

Inhaltsverzeichnis

Grundlagen und Einführung (1. Band)	1
1 Einleitung und Vorwort	1
1.1 Vorwort zur 10. Auflage	1
1.2 Voraussetzungen	2
1.3 Layoutkonventionen	3
1.4 Dokumenten-Erstellung und Online-Version	4
1.5 Dank	5
1.6 Autor	5
2 Java im Überblick	7
2.1 Insel, Kaffee und Programmiersprache	7
2.2 Programmiersprache Java	8
2.3 Java-Bytecode und die Virtuelle Maschine	12
2.4 Anwendungsbereiche und Einsatzgebiete	13
2.5 Entwicklung von Java	15
2.6 Java, JavaScript und JavaFX	16
2.7 Java und OpenSource	17
2.8 Bewertung und Fazit	17
3 Informationsdarstellung im Rechner	19
3.1 Zahldarstellung in Computern	19
3.2 Bit-Operationen auf Binärzahlen	20
3.3 Externe Zahldarstellung in Java	22
3.4 Der Unicode-Zeichensatz	22
3.5 Unicode-Darstellungsformate	23
3.6 Java und Unicode	26
3.7 Steuer- und Sonderzeichen	27
4 Hallo Java - Erste Programmierschritte	29
4.1 Das Erste Java-Programm	30
4.1.1 Hello World als Applikation	30
4.1.2 "Hello World" als Swing-Applikation	31
4.1.3 "Hello World" als Applet	32
4.1.4 Elemente von Java-Dateien	33
4.1.5 Leerzeichen, Zeilenumbrüche, Einrückungen	33
4.1.6 Kommentare	35
4.1.7 Groß-/Kleinschreibung	36
4.1.8 Bestandteile einer Java-Datei	36

4.1.9	Die package-Anweisung	37
4.1.10	Die import-Anweisung	37
4.1.11	Die class-Anweisung	38
4.1.12	Die Methoden main und paint	38
4.1.13	Die HTML-Datei	39
4.2	Java-Programme übersetzen und ausführen	39
4.3	Der Java-Compiler javac	40
4.4	Die Java-Laufzeitumgebungen java und appletviewer	40
5	Variablen, Werte und Referenzen	43
5.1	Bezeichner	43
5.2	Schlüsselwörter	45
5.3	Variablen	46
5.3.1	Deklaration und Grundeigenschaften	46
5.3.2	Wertzuweisung und Typkompatibilität	48
5.3.3	Initialisierung lokaler Variablen	48
5.3.4	Sichtbarkeit und Lebensdauer	49
5.3.5	Wert- und Referenztypen	50
6	Wertetypen und ihre Operationen	53
6.1	Wahrheitswerte	53
6.2	Zeichenwerte	54
6.2.1	Ganzzahlwerte	54
6.2.2	Gleitkommatypen	55
6.2.3	Typkonvertierungen bei Wertetypen	56
6.3	Ausdrücke und Operatoren	57
6.3.1	Eigenschaften von Ausdrücken und Operatoren	57
6.3.2	Arithmetische Operatoren	58
6.3.3	Relationale Operatoren	60
6.3.4	Logische Operatoren	61
6.3.5	Bit-Operatoren	63
6.3.6	Zuweisungsoperatoren	63
6.3.7	Sonstige Operatoren	63
6.3.8	Priorität der Operatoren	65
7	Referenztypen (Klassen)	67
7.1	Object - Der Basisreferenztyp	67
7.2	Arrays - <i>Überblick und Eigenschaften</i>	68
7.2.1	Deklaration und Anlegen von Arrays	71
7.2.2	Beispiele zu Arrays	72
7.2.3	Arrays von Arrays	72
7.2.4	Arrays kopieren	73
7.2.5	Die Klasse Arrays	74
7.3	Strings - Zeichenketten und Ihre Methoden	75
7.3.1	Methoden der Klasse String	75
7.3.2	Beispiele zu String-Operationen	76
7.3.3	Die Methode "format"	76
7.4	Wrapper-Klassen	77
7.4.1	Autoboxing	79
7.5	Zuweisung von Referenztypen	80

8	Kontrollstrukturen	83
8.1	Anweisungen und Blöcke	83
8.2	Kontrollstrukturen	84
8.2.1	if und if-else	84
8.2.2	switch-Anweisung	87
8.2.3	while-Schleife	88
8.2.4	do-Schleife	89
8.2.5	for-Schleifen	90
8.2.6	break und continue	92
8.2.7	return	94
8.2.8	assert	95
9	Methoden, Felder, Klassen und Objekte	97
9.1	Überblick und Einleitung	97
9.2	Methoden	98
9.3	Felder	100
9.4	Klassen	101
9.5	Objekte	103
10	Pakete, JAR-Dateien und Classpath	105
10.1	Pakete nutzen	106
10.1.1	Namen von Paketen	106
10.1.2	Anweisung import	107
10.1.3	Pakete, Verzeichnisse und JAR-Dateien	109
10.2	Eigene Pakete erstellen	110
10.2.1	Eigene Pakete, Verzeichnisse und CLASSPATH	112
10.2.2	Internationaler Namensraum für Pakete	112
10.2.3	Pakete und Zugriffsschutz	113
10.3	JAR-Dateien erzeugen und benutzen	114
10.4	Dokumentation javadoc	115
11	Elementare Ein/Ausgabe und Applikationen	119
11.1	Aufbau einer Applikation	119
11.1.1	Die Methode main()	120
11.1.2	Ausgabeweisungen	121
11.1.3	Aufrufparameter für Applikationen	123
11.1.4	Der Rückgabewert einer Applikation	124
11.2	Elementare Ein- und Ausgaben	125
12	Objektorientierte Programmierung	131
12.1	Grundideen und Begriffe der OOP	131
12.1.1	Grundideen der OOP	131
12.1.2	Klassen, Objekte und Instanzen	132
12.2	Klassen und Objekte in der OOP	133
12.2.1	Klasse Computer	134
12.2.2	Objekte einer Klasse	135
12.2.3	Konstruktoren	135
12.2.4	Datenkapselung	135
12.2.5	Nachrichten	137
12.3	Klassen in Java	138

12.3.1	Überblick und Beispiel	138
12.3.2	Instanzvariablen deklarieren	141
12.3.3	Instanzmethoden deklarieren	142
12.3.4	Überladen von Methoden	145
12.3.5	Verweis auf Instanzelemente mit <code>this</code>	147
12.3.6	Konstruktoren	148
12.3.7	Deklaration von Klassen	152
12.3.8	Modifikatoren	153
12.4	Instanzen/Objekte erzeugen	155
12.5	Aufruf von Instanzmethoden	156
12.5.1	Rückgabewerte und verschachtelte Methodenaufrufe	157
12.5.2	Methodenaufruf in einer Klasse	158
12.5.3	Parameterübergabe beim Methodenaufruf	159
12.6	Zugriff auf Instanzvariablen	163
12.7	Datenkapselung	165
12.8	Klassenvariable und Klassenmethoden	167
12.8.1	Klassenvariable	167
12.8.2	Konstanten	169
12.8.3	Klassenmethoden	170
12.8.4	Statische Initialisierungsblöcke	171
13	Vererbung	173
13.1	Konzept und Begriff der Vererbung	173
13.1.1	Vererbung	173
13.1.2	Subklassen erstellen mit <code>extends</code>	174
13.1.3	Klassenhierarchien	175
13.1.4	Typkompatibilität und Typkonvertierung	175
13.1.5	Einfach- und Mehrfachvererbung	177
13.2	Klassen erweitern - Vererbung in Java	177
13.2.1	Beispiel: Subklasse <code>Server</code>	178
13.2.2	Attribute vererben	179
13.3	Konstruktoren und <code>super()</code>	179
13.4	Überschreiben und <code>super</code>	184
13.4.1	Überschreiben von Methoden	184
13.4.2	Das Schlüsselwort <code>super</code>	185
13.4.3	Attribute verdecken	187
13.5	Dynamische Bindung	188
13.6	Vererbung und Überschreiben verhindern	191
13.7	Die Basisklasse <code>Object</code>	191
13.7.1	Methoden der Klasse <code>Object</code>	192
13.7.2	Referenzen vom Typ <code>Object</code>	193
13.7.3	Instanzen kopieren (<code>clone()</code>)	193
13.8	Abstrakte Klassen und Methoden	195
13.8.1	Programmierung abstrakte Klassen und Methoden	196
13.8.2	Beispiel zu abstrakten Klassen	200
13.9	Interfaces (Schnittstellen)	202
13.9.1	Interface deklarieren	203
13.9.2	Interface implementieren	203
13.9.3	Interface vererben	204
13.9.4	Referenztypen von Interfaces	205

