RRZN-Handbuch JAVA, 2. Band, 3. Auflage
(Das Inhaltsverzeichnis erstreckt sich auch über den 1. Band!)

Inhaltsverzeichnis

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 dazugehörenden 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 Zeichenfolgen-Bearbeitung 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,

`java.lang`	Paket für fundamentale Programmieraufgaben in Java (Typkonvertierung, String-Bearbeitng, mathematische Routinen). In diesem Kapitel werden die Klassen `System` (Abschnitt 15.4 "Die Klasse System", 11) sowie `String` und `StringBuffer` (Abschnitt 15.2 "String-Klassen", 5) und `Math` 15.3 "Klasse Math", 9, behandelt. Die Klassen aus dem Paket `java.lang` sind in jedem Java-Programm ohne `import`-Anweisung verfügbar.
`java.util`	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.
`javax.swing`	Das Swing-Toolkit ist die Bibliothek für grafische Benutzerschnittstellen. Die Programmierung mit dem Paket `java.swing` wird ausführlich in Kapitel 16 "Grafische Benutzeroberflächen" ab Seite 19 behandelt.
`java.net`	Net(working) enthält Klassen für die Netzwerk-Programmierung in Java. Die Programmierung mit den Klassen aus `java.net` wird in Kapitel 20 "Netzwerkprogrammierung", Seite 107 gezeigt.
`java.applet`	Das Paket `java.applet` enthält die Klasse `Applet`, 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 Editon (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.

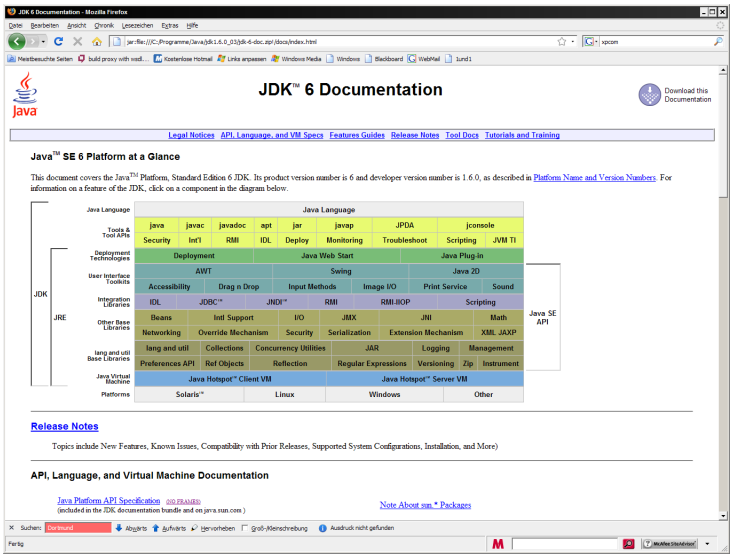
Bezüglich der Programmierung sind eigene Klassen und Methoden denjenigen aus API-Bibliotheken völlig gleichberechtigt und können genauso genutzt werden, d. h. es werden z. B. Instanzen der API-Klassen erstellt und Methoden der Klassen aufgerufen.

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.

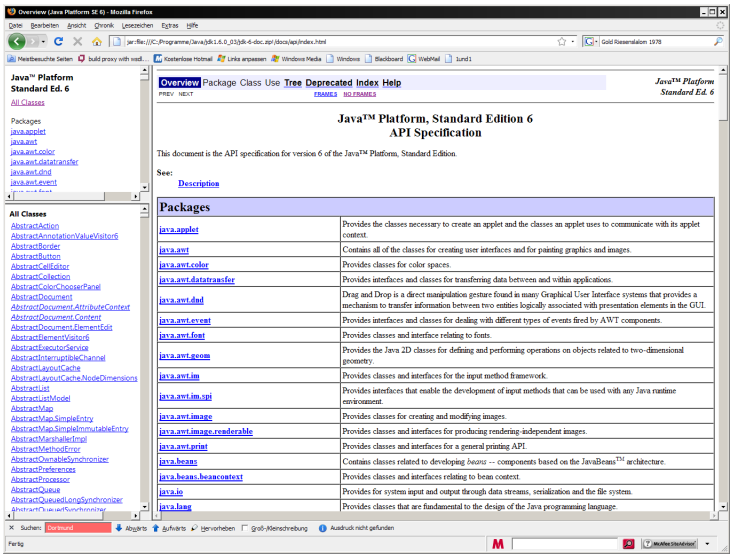
Die Dokumentation zur JDK-API gehört nicht zum Umfang der JDK-Installation und muss separat installiert werden. Hinweise zur Installation der Dokumentation enthält Abschnitt 23.7 "Dokumentation zum JDK", Seite 172.

