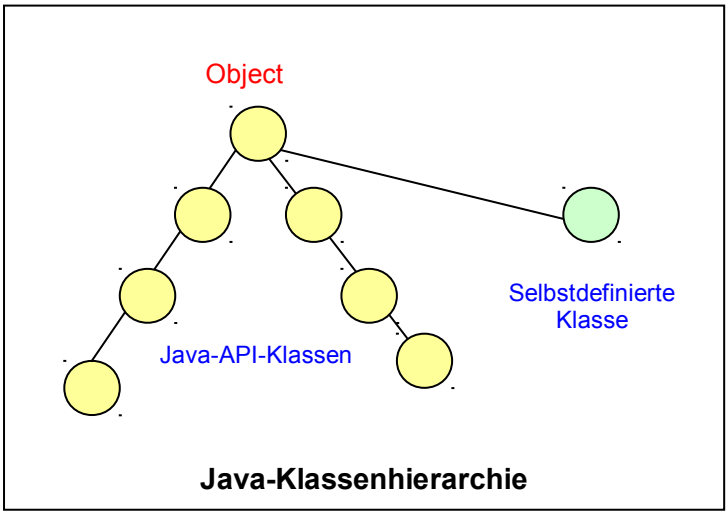
Systemklassen-Bibliothek (API)

Java verfügt über eine umfangreiche Sammlung von vorgefertigten Klassen (API-Bibliothek), in denen sowohl die gesamte Objekthierarchie als auch häufig gebrauchte Funktionsklassen implementiert sind. Die API-Bibliothek ist hierarchisch aufgebaut und jede Klasse hat darin ihren bestimmten Platz. Da Java eine strikt objektorientierte Sprache ist, beruht auch die API-Bibliothek auf dem Klassenmodell, im Gegensatz bspw. zur Windows-API-Bibliothek, die auch rein prozedurale Funktionen anbietet.

Die API-Bibliothek kann man sich als einen auf den Kopf gestellten Baum vorstellen, von dessen Wurzel (Object) alle Zweige ausgehen und abstammen. Auch selbstgeschriebene Klassen werden automatisch in ihre hierarchische Ordnung eingebunden und erben bestimmte Grundfunktionalitäten von der obersten Klasse Object.
Der Java-Compiler und der Java-Interpreter benutzen die API-Bibliothek, um die Grund- und Objektdatentypen zu erstellen, Nutzklassen zu definieren und deren Funktionalitäten aufzurufen. Ohne die API-Bibliothek könnte kein noch so kleines Java-Programm kompiliert oder ausgeführt werden. Der Compiler und der Interpreter selbst sind recht kleine Programme (wenige kB), wohingegen die API-Bibliothek komplex und umfangreich ist. Sie liegt vor in einem komprimierten Archiv (mehrere MB).



Java-Klassenhierarchie

Die verschiedenen Java-Versionen, die seit den ersten Anfängen entwickelt wurden, unterscheiden sich auch (und vor allem) durch den Umfangreichtum ihrer API-Bibliothek. Die neuesten API-Versionen haben zusätzliche Klassen und Funktionalität anzubieten, die die vorangegangenen Versionen ergänzen und erweitern. Grundsätzlich sind die API-Versionen abwärtskompatibel, das heißt sie bieten mindestens den Funktionsumfang aller Vorgängerversionen, es kommen aber neue und erweiterte Klassen und Funktionen hinzu. Es gibt stets auch einige Klassen, die zwar noch in der Hierarchie existieren, aber nicht mehr benutzt werden sollten, weil sie sich als fehlerhaft oder ungeeignet erwiesen haben (`deprecated` = verworfen).