13.10	Verschachtelte Klassen	206
13.10.1	Innere Klassen	207
14	Generische Typen und Collections	215
14.1	Typparameter	216
14.2	Implementation Generischer Methoden	217
14.3	Implementierung Generischer Klassen	218
14.4	Vector, Hashtable und Enumerationen	218
14.5	Das Java-Collections Framework	219
14.5.1	List	222
14.5.2	Klasse Vector	223
14.5.3	Map	224
14.5.4	Set	226
14.6	Initialisieren von Collection-Objekten	227
14.7	Collections und Objektprotokolle	228
14.7.1	Objektprotokolle	228
14.7.2	Die Methoden hashCode(), equals() und compareTo()	229
14.7.3	Berechnung von hashCode()	230
14.8	Iteratoren	230
14.9	Die Klasse Collections	232
	Fortgeschrittene Techniken und APIs (2. Band)	1
15	Application Programming Interfaces	1
15.1	API (Application Programming Interface)	1
15.1.1	Anwendung von API-Klassen und Methoden	3
15.1.2	Dokumentation zur JDK-API	3
15.2	String-Klassen	5
15.2.1	Klasse String	5
15.2.2	StringBuffer und StringBuilder	8
15.3	Klasse Math	9
15.4	Die Klasse System	11
15.5	Klasse Runtime	12
15.6	Properties	14
16	Grafische Benutzeroberflächen	19
16.1	Entwicklung der GUI-Programmierung	19
16.2	Von Komponenten und Containern	23
16.3	Programmieren mit Swing	28
16.4	Swing-Demo	29
16.5	Swing - Einfache Beispiele	30
16.5.1	Swing-Applet	30
16.5.2	Swing-Dialog	31
16.5.3	Look and Feel	33
16.5.4	Swing-Frame mit Menu und Icon	34
16.6	Ereignisgesteuertes Programmieren	37
16.6.1	Ereignisbehandlung unter Java	38
16.7	Dialogelemente	44
16.7.1	JLabel	45

16.7.2	JButton	45
16.7.3	JRadioButton und JCheckBox	46
16.7.4	JList	47
16.7.5	JComboBox	48
16.7.6	JTextField, JPasswordField, JFormattedTextField, JTextArea	48
16.7.7	JSlider und JSpinner	49
16.7.8	JSpinner	50
16.7.9	JProgressBar	50
16.7.10	JTextPane & JEditorPane	51
16.7.11	JTree	51
16.7.12	JTable	51
16.7.13	JMenuBar, JMenu, JMenuItem und JPopupMenu	53
16.8	Vordefinierte Dialoge	53
17	Fehlerbehandlung mit Exceptions	59
17.1	Exceptions	59
17.1.1	Klasse Exception	60
17.1.2	Exceptions auffangen und behandeln	60
17.2	Die try/catch-Anweisung	61
17.2.1	Aufbau der try/catch-Anweisung	62
17.2.2	try/catch-Beispiel bei Array-Zugriff	62
17.2.3	IOException mit try/catch auffangen	64
17.2.4	Mehrere catch-Blöcke	65
17.2.5	finally	66
17.2.6	Fehlersituationen mit RuntimeException	66
17.3	Weiterreichen von Exceptions	67
17.4	Methoden und Klassen mit Exceptions erstellen	68
17.4.1	Exceptions in Methoden erzeugen und weiterreichen	68
17.4.2	Eigene Exception-Klassen	69
18	Applets und Grafik	73
18.1	Applets im Überblick	73
18.2	Programmstruktur von Applets	74
18.2.1	Methode init()	74
18.2.2	Methode paint()	75
18.2.3	Weitere Methoden der Klasse Applet	75
18.3	Ausführen von Applets	76
18.3.1	HTML-Tag <applet>	77
18.3.2	Parameterübergabe an Applets	79
18.4	Koordinatensystem und Zeichenfläche	80
18.5	Grafikelemente aus Graphics	81
18.6	Farben	84
18.7	Textausgabe	85
18.7.1	Schrift mit der Klasse Font ändern	85
18.7.2	Schriftname, Stil und Größe abfragen	86
18.7.3	Schriftattribute - Fontmetrik und Grafikumgebung	87
18.7.4	Verfügbare Fonts abfragen	88
18.8	Bilder laden und darstellen	90
18.9	Sound in Java	92

19 Streams und Dateien	95
19.1 Streams	95
19.2 Character-Streams	96
19.3 Filter-Streams und Filter-Reader	97
19.3.1 BufferedReader	99
19.3.2 BufferedWriter	99
19.4 Datei- und Verzeichnis-Bearbeitung	100
19.5 Byte-Streams	101
19.6 Standard-Datenströme und Tastatureingabe	103
20 Netzwerkprogrammierung	107
20.1 Netzwerke und Protokolle	107
20.2 Beispiele zu InetAddress, Socket und URL	111
20.2.1 IP-Nummer und Domain-Namen	111
20.2.2 Socket-Verbindung	112
20.2.3 URL-Dokument im WWW-Browser anzeigen	113
20.3 Server programmieren und starten	114
21 Fortgeschrittene Java-APIs	117
21.1 Annotationen	118
21.2 Reflection	120
21.3 Objektpersistenz	125
21.4 Datenbanken und JDBC	129
21.5 Remote Method Invocation	137
22 Threads	145
22.1 Threads	145
22.2 Threads erstellen und ausführen	146
22.2.1 Klasse Thread	147
22.2.2 Interface Runnable	149
22.2.3 Applets und Klasse Thread	151
22.3 Synchronisation	152
22.4 Threads stoppen	153
22.5 Threads unterbrechen	155
22.6 Synchronisation mit wait und notify	156
23 Programmierwerkzeuge JDK	163
23.1 Das Java Development Kits JDK	163
23.2 Editionen des JDK	164
23.3 JDK Download und Installation	164
23.3.1 Download-Seiten und Java-Versions-Auswahl	164
23.3.2 Installation Windows	165
23.3.3 Systemvariablen anpassen	165
23.3.4 Installation Linux	166
23.4 Installation von jEdit	168
23.5 Programmierwerkzeuge des JDK	169
23.5.1 Aufrufsyntax von javac und java	169
23.6 Beispiele zur Java-Programmierung	171
23.7 Dokumentation zum JDK	172
23.8 Sourcecode des JDK	173

Stichwortverzeichnis 173

Kapitel 15

Application Programming Interfaces

Dieses Kapitel erläutert die Verwendung von Application Programming Interfaces (APIs). Für die Java-Programmierung ist der Umgang mit der JDK-API und das Verständnis der dazugehörigen API-Dokumentation notwendig. In Abschnitt 15.1 werden diese Themen behandelt.