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

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. Ingesamt 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 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 intern

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

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)

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

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

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

aufrufbar sind.

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

`double sin(double a)`
`double cos(double a)`
`double tan(double a)`
	Liefern den Sinus, Cosinus, und Tangens. Das Argument wird als Zahl im Bogenmass interpretiert. 90 Grad entsprechen `PI/2`.

`double asin(double a)`
`double acos(double a)`
`double atan(double a)`
	Liefern den Arkus-Sinus (Wert `0` bis `PI`), Arkus-Cosinus (Wert von `-PI/2` bis `PI/2`), und Arkus-Tangens (Wert von `-PI/2` bis `PI/2`). Das Argument für `acos` und `asin` muss eine Zahl zwischen `-1` und `1` sein.

`double toRa» dians(double deg) double to» Degrees(double grad)`	Konvertiert eine Radiant-Angabe in Grad bzw. umgekehrt.

`double cos(double a)`
`double sin(double a)`
`double tan(double a)`
	Liefern den trigonometrischen Cosinus, Sinus und Tangens.

`double log(double a)` `double log10(double a)`	Liefern Logarithmus zur Basis `e` bzw. `10`.
`double exp(double x)` `double pow(double a, double x)`	Liefern Exponentialwert zur Basis `e` bzw. zur Basis `a`.
`double sqrt(double a)`	Liefert die Quadratwurzel.
`double rint(double a)` `double floor(double a) double ceil(double a)`	Liefern zu einen `double`-Wert den gerundeten Wert als `double`. Die Methode `floor()` rundet ab, `ceil()` rundet auf und `rint()` rundet kaufmännisch).
`long round(double a)` `long round(float a)`	Liefern zu einen `double`- oder `float`-Wert, den gerundeten Wert als `long`. Beide Methoden runden kaufmännisch).
`double random()`	Liefert eine Pseudo-Zufallszahl im Bereich von `0` bis kleiner `1.0`.
`double abs(double x) double max(double x, double y) double min(double x, double y)`	Liefern den Absolutbetrag, das Maximum oder Minimum ihrer Argumente zurück. Statt `dou» ble` kann auch `float`, `int` oder `long` verwendet werden, dann ändert sich der Rückgabetyp entsprechend.
`int signum(double x)` `int signum(float x)`	Liefern das Vozeichen 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{
2    public static void main(String[] args){
3      System.out.println("\nQuadratwurzel ");
4      double w;
5      w = Math.sqrt(2);
6      System.out.println(" sqrt(2) "+w);
7      System.out.println("\nZufallszahlen und Vergleich");
8      double ra,rb;
9      ra = Math.random();
10     rb = Math.random();
11     System.out.println("Zufallswerte: " + ra + rb);
12     System.out.println("Minimum: " + Math.min(ra,rb));
13     System.out.println("\nTrigonometrische Funktionen");
14     double rad;
15     for(int i=1;i<=16;i*=2){
16       rad = Math.PI/i;
17       System.out.println(" sin(PI/"+i+") "+Math.sin(rad));
18       System.out.println(" asin("+Math.sin(rad)+") "+
19         "PI/"+Math.PI/(Math.asin(Math.sin(rad))));
20     }
21   }
22 }
23
```

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 Modifkator `static` ist in der folgenden Beschreibung nicht mit aufgeführt:

`void arraycopy(Object src, int srcPos, Object dst, int dstPos, int len)`	Kopiert Elemente von einem Array in ein anderes Array.
`long currentTimeMil» lis() long nanoTime()`	Die Methode liefert die aktuelle Systemzeit in Milli- bzw. Nanosekunden.
`void exit(int status)`	Mit `exit()` wird die aktuell laufende Java-VM beendet. Als `status` wird konventionsgemäß der Wert `0` übergeben, wenn das Programm in einem fehlerfreien Zustand beendet wird.
`void gc()`	Die Methode ruft den Garbage Collectorauf, die identische Methode ist `Runti» me.getRuntime().gc()`.
`void load(String fi» leName) void loadLibra» ry( String libraryName) String mapLibraryName( String libName)`	Diese drei Methoden laden in Maschinensprache erstellte Bibliotheken, die vollständige Beschreibung findet sich in der Klasse `Run» time`.

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

