixnotation (a): Ein neuzeitlicher Brauch, *Operatoren* **hinter** alle ihre *Operanden* zu schreiben. Die Postxfixnotation ist (anders als die *Infixnotation*) auch ohne zusätzliche Konventionen eindeutig. Diesen Vorteil verliert sie nur, wenn man sie mit anderen Notationen (z. B. mit der *Infixnotation*) mischt, wie es in Java der Fall ist.

Präfixnotation (a): Ein neuzeitlicher Brauch, *Operatoren* **vor** alle ihre *Operanden* zu schreiben. Die Präfixnotation ist (anders als die *Infixnotation*) auch ohne zusätzliche Konventionen eindeutig. Diesen Vorteil verliert sie nur, wenn man sie mit anderen Notationen (z. B. mit der *Infixnotation*) mischt, wie es in Java der Fall ist.

primitiver Typ (a): In einer Kneipe: Ein Mensch, der sich nicht über die Feinheiten von Programmiersprachen unterhalten mag. In Java: Einer der acht Typen `byte`, `char`, `short`, `int`, `long`, `float`, `double`, `boolean`.

Profil (einer Methode, engl. profile) (a): Besteht aus dem *Rückgabetyp* der Methode gefolgt von ihrer *Signatur*. Beispiel: Die `main`-Methode eines Java-Programms muss das Profil `void main String[]` haben.

Programm (a, s): Eine Folge von Befehlen, die ein *Programmierer* aufgeschrieben hat und die von einem *Ausführer* ausgeführt werden kann.

Programmierer (s): Eine Rolle im Rollenspiel des Programmierens. Der Programmierer schreibt Programme und übergibt sie dem *Ausführer*.

Proma (s): Ein Programm mit mehreren Ausführern. Damit ist ein Java-Programm gemeint, in dem Faden-Objekte erzeugt und gestartet werden. Jeder Faden wirkt wie ein selbständiger sequentieller

Ausführer, der nebenläufig zu den anderen Fäden/Ausführern bestimmte Befehle des Programms ausführt.

Prozedur (a): Ein *Unterprogramm*, welches einen Seiteneffekt hat (d. h. die Inhalte bestimmter Wertebehälter verändert), aber keinen *Wert* liefert. Gegensatz: *Funktion*. Prozeduren dienen dazu, die Inhalte von *Wertebehältern* zu verändern. Ein Aufruf einer Prozedur ist eine *Anweisung*.

prozedurale Programmiersprache (engl. procedural programming language) (a): Eine Programmiersprache, bei der es als Unterprogramme nicht nur *Funktionen*, sondern auch *Prozeduren* gibt, die veränderbare *Variablen* enthält und außer *Vereinbarungen* und *Ausdrücken* auch *Anweisungen* (insbesondere die *Zuweisungsanweisung*) kennt. Die meisten verbreiteten Programmiersprachen sind prozedural. Gegensatz: *funktionale Programmiersprache*.

Prozess (eines Betriebssystems) (a): Eine Verwaltungseinheit eines Betriebssystems. Einem Prozess können Betriebsmittel zugeordnet werden (z. B. Hauptspeicherbereiche, Geräte wie Drucker und Plattenlaufwerke, Dateien etc.). Ein Programm kann nur im Rahmen eines Prozesses ausgeführt werden. Im Rahmen eines Prozesses können nacheinander (sequentiell) mehrere Programme ausgeführt werden. Mehrere Prozesse werden vom Betriebssystem nebenläufig zueinander (*concurrently*, d. h. zeitlich unabhängig voneinander) ausgeführt. Heute noch übliche Computer mit nur einem Hauptprozessor führen Prozesse „stückchenweise abwechselnd“ aus. Um dabei von einem Prozess zu einem anderen umzuschalten müssen relativ viele Maschinenbefehle ausgeführt werden („ein Prozesswechsel ist teuer“). Siehe auch *Faden*.