Außer einer Einführung in die Nutzung der JDK-API werden in diesem Kapitel auch Beispiele für Klassen aus der JDK-API ausführlicher behandelt. Dazu wurden Klassen ausgewählt, die nicht unmittelbar zu einem der anderen Kapitel über Java-Anwendungen in diesem Buch gehören, aber für die Java-Programmierung wichtig sind.

In fast jedem Java-Programm werden Zeichenfolgen benötigt. Die JDK-API enthält für die Bearbeitung von Zeichenfolgen u. a. die Klassen `String` und `StringBuffer`. In den beiden Abschnitten 15.2 “String-Klassen” und 15.2.2 “StringBuffer und StringBuilder” werden die Klassen vorgestellt, die Unterschiede erläutert und Beispiele zur Programmierung mit diesen Klassen gezeigt.

Zu den auch in den Beispielen dieses Buches häufiger verwendeten Klassen gehören `System` und `Math` aus dem Paket `java.lang`. Die beiden Klassen werden in den Abschnitten 15.3 “Klasse Math” und 15.4 “Die Klasse System” erläutert.

Aus dem API-Paket `java.util` wurden die Collections schon im Abschnitt 14 “Generische Typen und Collections” ausführlich behandelt, insbesondere die Klassen `List`, `Vector`, `HashMap` und `HashSet`.

15.1 API (Application Programming Interface)

API

Eine API (Application Programming Interface) ist eine Sammlung von Klassen mit zugehörigen Methoden für Programmieraufgaben. Eine bereits vorgestellte API-Klasse aus dem Java-System ist die Klasse `String`. Die Klasse `String` enthält Methoden zur Programmierung von Aufgaben zur Zeichenfolgenbearbeitung in Java. Die Klasse `String` gehört zur API des Java-Standard JDK (JDK-API).

Die JDK-API wird automatisch zusammen mit JDK installiert und in vorgegebenen Systemverzeichnissen abgelegt. Die JDK-API wird auch als Java-API,

<code>java.lang</code>	Paket für fundamentale Programmieraufgaben in Java (Typkonvertierung, String-Bearbeitung, mathematische Routinen). In diesem Kapitel werden die Klassen <code>System</code> (Abschnitt 15.4 “Die Klasse System”, 11) sowie <code>String</code> und <code>StringBuffer</code> (Abschnitt 15.2 “String-Klassen”, 5) und <code>Math</code> 15.3 “Klasse Math”, 9, behandelt. Die Klassen aus dem Paket <code>java.lang</code> sind in jedem Java-Programm ohne <code>import</code> -Anweisung verfügbar.
<code>java.util</code>	Paket mit “nützlichen Klassen für viele Hilfsaufgaben” (Kalender, Listen, Arrays, Datum usw.). Auch die Klassen für die Collections (siehe Abschnitt 14 “Generische Typen und Collections”) sind hier untergebracht.
<code>javax.swing</code>	Das Swing-Toolkit ist die Bibliothek für grafische Benutzerschnittstellen. Die Programmierung mit dem Paket <code>java.swing</code> wird ausführlich in Kapitel 16 “Grafische Benutzeroberflächen” ab Seite 19 behandelt.
<code>java.net</code>	<code>Net(working)</code> enthält Klassen für die Netzwerk-Programmierung in Java. Die Programmierung mit den Klassen aus <code>java.net</code> wird in Kapitel 20 “Netzwerkprogrammierung”, Seite 107 gezeigt.
<code>java.applet</code>	Das Paket <code>java.applet</code> enthält die Klasse <code>Applet</code> , die zur Programmierung von Applets dient, siehe dazu auch Kapitel 18 “Applets und Grafik”, Seite 73.

Tabelle 15.1: JDK-Pakete

oder Java-Standard-API bezeichnet. Dies rührt daher, dass es noch eine Vielzahl weiterer APIs gibt (z.B. zur Nutzung von Web-Diensten, für den Versand von Mails), die nicht in diesem Handbuch beschrieben sind.

API-Pakete

APIs sind in Paketen sortiert (siehe Kapitel 10 “Pakete, JAR-Dateien und Classpath”, Seite 105). Die Pakete aus der JDK-API beginnen alle mit dem Namen `java` (`java.lang`, `java.util`, `java.net`, ...) oder `javax` (`javax.swing`). Einige Pakete beginnen mit dem Namen `sun`, diese Pakete sind aber nicht Teil der offiziellen API, sie können sich jederzeit ändern oder ganz entfallen.

Die folgende Tabelle enthält die Namen einiger API-Pakete aus dem JDK, eine kurze Beschreibung des jeweiligen Pakets und einige Klassen, die zu dem Paket gehören. Die Auswahl beschränkt sich auf die sehr wenigen Pakete und Klassen, die in diesem Buch verwendet werden.

Neben der JDK-API sind für Java mittlerweile weitere APIs für unterschiedlichste Anwendungsbereiche verfügbar. Für eine Reihe von Anwendungsbereichen werden von Oracle APIs veröffentlicht, z. B. APIs für professionelle Client-Server-Anwendungen (JEE, Java Enterprise Edition), XML, SmartCards usw. Die APIs und Informationen dazu sind auf den Java-Seiten von Oracle zu finden (<http://www.oracle.com/technetwork/java/index.html#10>).

Firmen aus den Bereichen Datenbanken, WWW-Server, Dokumentenverarbeitung, intelligente Hardware usw. bieten mittlerweile eigene Java-APIs zur Nutzung ihrer Produkte mit Java an. Die meisten APIs sind frei im Internet verfügbar. Sie müssen zusätzlich zum JDK installiert werden.

15.1.1 Anwendung von API-Klassen und Methoden

Um Klassen und Methoden aus APIs anwenden zu können, sind zwei Maßnahmen erforderlich:

- Dem Java-Compiler und dem Laufzeitsystem muss mitgeteilt werden, in welchem Verzeichnis die API zu finden ist. Dies erfolgt entweder mit der Umgebungsvariablen `CLASSPATH` oder den entsprechenden Optionen (`classpath`) beim Aufruf von Compiler, Interpreter oder Appletviewer, siehe Abschnitt 10 “Pakete, JAR-Dateien und Classpath”). Wichtig ist, dass in beiden Fällen die gleiche Version der Bibliothek benutzt wird. Ansonsten kann es zu hässlichen Fehlern kommen, weil Ihr Java-Bytecode zur Laufzeit Klassen verlangt, die u.U. in einer älteren Version der API nicht enthalten sind.

`CLASSPATH`



Für die Anwendung der APIs aus der installierten Java 2 Edition (J2SE) muss die Variable `CLASSPATH` nicht gesetzt werden. Das Java-System findet die Pakete aus dem JDK automatisch. Für die Beispielprogramm in diesem Buch sollte im Zweifelsfall die Variable `CLASSPATH` aus der Systemumgebung entfernt werden.

- Werden in einem Programm Klassen aus API-Paketen genutzt, muss dem Compiler mitgeteilt werden, aus welchem Paket die Klassen stammen. Dies erfolgt entweder durch Angabe der qualifizierten Namen (wird selten eingesetzt) oder durch Angabe der Paket-Namen mit `import` am Anfang des Quelltextes, siehe 10.1 “Pakete nutzen” und 10.1.2 “Anweisung import”.

`import`

Das Paket `java.lang` und die darin enthaltenen Klassen `String`, `Math` usw. sind in jedem Java-Programm implizit verfügbar. Um die Klassen aus diesem Paket zu nutzen, muss das Paket nicht explizit mit `import java.lang.*` vereinbart werden.