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.

`System.in`	Standard-Eingabestrom
`System.out`	Standard-Ausgabestrom
`System.err`	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 Modifkator `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].

JNI Java Native Interface
GC Garbage Collector

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:

`int availableProces» sors()`	Liefert die Zahl der vorhandenen Prozessoren.
`Process exec(String command) Process exec(String cmdarray[]) Process exec(String cm» darray[],String envp[]) Process exec(String cm» darray[],String envp[], File dir)`	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 `args[]` der Methode main wird. In `envp` werden die Umgebungsvariablen gespeichert, `dir` enthält das Startverzeichnis für das neue Programm.
`long freeMemory() long maxMemory() long total» Memory()`	Liefert den freien Speicher, den maximalen Speicher bzw. den gesamten Speicher der JVM zurück.
`void halt(int status)`	Mit `halt()` wird die aktuell laufende Java-VM ähnlich wie bei `exit()` beendet, ohne die Objekte korrekt zu schließen.
`void gc()`	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.
`void addShutdownHook (Thread hook) void removeShutdownHook (Thread hook)`	Registriert einen Thread (siehe Abschnitt 22 "Threads", der ausgeführt wird, sobald die JVM terminiert bzw. macht die Registrierung rückgängig.
`void load(String fi» leName) void loadLibra» ry(String libraryName) String mapLibraryNa» me(String libName)`	Diese drei Methoden werden bei der Integration von Java und Maschinensprache (C) benutzt.

Tabelle 15.2: Methoden der Klasse `Runtime`

`destroy()`	löscht den gestarteten Prozess.
`int exitValue()` `int waitFor()`	liefert das Argument des Aufrufs der Methode `System.exit()` des gestarteten Prozesses zurück. Während `exitValue()` einen Fehler liefert, sofern der Prozess noch läuft, wartet `waitFor()` auf das Ende des Prozesses.

`InputStream getErrorStream()`
`InputStream getInputStream()`
`OutputStream getOutputStream()`
	liefert die Streams `System.err`, `System.out` und `System.in` 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 Strings[1]. 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.

[1] Deshalb 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:

`Properties() Pro» perties(Properties default)`	Konstruiert ein leeres Properties-Objekt bzw. eines mit den angegebenen Voreinstellungen.
`String getProper» ty(String key) String getProperty(String key, String default)`	Sucht nach der mit key bezeichneten Eigenschaft. Falls key nicht gefunden wird, gibt die zweite Variante den angegebenen Standardwert zurück.
`void list(PrintStream out) void list(PrintWriter out)`	Gibt den Inhalt aller Properties auf der angegebenen Datei aus
`void load(InputStream inStream) void load» FromXML(InputStream in)`	Lädt die Property-Menge von der angegebenen Property-Datei oder XML-Datei
`Object setProper» ty(String key, String value)`	Fügt das Paar aus Schlüssel und Wert-Paar über den Hashtable-Aufruf put in die Property-Menge ein.
`Enumeration<?> proper» tyNames()`	Listet die Namen aller Schlüssel auf.
`void store(OutputStream out, String com» ments) void store» ToXML(OutputStream os, String comment)`	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;
 2 import java.util.Enumeration;
 3 import java.util.Properties;
 4 class PropertyDemo {
 5   public static void main(String []args)
 6       throws java.io.IOException {
 7     Properties properties = new Properties>
✂ ();
 8     properties.load(new FileInputStream(ar>
✂ gs[0]));
 9     Enumeration enumeration = properties.p>
✂ ropertyNames();
10     System.out.println("Eingelesene Daten:>
✂ ");
11     while( enumeration.hasMoreElements() ) {
12       String key=(String)enumeration.nextE>
✂ lement();
13       System.out.println(key + "=" +
14         properties.getProperty(key));
15     }
16     properties.remove("Pass");
17     properties.setProperty("User", "X92Mue>
✂ llerF");
18     properties.setProperty("Home", "/home/>
✂ muellerf");
19     System.out.println("Bearbeitete Daten:>
✂ ");
20     properties.storeToXML(System.out, "Kom>
✂ mentar");
21   }
22 }
23
24
```

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>
```