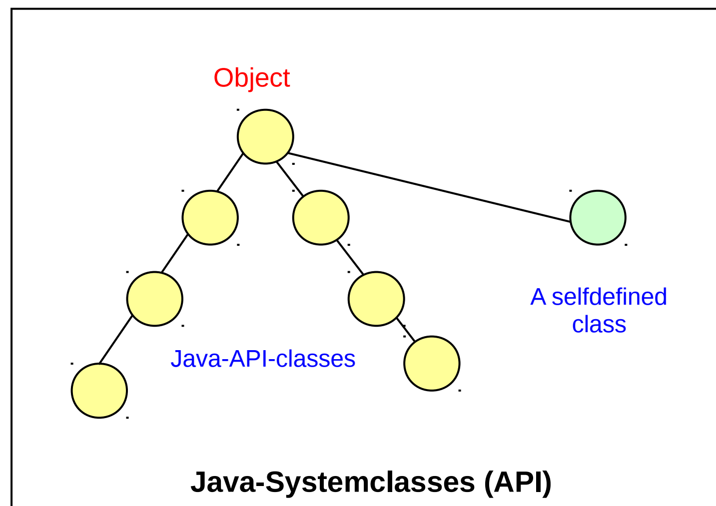
System class library (API)

Java includes an extensive collection of predefined classes (Java API library), in which both the entire object hierarchy and frequently used function classes are implemented. This API library is organized in strict hierarchical order, and each class has its specific place in it. Since Java is an object-oriented language, the API library too is based on its classmodel, in contrast to the Windows API library (WinAPI) that offers purely procedural (non-objectoriented) functioncalls.

The Java API library can be thought of as an upside-down tree, from its root (class *Object*) descend all the branches. Also selfwritten classes are automatically included in this hierarchical order and inherit certain basic functionalities and behaviour from the root class *Object*.

The Java compiler and the Java interpreter using the API library to create the basic data types and object to define Nutzklassen and invoke its functionalities. Without the API library no matter how small Java program could be compiled or executed. The compiler and the interpreter itself are quite small programs (a few kB), whereas the API library is complex and extensive. It is available in a compressed archive (several megabytes).



Java-Systemclasses (API)

The different versions of Java that have been developed since its beginning differ also (and especially) by the extent of their API library. The latest API versions offer additional classes and functionality that complement and extend the previous versions. Basically, the API versions are backward compatible, meaning that they provide at least the functionality of all previous versions, but adding new and advanced classes and functions. There are always a few classes that still exist in the hierarchy, but should not be used because they have proved to be defective or unsuitable (`deprecated` = rejected).

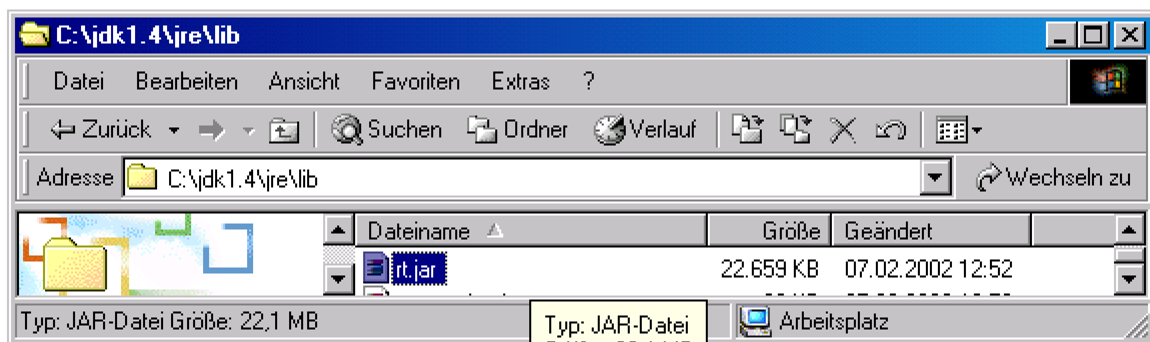
Runtime archive (compressed API Library)

Thus, the Java compiler and the Java interpreter at any time have access to the API library, it is anchored to a fixed location of the SDK directory structure. This path must not be made known through explicit indication folder because it is firmly encoded in the compiler and interpreter. The API library is compressed in a special Java archive subfolder in the `jre / lib directory` of the Java installation directory before.The name of the archive is `rt.jar,` where rt is the abbreviation for runtime (runtime library) is.