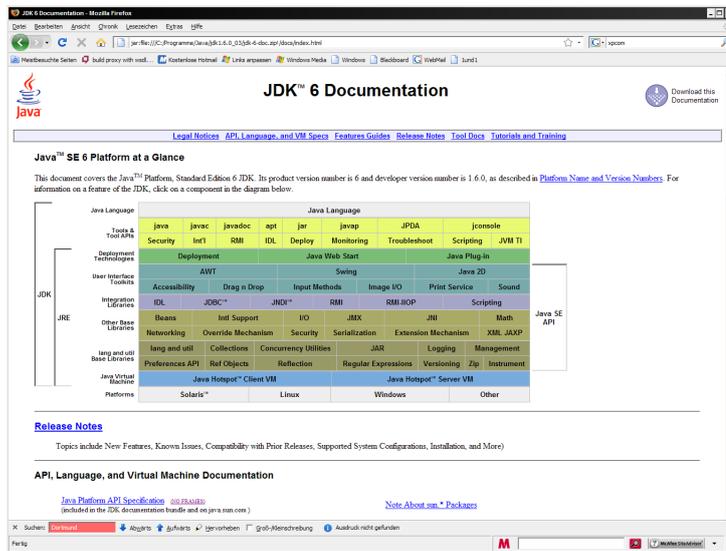
Bezüglich der Programmierung sind eigene Klassen und Methoden denjenigen aus API-Bibliotheken völlig gleichberechtigt und können genauso genutzt werden, d. h. es werden z. B. Instanzen der API-Klassen erstellt und Methoden der Klassen aufgerufen.

15.1.2 Dokumentation zur JDK-API

Die Erstellung von Java-Programmen ist ohne eine API-Dokumentation, in der die benötigten Klassen mit allen Methoden, Parametern und Rückgabewerten ausführlich beschrieben sind, kaum möglich.

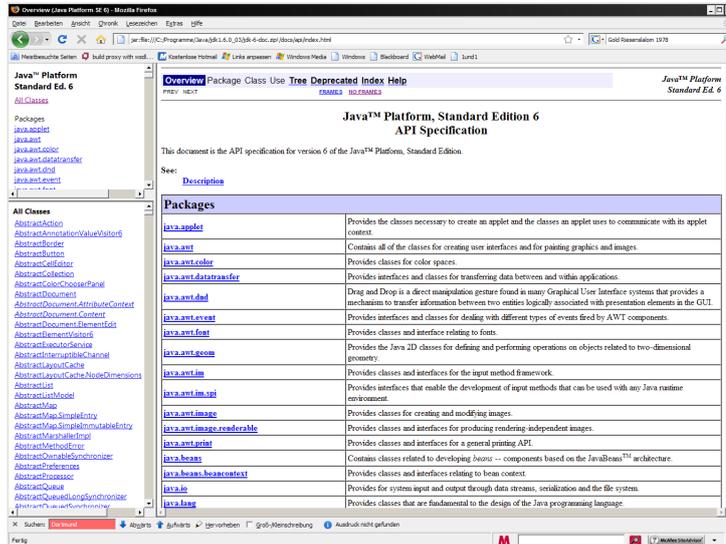
Die Dokumentation zur JDK-API gehört nicht zum Umfang der JDK-Installation und muss separat installiert werden. Hinweise zur Installation der Dokumentation enthält Abschnitt 23.7 “Dokumentation zum JDK”, Seite 172.

Ist die Dokumentation entsprechend den Empfehlungen im Verzeichnis des `jdk` installiert, findet sich im Verzeichnis `docs` die Datei `index.html` als Startseite für die gesamte JDK-Dokumentation. Die Datei `index.html` kann mit jedem modernen WWW-Browser geöffnet werden.



Die für die Programmierung wichtige API-Dokumentation ist direkt unter der Grafik bei dem Stichwort “API, Language, and Virtual Machine Documentation” der “Link Java Platform API Specification” zu finden. Der Link führt zur Seite `../docs/api/index.html`

Die Startseite der API-Dokumentation bietet in der Frame-Version folgendes Bild:



Im unteren Frame links finden Sie alle zum Standard-API gehörenden Klassen. Wählen Sie beispielsweise die Klasse `String` aus, erhalten Sie eine Dokumentation zu allen Methoden und Konstruktoren der Klasse `String`.

Eine weitere wichtige Navigationshilfe in der API-Dokumentation ist der Index, der in der oberen Zeile des rechten Hauptfensters anklickbar ist. Über den Index finden Sie nach alphabetischer Sortierung alle Klassen, Methoden und weiter Stichworte die zu den JDK-APIs gehören.

Die Dokumentation zum JDK enthält u. a. auch die Deklarationszeilen aller Methoden der JDK-API. Beispielsweise findet sich für die Methode `substring()` in der Dokumentation der Klasse `String` (`file:///.../docs/api/java/lang/String.html`)₁₁ folgende Zeile:

```
String substring(int beginIndex, int endIndex)
    Returns a new string that is a substring of this string
```

Die Zeilen aus der Dokumentation liefern folgende Information über die Methode:

- Der Name der Methode ist `substring` formale Parameter
- Es werden zwei formale Parameter festgelegt:
 - `int beginIndex, int endIndex` aktuelle Parameter Aktualparameter

Beim Aufruf der Methode sind für die beiden formalen Parameter aktuelle Werte anzugeben (Aktualparameter). Die beim Aufruf der Methode anzugebenden Werte für die Parameter müssen vom Typ `int` sein. Rückgabewert
- Der Rückgabewert der Methode `substring()` ist vom Typ `String`. Die Methode kann an jeder Stelle im Programm aufgerufen werden, an der ein `String` zulässig ist.

15.2 String-Klassen

Das Paket `java.lang` enthält zur Bearbeitung von Zeichenfolgen die drei Klassen `String` (`file:///.../docs/api/java/lang/String.html`)₁₂, `StringBuffer` (`file:///.../docs/api/java/lang/StringBuffer.html`)₁₃ und `StringBuilder` (`file:///.../docs/api/java/lang/StringBuilder.html`)₁₄. Alle Klassen gehören zum Paket `java.lang` und sind damit jedem Java-Programm standardmäßig ohne `import`-Anweisung verfügbar.

Die Klasse `String` war bereits in Kapitel 7.3 “Strings - Zeichenketten und Ihre Methoden” ab Seite 75 ausführlich besprochen worden. Hier werden wir nur diejenigen Details aufgreifen, die dort fehlen mussten. Umfangreiche Verkettungen von Zeichenfolgen und andere dynamische Operationen sollten mit den Klassen `StringBuffer` und `StringBuilder` realisiert werden.

15.2.1 Klasse `String`

`String` ist die “Standard-Klasse” zur Konstruktion und Bearbeitung von Zeichenfolgen in Java. Für einfache `String`-Anwendungen kann die Klasse `String` “intuitiv” genutzt werden, d. h. es können z. B. Zeichenfolgen verkettet und Teilzeichenfolgen gebildet werden, wie dies in Abschnitt 7.3 “Strings - Zeichenketten und Ihre Methoden” gezeigt wurde. Für weitergehende und umfangreiche Anwendungen mit Zeichenfolgen sind aber einige Eigenschaften der Klasse `String` zu beachten:

Strings sind Referenztypen

Alle Variablen und Literale vom Typ `String` sind Referenzvariablen. Für Vergleichsoperationen ist daher die Methode `equals()` zu nutzen, siehe Abschnitt 7 Referenzvariable

“Referenztypen (Klassen)”, Seite 67. Wir werden im weiteren noch sehen, dass nur unter ganz bestimmten Umständen der Vergleich mit `equals()` durch `==` ersetzt werden kann.

Strings sind schreibgeschützt

`final` Die Klasse `String` ist `final`, d. h. es können keine (eigenen) Subklassen von `String` mit erweiterten Methoden zur `String`-Bearbeitung erstellt werden.