Quellcode-Interpreter (source code interpreter) (a): Ein Programm, welches Quelldateien (einlesen und) ausführen kann, ohne dass der Benutzer sie vorher übersetzen lassen muss. Das Programm BeanShell (siehe `www.beanshell.org`) ist ein Quellcode-Interpreter für Java-Programme.

Referenz (a): Jede Variable besteht (mind.) aus einer Referenz und einem Wert. Speziell bei Referenzvariablen ist auch der Wert ein Referenzwert. Eine Referenzvariable kann den Wert `null` haben, aber `null` ist garantiert nicht die Referenz einer Variablen.

Referenztyp (a): Ein Klassen-, Schnittstellen- oder Reihungstyp. *Aufzählungstypen* und *parametrisierte Typen* sind Klassentypen und somit auch Referenztypen.

Referenzvariable (a): Eine *Variable* eines *Referenztyps*. Oder: Eine *Variable*, deren *Wert* eine Referenz ist. Wenn dieser Wert ungleich *null* ist, zeigt die Referenzvariable auf einen *Zielwert*.

Reflexion (a, s): Ein paar spezielle Klassen (namens `Class`, `Method`, `Field`, `Constructor` etc.) ermöglichen es einem Java-Programm, während seiner Ausführung Klassen, Methoden, Attribute etc. (insbesondere seine eigenen) zu untersuchen und zu manipulieren. Damit ist es z. B. möglich, in einem Java-Programm P Methoden einer Klasse aufzurufen, die noch gar nicht existierte, als P geschrieben wurde. Diese Möglichkeit ist vor allem für Werkzeuge wichtig, mit denen Java-Programme verwaltet (compiliert, ausgeführt, analysiert etc.) werden sollen.

Reihung (engl. array) (o): Ein Objekt, welches aus einer festen Anzahl von Variablen besteht, die alle zum selben Typ gehören. Die Variablen bezeichnet man auch als die *Komponenten* der Reihung.

Reihungstyp (engl. array type) (o): Ein Typ, dessen Objekte *Reihungen* sind. Beispiele für Reihungstypen: `String[]`, `int[]`, `float[][]`. Einen Reihungstyp kann man nicht *erweitern*.

rekursiv (a): Eine *Methode* (ein *Unterprogramm*, eine *Funktion*, eine *Prozedur*) ist rekursiv, wenn sie sich (direkt oder indirekt) selbst aufruft.

return: eine *einfache Anweisung*. Dient zum Beenden eines *Unterprogramms*.

roher Typ (engl. raw type) (a): Eine *generische Klasse* oder *Schnittstelle*, die man als Typ verwendet, ohne für ihre Parameter konkrete Typen festzulegen. Kommt vor allem in älteren, vorgenerischen Java-Programmen und als Verbindung mit solchen Programmen in neueren Programmen vor.

Rückgabetyp (einer Methode, engl. result type, return type) (a): Siehe *Ergebnistyp*.

Rumpf (engl. body) (a): *Klassen*, *Methoden*, und *zusammengesetzte Anweisungen* (*if*, *switch*, *while*, *-do-while* und *for*) haben einen Rumpf. Eine `if`-Anweisung kann zwei Rümpfe haben (einen dann-Rumpf und einen sonst-Rumpf). Der Rumpf einer Klasse oder Methode ist eine

Blockanweisung, der Rumpf einer zusammengesetzten Anweisung eine beliebige Anweisung (z. B. eine Blockanweisung).

Sammlung (engl. collection) (o): Ein Objekt einer *Sammlungsklasse*. In einer Sammlung kann man Objekte einfügen, löschen und suchen, kurz: sammeln.

Sammlungsklasse (engl. Collection class) (o): In Java: Eine Klasse, die die Schnittstelle `Collection` implementiert.

Schleife (engl. loop) (a): Eine *zusammengesetzte Anweisung* mit der man bewirken kann, dass die darin enthaltene Anweisung mehr als einmal ausgeführt wird (*Wiederholungsanweisung*). Siehe auch *for*, *do-while* und *while*.