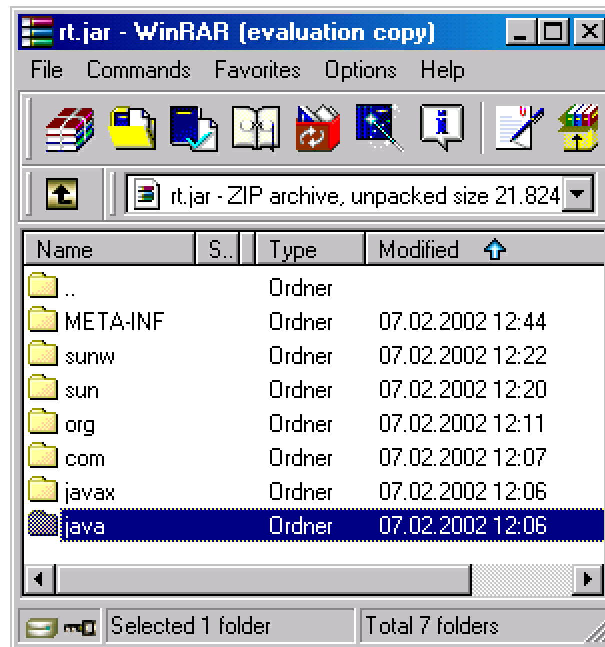
```
JAVA-Installation:  \ jre \ lib \ rt.jar

eg. c:\ jdk1.4 \ jre \ lib \ rt.jar
```

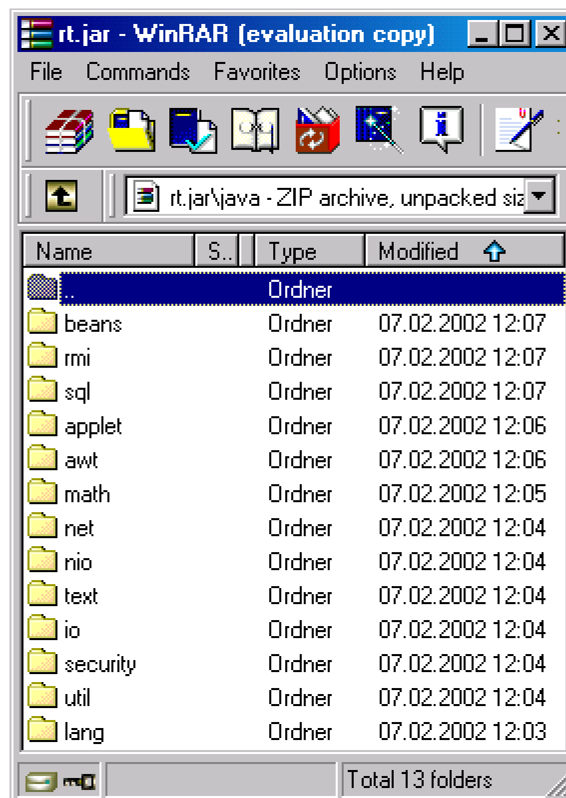
In MS-Windows, for example:



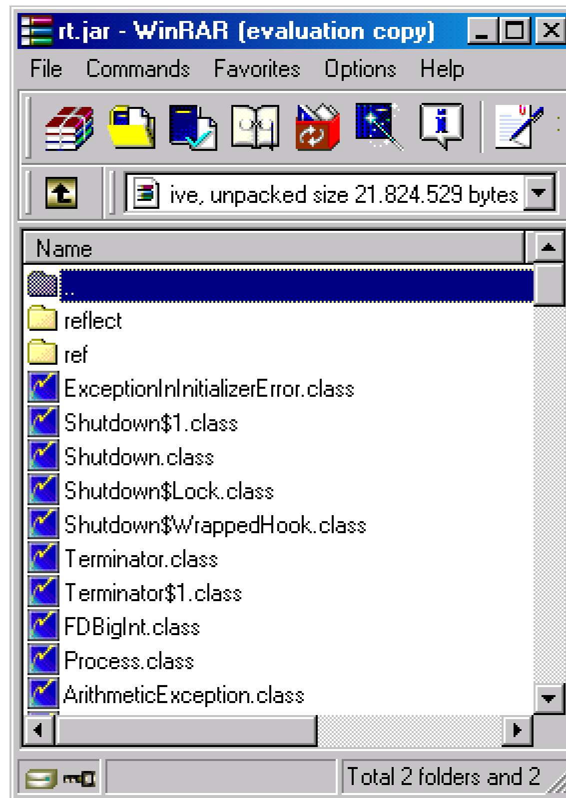
The rt-archive can be opened with any zip utility (WinZip or WinRAR), whereby the internal directory structure of the archive is visible:



The most commonly used classes are located in the folder named java:



It is obvious that the rt-archive is structured in folders that are named based on their functionality. Thus, all standard classes are available, for example, in the subfolder `java / lang` (language) that are needed by every standard Java program.



Integration of API classes in a Java program

When the compiler or interpreter has to use an API class that does not belong to the default directory (*java/lang*), its exact directory location must be made explicitly known. This can happen in two ways.

1. The class path directly

The path to the directory location is specified in the program directly with the use of the required class. A directory or a class within the folder structure of the library API is mapped by using the usual dot operator in Java instead of the slash.Thus, in a Java application, for example, the class vectors are used:

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

The path must be given naturally with *each* use of the API class.

2. Abbreviated path with import

If an API class is often used in a program, then at the beginning of the source code (before any other code) use the import statement with the path and name of the API class is specified.

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

The `import` statement makes the path to the API class known, so that it is later found without explicit path.The class can then be added at any time directly in the Java source code.

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

The `import` statement is not to be confused with the well-known statement of the programming language C / C++ ***include*** insofar as it defines only paths and no source code.