Schreibschutz

`String`-Instanzen sind schreibgeschützt. Die Verknüpfung zweier `String`s durch `+` hängt nicht den zweiten `String` an den ersten an, sondern erzeugt einen neuen `String`, der “zufällig” aus den Zeichen des ersten und des zweiten `String`s besteht. Sofern die ursprünglichen `String`s nirgends mehr referenziert sind, werden Sie dem Garbage Collector übergeben. Ganz besonders problematisch wird dieses Verhalten, wenn ein langer `String` aus vielen kurzen Abschnitten zusammengesetzt wird wie im folgenden Beispiel:

```

1 import static java.lang.System.currentTimeMillis; 12 out.println(String.format("%>
%< Millis; 13 fmt, i,
2 import static java.lang.System.out; 13 currentTimeMillis()-st>
3 14 art));
4 public class Test { 14 }
5 public static void main(String [] args> 15 }
%< ) { 16 out.println(String.format(fmt, 500>
6 String a=""; 16 00,
7 String fmt = "%d Zeichen in %d ms"; 17 currentTimeMillis()-start));
8 long start = currentTimeMillis(); 18 }
9 for ( int i=0; i<50000; i++) { 19 }
10 a=a+"a"; 20
11 if ( i%1000 == 0 ) {

```

Programm 15-1: Ein gutes Beispiel für die falsche Nutzung von `String`

Das obige Programm benötigt auf einem üblichen Rechner ca. 16 Millisekunden für die ersten 1000 Schleifendurchläufe und ca. 350 Millisekunden für die letzten 1000 Schleifendurchläufe. Dies liegt daran, dass beim x -ten Schleifendurchlauf zunächst $x-1$ Zeichen aus dem `String` `a` in einen neuen `String` umkopiert werden müssen, bevor das letzte “a” angehängt werden kann. Insgesamt führt das obige Programm also 50.000 Kopiervorgänge mit im Mittel 25.000 Zeichen aus. Das sind 1.250.000.000 kopierte Zeichen.

Instanzen und Literale von `String`

Jede in Hochkommata eingeschlossene Zeichenfolge wird als Instanz vom Typ `String` behandelt. Verweise auf `String`-Instanzen können mit Variablen vom Typ `String` erzeugt werden. Beispiel:

```
String a = new String("Asterix");
```

Es wird eine Instanz der Klasse `String` erzeugt. Als aktueller Wert wird beim Konstruktor-Aufruf das `String`-Literal “Asterix” angegeben. Die Variable `a` vom Typ `String` verweist auf den `String` mit dem Wert “Asterix”. Jedoch ist diese Art der Erzeugung von `String`-Objekten redundant, da schon das Literal “Asterix” selbst ein Instanz der Klasse `String` ist. Daher kann Variablen vom

Typ `String` direkt eine Referenz auf ein `String`-Literal direkt zugewiesen werden. Das Schlüsselwort `new` kann dabei entfallen. Beispiel:

```
String b = "Obelix";
```

Die Variable `b` vom Typ `String` verweist nach der Zuweisung auf die `String`-Instanz mit dem Wert `"Obelix"`. Intern bestehen `String`-Instanzen aus einer Folge von Einzelzeichen (Werte vom Grundtyp `char`), die über einen Laufindex ansprechbar sind und die in einem Array vom Typ `char` gespeichert sind. Im Gegensatz z. B. zu C oder C++ sind Strings in Java aber keine Arrays von Zeichen. Die einzelnen Zeichen können daher nicht durch `[]`, sondern nur durch die Methode `charAt()` extrahiert werden.

Die Konvertierung zwischen `char`-Arrays und `String`-Werten erfolgt über den Konstruktor bzw. die Methode `toCharArray()`.

Der Laufindex für einen `String` beginnt bei 0, d. h. das erste Zeichen im `String` `"Index"` ist das Zeichen `'I'` und hat den Index 0. Das letzte Zeichen ist `'x'` und hat den Laufindex 4.

String-Pool und intern()

Alle zur Programmlaufzeit benötigten `String`-Konstanten werden von der Klasse `String` in einem internen Pool verwaltet. In diesem Pool werden keine `String`-Instanzen mehrfach vorgehalten. Variablen vom Typ `String` verweisen, wenn möglich, auf dieselbe `String`-Instanz im Pool. String-Pool

In den folgenden Programmzeilen werden zwei Variablen `o` und `o2` vom Typ `String` erzeugt. Beiden Variablen wird die Zeichenfolge `"Obelix"` zugewiesen. Da im internen `String`-Pool nur eine Instanz mit dem Wert `"Obelix"` verwaltet wird, verweisen beide Variablen auf dieselbe Instanz, d. h. die Referenzwerte beider Variablen sind gleich.

```
String o1="Obelix";
String o2="Obelix"; // o und o2 verweisen auf dieselbe
                  // Instanz Obelix aus dem String-Pool
System.out.println(o1 == o2);// ergibt true
char []oa = {'O','b','e','l','i','x'};
String o3 = new String(oa);
System.out.println(o1 == o3);// ergibt false
```

Der aus dem Zeichen-Array `oa` konstruierte `String` `o3` wird an anderer Stelle im Hauptspeicher platziert, da der Compiler die (inhaltliche) Übereinstimmung zwischen `o1`, `o2` und `o3` zur Übersetzungszeit nicht automatisch prüft.

Der Aufruf

```
einString.intern();
```

`intern`

liefert eine Referenz auf einen `String` aus dem Pool genau dann, wenn bereits ein identischer `String` mit dem Inhalt von `einString` vorliegt. Andernfalls wird der Wert der `einString`-Instanz dem Pool zugefügt. Die Methode `intern()` kann genutzt werden, um einen schnellen Vergleich von `String`-Instanzen durchzuführen. Es kann statt der "langsameren" Methode `equals()` für `String`-Instanzen eingesetzt werden. Ein Beispiel für die Verwendung von `intern()` zeigt das Programm `StringDaten04`.

```
String o1="Obelix";
String o2="Obelix"; // o und o2 verweisen auf dieselbe
                  // Instanz Obelix aus dem String-Pool
System.out.println(o1 == o2);// ergibt true
```

```

char []oa = {'0','b','e','l','i','x'};
String o3 = new String(oa);
o3 = o3.intern()
System.out.println(o1 == o3);// ergibt true

```

In diesem Beispiel ergibt der letzte Vergleich `true`, weil der Aufruf von `o3.intern()` den im Stringpool gespeicherten Wert von `o1` bzw. `o2` zurückliefert. Durch Zuweisung dieser Referenz an `o3` verweisen jetzt alle 3 Strings auf dasselbe Objekt.

15.2.2 StringBuffer und StringBuilder

Die (ältere) Klasse `StringBuffer` und die (neuere) Klasse `StringBuilder` sind die Klassen für dynamische Operationen auf Zeichenketten. Bei umfangreichen Operationen auf Zeichenketten sollten aus Performance-Gründen Objekte dieser beiden Klassen eingesetzt werden. Vom Java-System werden beide Klassen intern zur Verkettung von Zeichenfolgen benutzt. Die folgenden Programmzeilen sind eine Verkettung von `String`-Literalen.

Verkettung von Zeichenketten

```

1 import static java.lang.System.currentTimeMillis; 12 out.println(String.format(
2< Millis; 13 fmt, i,
2 import static java.lang.System.out; 13 currentTimeMillis()-st
3 13 art));
4 public class Test { 14 }
5 public static void main(String [] args) { 15 }
6< ) { 16 out.println(String.format(fmt, 500
7 StringBuffer a=new StringBuffer(); 16 00,
8 String fmt = "%d Zeichen in %d ms"; 17 currentTimeMillis()-start));
9 long start = currentTimeMillis(); 18 }
10 for ( int i=0; i<50000; i++) { 19 }
11 a.append("a"); 20
11 if ( i%1000 == 0 ) {

```

Programm 15-2: Verbesserte Version der Zeichenkettenkonstruktion

Die verbesserte Form erzeugt nicht mehr nach jedem Schleifendurchlauf ein neues Objekt, sondern verwaltet den Text intern als änderbares Array von Zeichen. Läuft dieses Array über, wird es durch ein neues Array mit doppelt soviel Speicherplatz ersetzt, dadurch wird im Mittel jedes Zeichen höchstens 2 Mal kopiert.