Schnittstelle (engl. interface) (o, s): Eine Schnittstelle ist eine Art „total *abstrakte Klasse*“. Sie darf nur *abstrakte Methoden* (Objektmethoden) und unveränderbare (`final`) *Klassenattribute* enthalten. Eine Schnittstelle darf beliebig viele (0, 1, 2, 3, ...) andere Schnittstellen erweitern (beerben). Eine *Klasse* darf beliebig viele Schnittstellen implementieren.

semantische Größe (a, s): Eine Größe, die während der Ausführung eines Programms erzeugt oder berechnet wird, z. B. eine Methode (wird auf Grund einer Vereinbarung erzeugt) oder der Wert eines Ausdrucks (wird berechnet). Gegensatz: *syntaktische Größe*.

Shell (unter Linux) (a): siehe *Konsole*.

Signatur (einer Methode) (a): Besteht aus dem Namen der Methode gefolgt von den Namen der Typen ihrer Parameter. Beispiel: Die `main`-Methode eines Java-Programms muss die Signatur `main String[]` haben.

simple expression (a): siehe *einfacher Ausdruck*.

simple statement (a): siehe *einfache Anweisung*.

SPACE (o): Das Unicode-Zeichen mit dem Code `u0020` (wird auch als Leerzeichen oder Blank bezeichnet. Ist bei den meisten Tastaturen mit der breitesten Taste verbunden). Sollte nicht mit einem *leeren String* verwechselt werden.

Standardkonstruktor (a): Ein *Konstruktor* mit 0 Parametern. Wenn der Programmierer innerhalb einer Klasse keinen Konstruktor vereinbart, „schenkt“ der Ausführer dieser Klasse einen Standardkonstruktor mit leerem Rumpf.

statement (a): siehe *Anweisung*.

static (o): Mit diesem Schlüsselwort werden die *Klassenelemente* einer Klasse gekennzeichnet (um sie von den *Objektelementen* der Klasse zu unterscheiden).

static import (a): Ein Java-Befehl, mit dem man Abkürzungen für die Namen von Klassenelementen (static members) einer Klasse einführen kann. Beispiel: Nach dem Befehl `import static java.lang.Math.max;` darf man das Klassenelement `max` der Klasse `Math` auch mit seinem einfachen Namen `max` statt mit seinem vollen Namen `java.lang.Math.max` (oder dem abgekürzten Namen `Math.max`) bezeichnen. Nach dem Befehl `import static java.lang.Math.*;` darf man alle Klassenelemente der Klasse `Math` mit ihren einfachen Namen (`max`, `min`, `sin`, `cos` etc.) bezeichnen. Diese Variante des `import`-Befehls gibt es erst seit Java 5.0.

Stelligkeit (einer *Operation*) (a): Anzahl der Operanden, auf die eine *Operation* angewendet werden darf/muss. Beispiel: Alle Multiplikationsoperationen namens `*` sind zweistellig, alle Vorzeichenoperationen namens `-` sind einstellig, alle Subtraktionsoperationen namens `-` sind zweistellig etc. In Java sind alle Operationen ein- oder zweistellig, nur die bedingte Operation `...?...:...` ist dreistellig. Häufig überträgt man die Eigenschaft der Stelligkeit von einer *Operation* auf ihren Namen, d. h. auf ihren *Operator*. Dann muss man aber sorgfältig z. B. zwischen dem einstelligen Operator `-` (dem Vorzeichen Minus) und dem zweistelligen Operator `-` (dem Subtraktionsoperator) unterscheiden.

Steuerfaden (engl. thread of control) (o, s): siehe *Faden*.

Strohpuppe (s): Bei Dorffesten: Eine beliebte Dekoration aus getrockneten Getreidehalmen. In Java-Programmen: Eine Methode mit einem leeren Rumpf (dient meistens zum Implementieren einer *Schnittstelle*, die mehrere Methoden enthält).