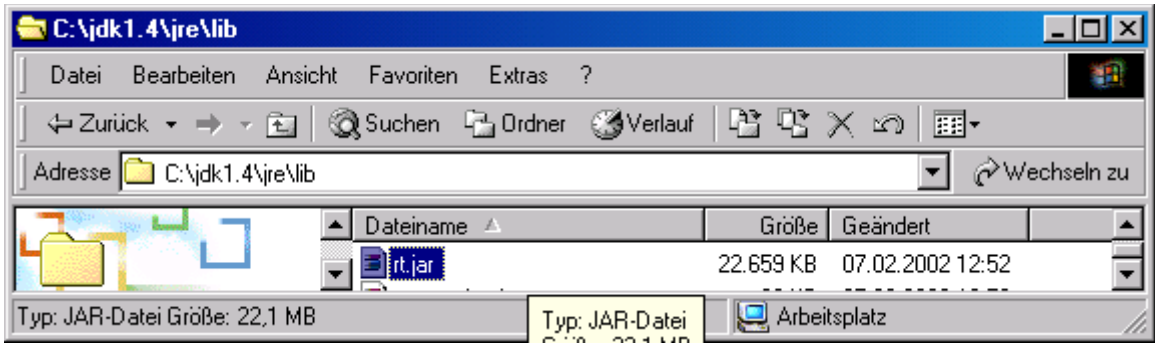
Runtime-Archiv (komprimierte API-Bibliothek)

Damit der Java-Compiler und der Java-Interpreter jederzeit Zugriff auf die API-Bibliothek haben, ist sie an einem festgelegten Ort der SDK-Verzeichnisstruktur verankert. Dieser Pfad muss nicht durch ausdrückliche Ordnerangabe bekannt gemacht werden, da er im Compiler und Interpreter fest eincodiert ist. Die API-Bibliothek liegt komprimiert in einem speziellen Java-Archiv in dem Unterordner `jre/lib` des Java-Installationsverzeichnisses vor. Der Name des Archivs ist `rt.jar`, wobei rt die Abkürzung für Runtime (Laufzeitbibliothek) ist.

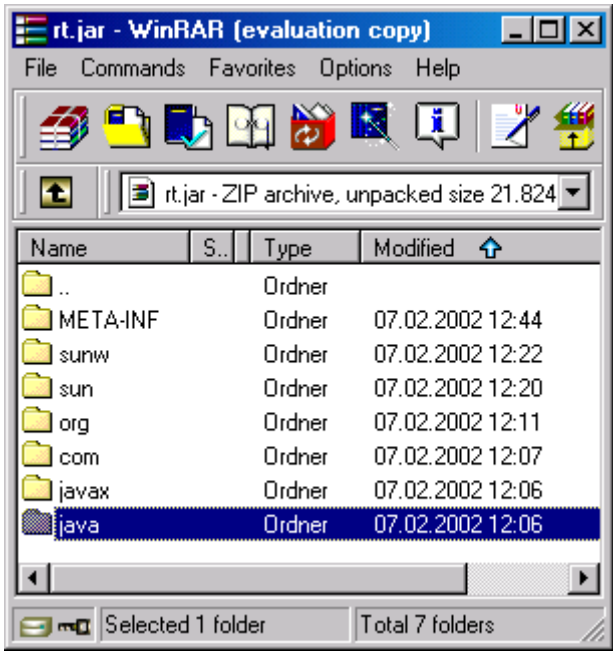
```
JAVA-Installationsverzeichnis \ jre \ lib \ rt.jar

z.B.: c:\ jdk1.4 \ jre \ lib \ rt.jar
```

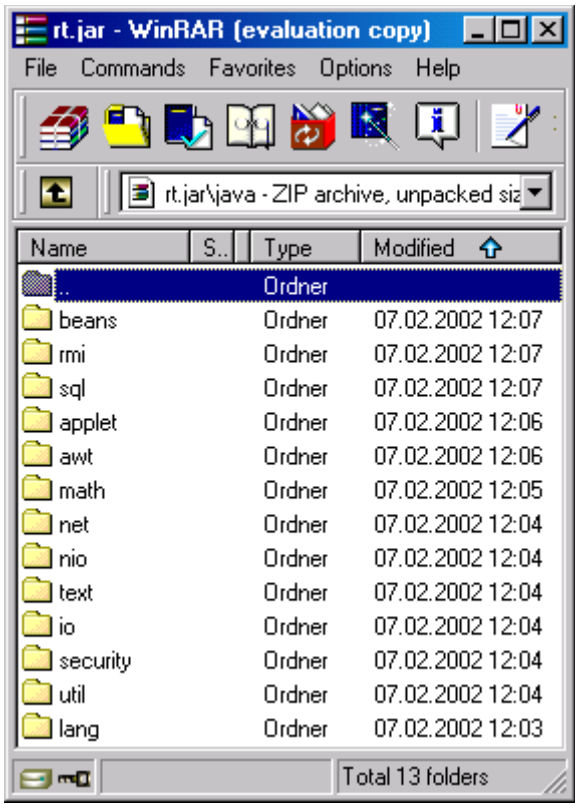
Bei MS-Windows zum Beispiel:



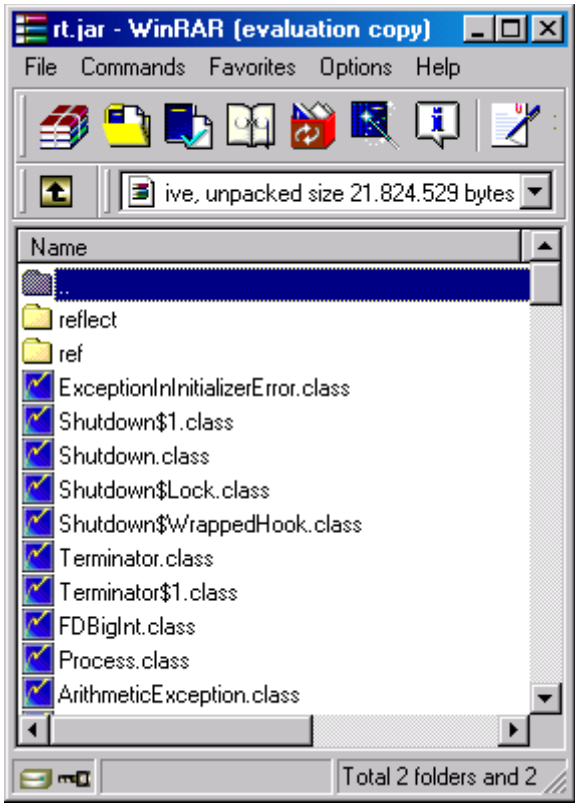
Das rt-Archiv kann mit jedem ZIP-Programm (WinZip oder WinRAR) geöffnet werden, wodurch die interne Verzeichnisstruktur des Archivs erkennbar wird:



Die am häufigsten benötigten Klassen liegen in dem Ordner namens java:



Es wird erkennbar, dass das rt-Archiv in Ordnern strukturiert ist, die den Namen ihrer Basis-Funktionalität tragen. So sind beispielsweise in dem Unterordner `java/lang` (language) alle Standard-Klassen vorhanden, die standardmäßig von jedem Java-Programm benötigt werden.



Einbinden von API-Klassen in einem Java-Programm

Wenn der Compiler oder Interpreter eine API-Klasse verwenden soll, die nicht zum Standard-Verzeichnis (java/lang) gehört, muss ihr exakter Verzeichnisort ausdrücklich bekannt gemacht werden. Dies kann auf zweierlei Art und Weise geschehen.

1. Angabe des Klassenpfads unmittelbar beim Aufruf

Der Pfad zum Verzeichnisort wird innerhalb des Programms direkt bei der Verwendung der benötigten Klasse angegeben. Ein Verzeichnis oder eine Klasse innerhalb der Ordnerstruktur der API-Bibliothek wird abgebildet durch die Verwendung des in Java üblichen Punktoperators anstelle des Schrägstriches. So kann in einem Java-Programm beispielsweise die Klasse Vector verwendet werden:

```
. . .
java.util.Vector vec = new java.util.Vector(100);
. . .
```

Die Pfadangabe muss dann natürlich bei *jeder* Verwendung der API-Klasse erfolgen.

2. Abgekürzte Pfadangabe mit import

Wenn in einem Programm eine API-Klasse häufig verwendet werden soll, dann kann am Beginn des Quellcodes (vor allem anderen Code) die Anweisung import mit dem Pfad und Namen der API-Klasse angegeben wird.

```
import <Pfad>;
z.B: import java.util.Vector;
```

Die Anweisung `import` macht den Pfad zu der API-Klasse bekannt, sodass sie später auch ohne ausdrückliche Pfadangabe gefunden wird. Die Klasse kann dann im Java-Quellcode jederzeit direkt angeführt werden.

```
. . .
Vector vec = new Vector(100);
. . .
```

Die `import`-Anweisung ist nicht zu verwechseln mit der von der Programmiersprache C / C++ bekannten Anweisung include. Es werden nur Pfadangaben vereinbart und kein Quellcode oder Text eingebunden!