Entsprechend benötigt das Verfahren auf einem handelsüblichen Rechner nicht mehr ca. 7,5 Sekunden, sondern nur 75 Millisekunden. Nur durch die Verwendung einer geeigneten Klasse wurde die Laufzeit also um den Faktor 100 verbessert!!

Im Folgenden sind Konstruktoren und Methoden der Klasse `StringBuffer` aufgeführt. Alle Konstruktoren und Methoden existieren auch für die Klasse `StringBuilder` (abgesehen vom geänderten Namen für den Konstruktor)

<code>StringBuffer()</code>	Der Konstruktor erzeugt eine <code>StringBuffer</code> -Instanz.
<code>StringBuffer(int n)</code>	Es wird eine <code>StringBuffer</code> -Instanz mit einer Kapazität von <code>n</code> Einzelzeichen gebildet.
<code>StringBuffer(String s)</code>	Die erzeugte Instanz von <code>StringBuffer</code> hat eine Anfangskapazität, die der Länge des übergebenen Strings entspricht.

Methoden aus der Klasse `StringBuffer` sind u. a.:

<code>StringBuffer append(String s)</code>	Die Methode hängt den <code>String s</code> an eine Instanz von <code>StringBuffer</code> an.
<code>StringBuffer insert(int p, String s)</code>	Die Methode fügt den <code>String s</code> an der Position <code>p</code> in einen <code>StringBuffer</code> ein.
<code>StringBuffer delete(int start, int end)</code>	Löschen der Daten von Position <code>start</code> bis vor Position <code>end</code> ein.
<code>StringBuffer replace(int start, int end, String s)</code>	Ersetzt der Daten von Position <code>start</code> bis vor Position <code>end</code> ein durch <code>s</code> .
<code>int indexOf(String s)</code>	Suche nach der Anfangsposition von <code>s</code> im <code>StringBuffer</code> .
<code>String substring(int start, int end)</code>	Teilstring von Position <code>start</code> bis vor Position <code>end</code> ein. Grund für diese Wahl ist, dass der ausgeschnittene Teil genau die Länge <code>end-start</code> haben soll.
<code>String toString()</code>	Die <code>StringBuffer</code> -Instanz wird in eine <code>String</code> -Instanz konvertiert.

Eine Verkettung von Strings der Art "Hallo"+" "+"Java" wird mit einer `StringBuffer`-Instanz folgendermaßen formuliert:

```
StringBuffer hallo = new StringBuffer();
hallo.append("Hallo");
hallo.append(" ");
hallo.append("Java");
System.out.println(hallo.toString());
```

Da die Methode `append` eine Referenz auf ihr eigenes Objekt zurück gibt, kann der obige Code auch etwas kürzer formuliert werden:

```
StringBuffer hallo = new StringBuffer();
hallo.append("Hallo").append(" ").append("Java");
System.out.println(hallo.toString());
```

Alternativ zu `StringBuffer` kann die Klasse `StringBuilder` benutzt werden, bei der die Operationen nicht synchronisiert sind. Der hierbei erzielbare Gewinn an Geschwindigkeit ist allerdings nicht zu groß, weil beide Klassen intern die Methoden der gemeinsamen Basisklasse `AbstractStringBuilder` benutzen. In den meisten Applikationen lassen sich beide Klassen daher parallel einsetzen.

15.3 Klasse Math

Die Klasse `Math` (file:///.../docs/api/java/lang/Math.html)¹⁵ gehört ebenfalls zum Paket `java.lang` und ist damit in allen Java-Programmen standardmäßig verfügbar. `Math` enthält Methoden für mathematische Berechnungen. Alle Methoden von `Math` sind statische Methoden, die somit direkt mit dem Klassennamen

aufzurufen sind.

Im Folgenden sind einige Methoden aus `Math` aufgeführt und kurz erläutert. Der Modifikator `static` wurde in der folgenden Beschreibung zwar nicht mit aufgeführt, gehört aber zum Deklarations-Kopf aller Methoden der Klasse `Math`.

<code>double sin(double a)</code> <code>double cos(double a)</code> <code>double tan(double a)</code>	Liefern den Sinus, Cosinus, und Tangens. Das Argument wird als Zahl im Bogenmass interpretiert. 90 Grad entsprechen $\pi/2$.
<code>double asin(double a)</code> <code>double acos(double a)</code> <code>double atan(double a)</code>	Liefern den Arkus-Sinus (Wert 0 bis π), Arkus-Cosinus (Wert von $-\pi/2$ bis $\pi/2$), und Arkus-Tangens (Wert von $-\pi/2$ bis $\pi/2$). Das Argument für <code>acos</code> und <code>asin</code> muss eine Zahl zwischen -1 und 1 sein.
<code>double toRadians(double deg)</code> <code>double toDegrees(double grad)</code>	Konvertiert eine Radiant-Angabe in Grad bzw. umgekehrt.
<code>double cos(double a)</code> <code>double sin(double a)</code> <code>double tan(double a)</code>	Liefern den trigonometrischen Cosinus, Sinus und Tangens.
<code>double log(double a)</code> <code>double log10(double a)</code>	Liefern Logarithmus zur Basis e bzw. 10.
<code>double exp(double x)</code> <code>double pow(double a, double x)</code>	Liefern Exponentialwert zur Basis e bzw. zur Basis a .
<code>double sqrt(double a)</code>	Liefert die Quadratwurzel.
<code>double rint(double a)</code> <code>double floor(double a)</code> <code>double ceil(double a)</code>	Liefern zu einem <code>double</code> -Wert den gerundeten Wert als <code>double</code> . Die Methode <code>floor()</code> rundet ab, <code>ceil()</code> rundet auf und <code>rint()</code> rundet kaufmännisch).
<code>long round(double a)</code> <code>long round(float a)</code>	Liefern zu einem <code>double</code> - oder <code>float</code> -Wert, den gerundeten Wert als <code>long</code> . Beide Methoden runden kaufmännisch).
<code>double random()</code>	Liefert eine Pseudo-Zufallszahl im Bereich von 0 bis kleiner 1.0.
<code>double abs(double x)</code> <code>double max(double x, double y)</code> <code>double min(double x, double y)</code>	Liefern den Absolutbetrag, das Maximum oder Minimum ihrer Argumente zurück. Statt <code>double</code> kann auch <code>float</code> , <code>int</code> oder <code>long</code> verwendet werden, dann ändert sich der Rückgabetypp entsprechend.
<code>int signum(double x)</code> <code>int signum(float x)</code>	Liefern das Vorzeichen ihres Argumentes zurück, 1 für positive Zahlen, 0 für 0 und -1 für negative Zahlen.