Strom (engl. stream) (o, s): Ein Objekt einer Stromklasse. Ein Strom-Objekt verbindet ein Java-Programm mit einer Datenquelle (von der man Daten einlesen kann) bzw. mit einer Datensenke (zu der man Daten ausgeben kann).

Stromklasse (engl. stream class) (o, s): In Java gibt es mehr als 40 Stromklassen. Die meisten gehören zum Paket `java.io` und sind Unterklassen der vier abstrakten Klassen `InputStream`, `OutputStream`, `Reader` bzw. `Writer`.

subclass (a): siehe *Unterklasse*.

super (als Name der direkten Oberklasse) (o): Angenommen, eine Klasse `U` hat von ihrer direkten Oberklasse `DO` ein Element `eg` namens `otto` geerbt und durch ein (in U vereinbartes) Element `ev` namens `otto` ersetzt (d. h. *überschrieben* oder *verdeckt*), dann bezeichnet innerhalb von `U` der einfache Name `otto` das in `U` vereinbarte Element `ev` und der zusammengesetzte Name `super.otto` das geerbte und ersetzte Element `eg`.

super (als Name eines Konstruktors) (o): Wenn man am Anfang eines Konstruktors einer Klasse `U` einen Konstruktor der direkten Oberklasse `DO` von `U` aufrufen will, muss man anstelle seines richtigen Namens das Schlüsselwort `super` angeben. Ein solcher `super-`Aufruf ist nur als erster Befehl eines Konstruktors erlaubt.

superclass (a): siehe *Oberklasse*.

switch (a): Eine *zusammengesetzte Anweisung*, eine *Fallunterscheidungsanweisung*.

syntaktische Größe (a, s): Eine Größe, die im Quelltext eines Programms vorkommt, z. B. ein *Literal* oder eine *Vereinbarung*. Gegensatz: *semantische Größe*. Viele Begriffe bezeichnen gleichzeitig eine syntaktische und eine eng damit zusammenhängende semantische Größe. Mit „Methode“ ist manchmal die Vereinbarung einer Methode im Quelltext eines Programms gemeint (eine syntaktische Größe) und manchmal die Methode, die der Ausführer bei einer Ausführung des Programms daraus erzeugt (eine semantische Größe). Ähnliches gilt auch für Variablen, Klassen und weitere Größen. Dagegen bezeichnet der Begriff *Literal* nur eine syntaktische Größe und der Begriff *Wert* nur eine semantische Größe.

this (als Name einer Referenzvariablen) (o): Jedes Objekt `ob` enthält eine unveränderbare Referenzvariable namens `this`, die auf `ob` zeigt. Innerhalb einer Methode mit einem Parameter namens `otto` bezeichnet der einfache Name `otto` den Parameter und der zusammengesetzte Name `this.otto` ein Attribut des umgebenden Objekts (falls das Objekt ein Attribut namens `otto` besitzt).

this (als Name eines Konstruktors) (o): Wenn man innerhalb eines Konstruktors einer Klasse K einen anderen Konstruktor derselben Klasse K aufrufen will, muss man anstelle seines richtigen Namens das Schlüsselwort `this` angeben. Ein solcher `this-`Aufruf ist nur als erster Befehl eines Konstruktors erlaubt.

thread (o): siehe *Faden*, *nebenläufig*, *Prozess*.

throw (a): eine *einfache Anweisung*. Dient dazu *Ausnahmen* zu werfen. Beispiel:
`throw new ArithmeticException("Konto ist negativ!");`.

throws (a): Am Anfang einer Methodenvereinbarung müssen nach dem Schlüsselwort `throws` die Klassennamen aller *geprüften Ausnahmen* aufgeführt werden, die in dieser Methode auftreten können und nicht (mit einem `try-catch`-Befehl) gefangen-und-behandelt werden.

Top-Paket (o): Ein *Paket*, welches in keinem anderen Paket enthalten ist. Die Java-Standardbibliothek enthält drei Top-Pakete namens `java`, `javax` und `org`. Der Programmierer kann weitere Top-Pakete erzeugen lassen.