Die Klasse `Math` definiert außerdem die mathematischen Konstanten `PI` und `E`:

Das Programm `MathDemo` zeigt die Anwendung einiger Methoden aus `Math`.

```

1 public class MathDemo{                               %< Funktionen");
2   public static void main(String[] args){           14   double rad;
3     System.out.println("\nQuadratwurzel ");        15   for(int i=1;i<=16;i*=2){
4     double w;                                       16     rad = Math.PI/i;
5     w = Math.sqrt(2);                               17     System.out.println(" sin(PI/"+i+" ">>%
6     System.out.println(" sqrt(2) "+w);             %< +Math.sin(rad));
7     System.out.println("\nZufallszahlen un>%      18     System.out.println(" asin("+Math.sin>%
%< d Vergleich");                                   %< (rad)+") "+
8     double ra,rb;                                   19     "PI/"+Math.PI/(Math.asin(Math.sin(>%
9     ra = Math.random();                             %< rad)))));
10    rb = Math.random();                              20   }
11    System.out.println("Zufallswerte: " + %<      21   }
%< ra + rb);                                         22   }
12    System.out.println("Minimum: " + Math.>%      23
%< min(ra,rb));
13    System.out.println("\nTrigonometrische>%

```

Programm 15-3: Anwendung verschiedener Methoden aus Math

15.4 Die Klasse System

Die Klasse System (file:///.../docs/api/java/lang/System.html)¹⁶ gehört zum Paket `java.lang` und enthält viele "hilfreiche" Methoden.

Zur Klasse System gehören u. a. die folgenden Methoden. Alle Methoden sind statisch, der Modifikator `static` ist in der folgenden Beschreibung nicht mit aufgeführt:

<code>void arraycopy(Object src, int srcPos, Object dst, int dstPos, int len)</code>	Kopiert Elemente von einem Array in ein anderes Array.
<code>long currentTimeMilis() long nanoTime()</code>	Die Methode liefert die aktuelle Systemzeit in Milli- bzw. Nanosekunden.
<code>void exit(int status)</code>	Mit <code>exit()</code> wird die aktuell laufende Java-VM beendet. Als <code>status</code> wird konventiongemäß der Wert 0 übergeben, wenn das Programm in einem fehlerfreien Zustand beendet wird.
<code>void gc()</code>	Die Methode ruft den Garbage Collector auf, die identische Methode ist <code>Runtime.gc()</code> .
<code>void load(String filename) void loadLibrary(String libraryName) String mapLibraryName(String libName)</code>	Diese drei Methoden laden in Maschinsprache erstellte Bibliotheken, die vollständige Beschreibung findet sich in der Klasse <code>Runtime</code> .

JNI Java Native Interface

Die Klasse System enthält mit `getProperties()` und `setProperties()` auch Verfahren, um Eigenschaften des umgebenden Systems abzufragen und ggf. zu setzen. Die Details finden sich um Abschnitt 15.6

Systemeigenschaften Properties

Umgebungsvariablen

Mit der Methode `getenv(String name)` und `getenv()` lassen sich Umgebungsvariablen, z. B. `PATH` oder `CLASSPATH` abfragen. Während `getenv(String name)` nur den Inhalt einer bestimmten Umgebungsvariablen auslesen kann, liefert `getenv()` eine nicht modifizierbare `Map<String, String>` zurück. Das folgende Programm zeigt zwei Möglichkeiten, diese auszugeben:

```

1 import java.util.Map;                9 // Zweite Lösung
2 class Environment {                  10 for (Map.Entry<String, String> s: env.>
3     public static void main(String [] args) { >
4         Map<String, String> env = System.geten> 11     System.out.println(s.getKey() + "=" >
5         v();                          12     + s.getValue());
6         // Erste Lösung                13     }
7         for (String s: env.keySet()) {    14     }
8         System.out.println(s + "=" + env.get> 15     }
9         (s));
10    }

```

Programm 15-4: Auslesen von Umgebungsvariablen

Standard-Ein/Ausgabe

Ebenfalls zur Klasse `System` gehören die drei statischen Variablen `in`, `out` und `err` für die Standard-Ein- und Ausgabe-Ströme. Die Standard-Ein-/Ausgabe-Ströme werden im Zusammenhang mit Streams in Abschnitt 19.6 "Standard-Datenströme und Tastatureingabe" behandelt.

<code>System.in</code>	Standard-Eingabestrom
<code>System.out</code>	Standard-Ausgabestrom
<code>System.err</code>	Standard-Ausgabestrom für Fehlermeldungen

15.5 Klasse Runtime

Die Klasse `Runtime` (<file:///.../docs/api/java/lang/Runtime.html>)¹⁷ gehört zum Paket `java.lang`. Während die Klasse `System` eher die Betriebssystemschicht abstrahiert, kontrollieren die Methoden von `Runtime` die Java-Laufzeitumgebung.

Zur Klasse `System` gehören u. a. die folgenden drei Methoden. Alle drei Methoden sind statisch, der Modifikator `static` ist in der folgenden Beschreibung nicht mit aufgeführt:

Die Details der Integration zwischen Java und Maschinensprache (JNI) sind in diesem Handbuch nicht beschrieben, es gibt aber eine Dokumentation unter (<http://java.sun.com/developer/onlineTraining/Programming/JDCBook/jni.html>)¹⁸.

Die Klasse `Runtime` besitzt keinen Konstruktor, daher gibt es die Methode `Runtime.getRuntime()`, die das einzige Objekt dieser Klasse (ein sogenanntes Singleton) zurückliefert.

Prozesse

Im letzten Unterabschnitt wurde mit der Methode `Runtime.exec()` eine Möglichkeit zur Ausführung anderer Programme vorgestellt. Diese Methode liefert ein Objekt vom Typ `Process` zum Zugriff auf die Eigenschaften des Prozesses:

<code>int availableProcessors()</code>	Liefert die Zahl der vorhandenen Prozessoren.
<code>Process exec(String command) Process exec(String cmdarray[]) Process exec(String cmdarray[],String envp[]) Process exec(String cmdarray[],String envp[], File dir)</code>	Führt ein Programm aus und liefert ein Objekt zurück, das diesen Prozess beschreibt. Während die erste Variante eine einzelne Kommandozeile als Parameter übergeben bekommt, wird bei den anderen ein Array übergeben, das zum Argument <code>args[]</code> der Methode <code>main</code> wird. In <code>envp</code> werden die Umgebungsvariablen gespeichert, <code>dir</code> enthält das Startverzeichnis für das neue Programm.
<code>long freeMemory() long maxMemory() long totalMemory()</code>	Liefert den freien Speicher, den maximalen Speicher bzw. den gesamten Speicher der JVM zurück.
<code>void halt(int status)</code>	Mit <code>halt()</code> wird die aktuell laufende Java-VM ähnlich wie bei <code>exit()</code> beendet, ohne die Objekte korrekt zu schließen.
<code>void gc()</code>	Die Methode ruft den Garbage Collector auf. Der explizite Aufruf des Garbage Collectors ist nur notwendig, wenn der Zeitpunkt zur Speicherfreigabe kontrolliert werden soll. Nicht benötigter Speicherplatz wird ansonsten vom Java-Laufzeitsystem mit dem Garbage Collector bei Bedarf freigegeben.
<code>void addShutdownHook (Thread hook) void removeShutdownHook (Thread hook)</code>	Registriert einen Thread (siehe Abschnitt 22 "Threads", der ausgeführt wird, sobald die JVM terminiert bzw. macht die Registrierung rückgängig.
<code>void load(String filename) void loadLibrary(String libraryName) String mapLibraryName(String libName)</code>	Diese drei Methoden werden bei der Integration von Java und Maschinensprache (C) benutzt.

Tabelle 15.2: Methoden der Klasse `Runtime`

<code>destroy()</code>	löscht den gestarteten Prozess.
<code>int exitValue()</code> <code>int waitFor()</code>	liefert das Argument des Aufrufs der Methode <code>System.exit()</code> des gestarteten Prozesses zurück. Während <code>exitValue()</code> einen Fehler liefert, sofern der Prozess noch läuft, wartet <code>waitFor()</code> auf das Ende des Prozesses.
<code>InputStream ge»</code> <code>tErrorStream()</code> <code>InputStream ge»</code> <code>tInputStream()</code> <code>OutputStream ge»</code> <code>tOutputStream()</code>	liefert die Streams <code>System.err</code> , <code>System.out</code> und <code>System.in</code> des gestarteten Prozesses zurück.