Typ (a): Eine Menge von *Werten* und eine Menge von *Befehlen*, die man auf die Werte anwenden darf.

Typ (s): Ein Bauplan für *Variablen*.

Typumwandlung (engl. type conversion) (a): Eine Berechnung, die auf Grund eines *Cast-Befehls* oder von einer speziellen Funktion durchgeführt wird. Dabei wird aus einem Wert `w1` eines Typs `t1` ein entsprechender Werte `w2` eines anderen Typs `t2` berechnet. Beispiel: Der Cast-Befehl `(int) 17.85` berechnet aus dem `double`-Wert `17.85` den `int`-Wert `17`. Der Funktionsaufruf `Integer.parseInt("17")` berechnet aus dem String `"17"` den `int`-Wert `17`.

übergeben (s): Eine Tätigkeit des *Programmierers*. Java-Programme übergibt man dem Ausführer heutzutage häufig, indem man sie compilieren läßt.

überladen (einen Methodennamen, engl. overload) (a): Ein Methodenname ist überladen, wenn er mehrere *Methoden* mit unterschiedlichen *Signaturen* bezeichnet. Einen Methodennamen kann man überladen, indem man z. B. innerhalb einer Klasse mehrere Methoden mit demselben Namen, aber unterschiedlichen Parametertypen vereinbart. Es ist nicht erlaubt, innerhalb einer Klasse mehrere Methoden mit gleichen Signaturen zu vereinbaren.

überschreiben (eine Objektmethode, engl. override) (a): Ein Spezialfall von *ersetzen*. Angenommen, eine Unterklasse U erbt von ihrer direkten Oberklasse eine Objektmethode `m1` mit einer Signatur `s1`. Wenn man in `U` eine Objektmethode `m2` mit der gleichen Signatur `s1` vereinbart, dann überschreibt die Methode `m2` die Methode `m1`. Wenn man in U eine Objektmethode `m3` mit dem gleichen Namen wie `m1`, aber einer anderen Signatur als `m1` vereinbart, wird nur der Name von `m1` und `m3` *überladen*, aber die Methode `m1` wird durch `m3` nicht überschrieben.

UCS (o): Abkürzung für: Universal Multiple-Octet Coded Character Set. Siehe auch *UTF-8* und *Unicode*.

unchecked exception (o): siehe *ungeprüfte Ausnahme*.

ungeprüfte Ausnahme (engl. unchecked exception) (o): Ein Objekt der Klasse `Error` oder einer Unterklasse. Kann, muss aber nicht am Anfang einer Methodenvereinbarung nach `throws` erwähnt (und damit dokumentiert) werden.

ungetypte Variable (a): siehe *getypte/ungetypte Variable*.

Unicode (o): Der Unicode umfasst etwa 60 Tausend 16-Bit-Codezahlen für häufig verwendete Zeichen und ungefähr 1 Million 32-Bit-Codezahlen für weniger häufig verwendete Zeichen (genau sind es 63.488 16-Bit.-Codezahlen und 1.048.576 32-Bit-Codezahlen). Viele der 16-Bit-Codezahlen sind schon bestimmten Zeichen zugeordnet (darunter allen lateinischen, griechischen russischen, koreanischen, vietnamesischen etc. Buchstaben und mehr als 27.000 chinesischen Schriftzeichen). Die meisten der 32-Bit-Codezahlen sind noch frei für zukünftige Anwendungen. Siehe auch *UTF-8*.

Unterklasse (engl. subclass) (o): Seien `DU1`, `DU2`, ... etc. *direkte Unterklassen* einer Klasse `O`. Dann sind `DU1`, `DU2`, ... etc. und alle Unterklassen von `DU1` und alle Unterklassen von `DU2` ... etc. Unterklassen von `O`.

Unterprogramm (engl. subprogram) (a): Eine vom Programmierer zusammengefasste und mit einem Namen und (0 oder mehr) formalen Parametern versehene Befehlsfolge. Der Programmierer muss ein Unterprogramm einmal vereinbaren und darf es dann beliebig oft aufrufen. Siehe auch *Methode*, *Funktion*, *Prozedur*.

unveränderbare Variable (a): Eine Variable, deren Wert nicht verändert werden kann. In Java: Eine als `final` vereinbarte Variable. Wird manchmal (aber nicht immer) von einer *Konstanten* (die keine Referenz oder Adresse hat) unterschieden.

Unzahlen (engl. NaN) (s): Viele Werte der Typen `float` und `double` stellen keine Zahlen dar und werden hier als Unzahlen bezeichnet. Als Ergebnis einer Berechnung wird immer dann eine Unzahl geliefert, wenn das Ergebnis überdefiniert ist, d. h. nach mehreren vernünftigen, aber sich

widersprechenden Regeln berechnet werden könnte. Beispiel: Was ist `0.0/0.0` ? Regel 1: `0.0/irgendeine-Zahl` ist gleich `0.0`. Regel 2: `irgendeine-positive-Zahl/0.0` ist gleich `infinity`. Regel 3: `x/x` ist gleich `1`. Diese drei Regeln legen für die Berechnung von `0.0/0.0` also die drei Ergebnisse `0`, `infinity` und `1` nahe. Der Ausführer entscheidet sich nicht willkürlich für eines davon, sondern liefert eine Unzahl (einen *NaN*-Wert) als Ergebnis. Welche Unzahl er liefert, wird durch die Sprache Java nicht festgelegt.

user (a): siehe *Benutzer*.

UTF-8 (o): Abkürzung für: *UCS* Transformation Format, 8-Bit form. Der UTF-8-Code ordnet exakt denselben Zeichen Codezahlen zu wie der Unicode, aber andere Codezahlen. Die UTF-8-Codezahlen haben unterschiedliche Längen (8, 16, 24 bzw. 32 Bits). Etwas vereinfacht gesagt gilt: Je häufiger ein Zeichen in westlichen Ländern verwendet wird, desto kürzer ist seine UTF-8-Codezahl. Die 128 ASCII-Zeichen haben 8-Bit-Codezahlen (die selben wie im ASCII-Code), Zeichen wie ü, ö, ä etc. haben 16-Bit-Codezahlen und chinesische Zeichen haben 24-Bit und 32-Bit Codezahlen. Bei einem Text, der nur aus den 128 ASCII-Zeichen besteht, belegt jedes Zeichen also nur 1 Byte. Dagegen belegt bei einem rein chinesischen Text jedes Zeichen bis zu 4 Byte. Der Platzbedarf für andere im UTF-8-Code gespeicherte Texte liegt irgendwo zwischen diesen Extremen.

value (a): siehe *Wert*.

Variable (a, s): Ein Behälter für *Werte*, dessen Inhalt vom *Ausführer* beliebig oft verändert werden kann. Eine Variable enthält immer genau einen *Wert* (sie kann also nicht leer sein und nicht mehr als einen Wert enthalten). Wenn man einer Variablen einen neuen Wert zuweist, wird der alte Wert zerstört. Eine Variable besteht aus mindestens zwei Teilen, einer *Referenz* und einem *Wert*. Zusätzlich kann eine Variable einen Namen und/oder einen *Zielwert* haben. Die Referenz einer Variablen ist unveränderbar, der Wert kann (z. B. durch Zuweisungen) beliebig oft verändert werden. Der Name einer Variablen wird vom *Programmierer*, die Referenz vom *Ausführer* festgelegt. Der Ausführer legt für keine Variable *null* als Referenz fest und garantiert, dass zwei verschiedene Variablen verschiedene Referenzen haben. *Referenzvariablen* können aber den Wert *null* und zwei verschiedene Variablen können gleiche *Werte* haben. Siehe auch *getypte/ungetypte Variable*.

verdecken (ein Attribut, engl. to hide) (a): Ein Spezialfall von *ersetzen*. Angenommen, eine Unterklasse `U` erbt von ihrer direkten Oberklasse ein Attribut `a1` mit dem Namen `n1`. Wenn man in `U` ein Attribut `a2` mit dem gleichen Namen `n1` vereinbart, dann verdeckt das Attribut `a2` das Attribut `a1`.