Tabelle 15.3: Methoden der Klasse `Process`

15.6 Properties

Damit ein Programm hinreichend flexibel an geänderte Umgebungen angepasst werden kann, ist es im allgemeinen erforderlich, sich ändernde Informationen dem Programm zur Laufzeit übergeben zu können. Prinzipiell kann man diese Informationen natürlich auch über die Kommandozeile übergeben, allerdings werden die Kommandozeilen dadurch nicht gerade übersichtlicher. Auch die Umgebungsvariablen sind nur bedingt geeignet, weil sie für alle Applikationen auf einmal gelten.

Unter Microsoft Windows gibt es hierfür die System-Registry, diese Einrichtung ist allerdings nicht betriebssystemübergreifend verfügbar. Ein weiterer Nachteil der Registry ist die schlechte Transportierbarkeit der enthaltenen Informationen und die Schwierigkeit, Programme mit verschiedenen Konfigurationen starten zu können. Auch aus deswegen hat Microsoft bei der Einführung der .NET-Umgebung Wert auf die Möglichkeit der sog. XCOPY-Installation gelegt, bei der ein Programm lediglich durch Kopieren einiger Dateien installiert wird.

Java bietet mit der Klasse `Properties` eine auf allen Plattformen identisch nutzbare API an, mit denen sich solche Informationen übertragen lassen. Die Klasse `Properties` ist von der Klasse `Hashtable` abgeleitet, d. h. ein Objekt dieser Klasse verwaltet eine ganze Menge von Eigenschaften. Eine einzelne Eigenschaft besteht aus einem Schlüssel und einem Wert, beide sind Strings¹. Die Namen von Eigenschaften sind ähnlich wie Paketnamen durch '.' strukturiert. Eine einfache Methode der Namensbildung ist die Benutzung des qualifizierten Paketnamens (plus einem Namen für die Eigenschaft selbst).

Alle Property-Werte entspringen einer von drei Quellen. Zunächst gibt es die System-Properties, die von der JVM erzeugt werden. Dazu gehören der Name des aktuellen Verzeichnisses (`user.dir`), der Name des aktuellen Benutzers (`user.name`), das Pfadtrennzeichen (`file.separator`).

Als zweite Möglichkeit lassen sich beliebige System-Properties beim Start der JVM über die Kommandozeile übergeben. Diese System-Properties werden über die Option `-D` definiert, z. B.

¹Deshalb sollte man auf `Properties` nicht über die Methoden von `Hashtable` zugreifen.

```
java -Dde.myClass.Number=1234 de.MyClass .
```

Die dritte Möglichkeit zum Einlesen von Properties besteht darin, diese aus Dateien auszulesen. Dateien können in einem von zwei Formaten vorliegen:

- Als XML-Datei. Die Syntax dieser XML-Datei wird durch die DTD (<http://java.sun.com/dtd/properties.dtd>)¹⁹ beschrieben. Eine solche XML-Datei könnte wie folgt aussehen:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM
    "http://java.sun.com/dtd/properties.dtd">
<properties>
  <comment/>
  <entry key="java.class.version">50.0</entry>
  <entry key="line.separator">&#13;</entry>
  <entry key="os.version">8.3</entry>
  <entry key="java.version">1.6.0_07</entry>
</properties>
```

- Als Property-Datei. In einer Property-Datei steht in jeder logische Zeile ein Schlüssel und ein Wert, durch '=' oder ':' getrennt. Eine logische Zeile kann sich über mehrere reale Zeilen erstrecken, indem das Zeilenende durch ein
unmittelbar davor maskiert wird.

Zur Darstellung der Sonderzeichen Tabulator, Papiervorschub, Wagenrücklauf und Zeilenumbruch dienen die Sonderzeichen

t,

f,

r und

n. Darüberhinaus müssen die Zeichen #, !, =, ' ' und : durch ein vorangestelltes

maskiert werden, alternativ können auch die Unicode-Escapes z. B.

u0020 für das Leerzeichen benutzt werden. Die Zeichen # oder ! markieren den Rest der aktuellen Zeile als Kommentar.

Die Klasse `java.util.Properties` bietet u. a. die folgenden Methoden:

<code>Properties() Properties(Properties default)</code>	Konstruiert ein leeres Properties-Objekt bzw. eines mit den angegebenen Voreinstellungen.
<code>String getProperty(String key) String getProperty(String key, String default)</code>	Sucht nach der mit key bezeichneten Eigenschaft. Falls key nicht gefunden wird, gibt die zweite Variante den angegebenen Standardwert zurück.
<code>void list(PrintStream out) void list(PrintWriter out)</code>	Gibt den Inhalt aller Properties auf der angegebenen Datei aus
<code>void load(InputStream inStream) void loadFromXML(InputStream in)</code>	Lädt die Property-Menge von der angegebenen Property-Datei oder XML-Datei
<code>Object setProperty(String key, String value)</code>	Fügt das Paar aus Schlüssel und Wert-Paar über den Hashtable-Aufruf put in die Property-Menge ein.
<code>Enumeration<String> getPropertyNames()</code>	Listet die Namen aller Schlüssel auf.
<code>void store(PrintStream out, String comment) void store(PrintWriter out, String comment) void storeToXML(PrintStream out, String comment)</code>	Schreibt Schlüssel und Werte dieser Property-Menge in eine Datei als Paare von Schlüssel und Wert oder als XML.

Beachten Sie bitte, dass Properties von Hashtables abgeleitet sind. Normalerweise sollten alle Schlüssel und Werte Strings sein. Durch die direkte Benutzung der entsprechenden Hashtable-Methoden können allerdings beliebige Werte eingeschleust werden, so dass `getProperty()`, `setProperty()` und `propertyNames()` beliebige Werte zurückgeben.

Im Normalfall entnimmt das Java-Programm den Namen der Property-Datei den Kommandozeilenargumenten wie im folgenden Beispiel:

```

1 import java.io.FileInputStream;           13         System.out.println(key + "=" +
2 import java.util.Enumeration;           14             properties.getProperty(key));
3 import java.util.Properties;             15     }
4 class PropertyDemo {                     16     properties.remove("Pass");
5     public static void main(String []args) 17     properties.setProperty("User", "X92Mue-
6         throws java.io.IOException {      18     properties.setProperty("Home", "/home/
7         Properties properties = new Properties() 19     System.out.println("Bearbeitete Daten:
8         properties.load(new FileInputStream(ar 20     properties.storeToXML(System.out, "Kom
9         Enumeration enumeration = properties.p 21     }
10        ropropertyNames();                22     }
11        System.out.println("Eingelesene Daten: 23
12        ");                                24
13        while( enumeration.hasMoreElements() ) {
14            String key=(String)enumeration.nextElement();

```

Programm 15-5: Beispielanwendung für Properties

Das obenstehende Programm produziert aus den Eingabedaten:

Pass=Geheimes

Kennwort

User= Benutzer

Database=MyDatabase:MySchema

die folgende Ausgabe:

Eingelesene Daten:

Pass=Geheimes Kennwort

Database=MyDatabase:MySchema

User=Benutzer

Bearbeitete Daten:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

```
<!DOCTYPE properties SYSTEM
```

```
"http://java.sun.com/dtd/properties.dtd">
```

```
<properties>
```

```
<comment>Kommentar</comment>
```

```
<entry key="Home">/home/muellerf</entry>
```

```
<entry key="Database">MyDatabase:MySchema</entry>
```

```
<entry key="User">X92MuellerF</entry>
```

```
</properties>
```