Vereinbarung (engl. declaration) (a, s): Ein Befehl des *Programmierers* an den *Ausführer*, etwas zu erzeugen, z. B. eine *Variable*, ein *Unterprogramm*, eine *Klasse* oder eine andere Größe. *Werte* werden nicht vereinbart und erzeugt, sondern berechnet.

vererben (an eine Unterklasse, engl. to let, to bequeath) (a, s): Wenn eine Klasse UK eine Klasse OK *erweitert*, vererbt OK all ihre Elemente an UK. Jede Klasse OK darf ihre Elemente an beliebig viele Unterklassen UK vererben (das gilt für alle objektorientierten Programmiersprachen). Konstruktoren werden nicht vererbt.

Verklemmung (eines Systems von nebenläufigen Einheiten, engl. deadlock) (a): Angenommen, zwei nebenläufigen Einheiten (Fäden oder Prozesse) `NE1` und `NE2` brauchen beide alleinigen Zugriff auf zwei Betriebsmittel `BMA` und `BMB` und versuchen, diese für sich zu reservieren. Wenn es dann der Einheit `NE1` gelingt, das Betriebsmittel `BMA` für sich zu reservieren und es `NE2` gelingt, `BMB` für sich zu reservieren, liegt eine Verklemmung vor. In einer solchen Situation kann keine der nebenläufigen Einheiten weiterlaufen. Es gibt viele verschiedene Strategien zur Vermeidung solcher Verklemmungen und einige zu ihrer Auflösung, nachdem sie eingetreten sind.

Verwender (s): Eine Rolle im Rollenspiel des Programmierens. Der Verwender schreibt (ähnlich wie der *Programmierer*) Programme. Dabei möchte er Programme und Teile von Programmen, die der Programmierer schon geschrieben hat, wiederverwenden

void (o): Ein Typ, zu dem keine Werte gehören. Wird manchmal auch als Pseudotyp bezeichnet. Muss beim Vereinbaren einer *Prozedur* anstelle eines richtigen *Ergebnistyps* angegeben werden.

virtueller Speicher (a): Schnelle Speicherelemente (z. B. RAM-Speicher-Chips) sind relativ teuer, billige Speicherelemente (z. B. Festplatten) sind relativ langsam. Ein virtueller Speicher ist eine kostengünstige Kombination aus einem kleinen schnellen und einem großen langsamen Speicher, die dem Benutzer wie ein großer schneller Speicher erscheint. Dazu wird mit Hilfe raffinierter Algorithmen dafür gesorgt, dass Daten fast immer im schnellen Speicher stehen, wenn sie benötigt werden.

voller Name (einer Klasse, engl. full name of a class) (o): Falls eine Klasse `K` zu einem Paket-mit-Namen `p` gehört, besteht ihr voller Name aus dem vollen Namen von p gefolgt von einem Punkt `'.'` und dem Namen von `K`. Falls `K` zum namenlosen Paket gehört, ist der volle Name von `K` gleich dem Namen von `K`. Beispiel: Die Standardklasse `ArrayList` hat den vollen Namen `java.util.ArrayList`.

voller Name (eines Paketes, engl. full name of a package): Der volle Name eines *Top-Paketes* besteht nur aus dem Namen des Paketes. Der volle Name eines Paketes `P5`, welches in einem Paket `P4` enthalten ist, besteht aus dem vollen Namen von `P4`, gefolgt von einem Punkt `'.'` gefolgt vom Namen von `P5`. Beispiel: Das Standardpaket `renderable` hat den vollen Namen `java.awt.image.renderable`.

Warter (engl. maintainer) (s): Eine Rolle im Rollenspiel des Programmierens. Der Warter wartet die Programme, die der *Programmierer* geschrieben hat, d. h. er korrigiert und erweitert sie oder passt sie neuen Anforderungen des *Benutzers* oder des *Ausführers* an.

Wert (engl. value) (a): Ein Ding, welches während der Ausführung eines Programms berechnet und eventuell in einer Variablen gespeichert oder in einen anderen Wertebehälter getan wird. In einem Programm werden Werte durch *Ausdrücke* beschrieben. Eine Variable besteht mindestens aus einer Referenz und einem Wert. Siehe auch *Literal* und *Konstante*.

Wertebehälter: Eine *Variable* oder ein Ein-/Ausgabegerät (z. B. eine Tastatur, ein Bildschirm, ein Drucker, eine Datei auf einer Festplatte etc.).

while: eine *zusammengesetzte Anweisung*, eine *Wiederholungsanweisung* (*Schleife*).

Wiederholungsanweisung (a): Eine *zusammengesetzte Anweisung* mit der man bewirken kann, dass die darin enthaltene Anweisung mehr als einmal ausgeführt wird (*Schleife*). Siehe auch *for*, *do-while* und *while*.

wrapper class (a): siehe *Hüllklasse*.

zeichenorientierter Strom (engl. character stream) (o, s): Dient zum Einlesen und Ausgeben von Texten, die aus Zeichen bestehen. Beim Ausgeben müssen die Zeichen aus dem internen *Unicode* in Bytefolgen eines externen Codes (z. B. *ASCII* oder *UTF-8* etc.) umgewandelt werden. Beim Einlesen müssen umgekehrt Zeichen eines externen Codes in interne Unicode-Zeichen umgewandelt werden.

Zeilenende-Markierung (a): Wie in einer Textdatei das Ende von Zeilen markiert wird, ist plattformabhängig. Eine Zeilenende-Markierung besteht unter einem Unix-Betriebssystem (z. B. Linux) aus einem *LF-Zeichen*, unter einem Mac-Betriebssystem aus einem *CR-Zeichen* und unter einem Windows-Betriebssystem aus einem *CR-Zeichen* gefolgt von einem *LF-Zeichen*.

Zielwert (engl. target vlaue) (s): Der *Wert*, auf den eine Referenzvariable zeigt. In Java ist ein Zielwert immer ein Objekt. Eine Referenzvariable mit dem Wert *null* zeigt nicht auf einen Zielwert.

zusammengesetzte Anweisung (compound statement) (a): Eine *Anweisung*, die andere Anweisungen enthält. Beispiel: Die zusammengesetzte Anweisung `if (a<b) a = 2*b;` enthält die Anweisung `a = 2*b;`. Mit einer zusammengesetzten Anweisung kann man bewirken, dass die darin enthaltenen Anweisungen weniger als einmal (also nicht) oder mehr als einmal ausgeführt werden. Siehe auch *Fallunterscheidungsanweisung* und *Wiederholungsanweisung*.

zusammengesetzter Ausdruck (conpound expression) (a): Ein *Ausdruck*, der andere Ausdrücke enthält. Beispiel: Der zusammengesetzte Ausdruck `a + b * c` enthält den *einfachen Ausdruck* `a` und den zusammengesetzten Ausdruck `b * c`.

Zusicherung (eine assert-Anweisung, engl. assertion) (a): Eine Anweisung zum Testen und Überprüfen eines Programms. Besteht im Wesentlichen aus einer Bedingung. Wenn die nicht erfüllt ist (und Zusicherungen aktiviert sind), wird eine Ausnahme geworfen.

zweistellig (a): siehe *Stelligkeit*.

Zwiebel-Darstellung (s): Eine Darstellung von Objekten die hervorhebt, dass jedes Objekt einer Unterklasse ein Objekt ihrer direkten Oberklasse enthält.