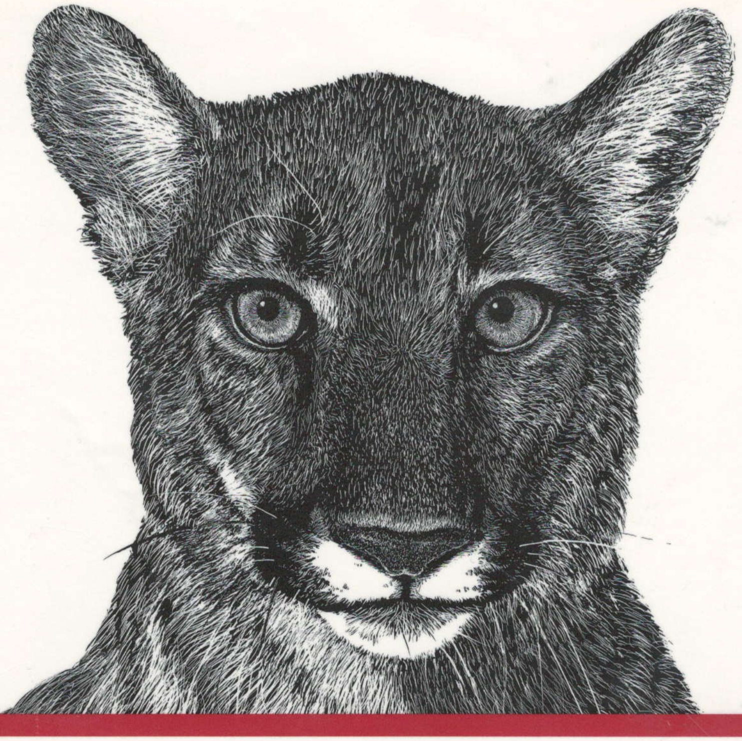

JAVA™ FOUNDATION CLASSES

IN A NUTSHELL

A Desktop Quick Reference

O'REILLY®

David Flanagan

ated a JAR file like this, you can tell a web browser about it with the TML tags:

```
ARCHIVE="myapplet.jar" CODE="myapplet.class" WIDTH=400 HEIGHT=200>
>
```

E attribute does not replace the CODE attribute. ARCHIVE specifies where files, but CODE is still required to tell the browser which of the files in is the applet class file to be executed. The ARCHIVE attribute may actu- a comma-separated list of JAR files. The web browser or applet viewer ese archives for any files the applet requires. If a file is not found in an wever, the browser falls back upon its old behavior and attempts to from the web server using a new HTTP request.

ers introduced support for the ARCHIVE attribute at about the same time 1 was introduced. Some Java 1.0 browsers do not recognize ARCHIVE re ignore it. If you want to maintain compatibility with these browsers, make your applet files available in an unarchived form, in addition to ficient archived form.

plets with the Java Plug-in

a-enabled web browser encounters an <APPLET> tag, it starts up its ava VM, downloads the class files that implement the applet, and starts em. This approach has run into difficulties because web browser not synchronized with releases of new versions of Java. It was quite a he release of Java 1.1 before commonly used browsers supported this he language, and there are still quite a few browsers in use that sup- va 1.0. It is not at all clear when, or even if, browsers will include sup- Java 2 platform. Furthermore, because of the lawsuit between Sun and he future of integrated Java support in the popular Internet Explorer r is questionable.

asons, Sun has produced a product called the Java Plug-in. This prod- a VM that acts as a Netscape Navigator plug-in and as an Internet tiveX control. It adds Java 1.2 support to these browsers for the Win- olaris platforms. In many ways, Java support makes the most sense as sing the Java Plug-in may be the preferred method for distributing Java e future.

atch, however. To run an applet under the Java Plug-in, you cannot LET> tag. <APPLET> invokes the built-in Java VM, not the Java Plug-in. must invoke the Java Plug-in just as you would invoke any other Nav- in or Internet Explorer ActiveX control. Unfortunately, Netscape and ve defined different HTML tags for these purposes. Netscape uses the and Microsoft uses the <OBJECT> tag. The details of using these tags ing them in a portable way are messy and confusing. To help applet Sun distributes a special HTML converter program that you can run TML files. It scans for <APPLET> tags and converts them to equivalent <OBJECT> tags.

Consider the simple HTML file we used for the first applet example in this chapter:

```
<APPLET code="MessageApplet.class" width=350 height=125>
  <PARAM name="message" value="Hello World">
</APPLET>
```

When run through the HTML converter, this tag becomes something like this:

```
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
codebase=
"http://java.sun.com/products/plugin/1.2/jinstall-12-win32.cab#Version=1,2,0,0"
       WIDTH=350 HEIGHT=125>
  <PARAM NAME=CODE VALUE="MessageApplet.class" >
  <PARAM NAME="type" VALUE="application/x-java-applet;version=1.2">
  <PARAM NAME="message" VALUE="Hello World">

  <COMMENT>
    <EMBED type="application/x-java-applet;version=1.2"
           pluginspage=
            "http://java.sun.com/products/plugin/1.2/plugin-install.html"
           java_CODE="MessageApplet.class"
           WIDTH=350 HEIGHT=125 message="Hello World">
    </EMBED>
  </COMMENT>
</OBJECT>
```

When Navigator reads this HTML file, it ignores the <OBJECT> and <COMMENT> tags that it does not support and reads only the <EMBED> tag. When Internet Explorer reads the file, however, it handles the <OBJECT> tag and ignores the <EMBED> tag that is hidden within the <COMMENT> tag. Note that both the <OBJECT> and <EMBED> tags specify all the attributes and parameters specified in the original file. In addition, however, they identify the plug-in or ActiveX control to be used and tell the browser from where it can download the Java Plug-in, if it has not already downloaded it.

You can learn more about the Java Plug-in and download the HTML converter utility from *http://java.sun.com/products/plugin.*

Applet Security

One of the most important features of Java is its security model. It allows untrusted code, such as applets downloaded from arbitrary web sites, to be run in a restricted environment that prevents that code from doing anything malicious, like deleting files or sending fake email. The Java security model has evolved considerably between Java 1.0 and Java 1.2 and is covered in detail in *Java in a Nutshell*.

To write applets, you don't need to understand the entire Java security model. What you do need to know is that when your applet is run as untrusted code, it is subject to quite a few security restrictions that limit the kinds of things it can do. This section describes those security restrictions and also describes how you can attach a digital signature to applets, so that users can treat them as trusted code and run them in a less restrictive environment.

The following list details the security restrictions that are typically imposed on untrusted applet code. Different web browsers and applet viewers may impose erent security restrictions and may allow the end user to customize or elax the restrictions. In general, however, you should assume that your pplet are restricted in the following ways:

ed code cannot read from or write to the local filesystem. This means trusted code cannot:

- ad files
- t directories
- eck for the existence of files
- otain the size or modification date of files
- otain the read and write permissions of a file
- st whether a filename is a file or directory
- ite files
- lete files
- eate directories
- name files
- ad or write from FileDescriptor objects

ed code cannot perform networking operations, except in certain d ways. Untrusted code cannot:

- eate a network connection to any computer other than the one from ich the code was itself loaded
- en for network connections on any of the privileged ports with num- s less than or equal to 1,024
- cept network connections on ports less than or equal to 1,024 or from host other than the one from which the code itself was loaded
- multicast sockets
- ate or register a SocketImplFactory, URLStreamHandlerFactory, or tentHandlerFactory

d code cannot make use of certain system facilities. It cannot:

- the Java interpreter by calling System.exit() or Runtime.exit()
- wn new processes by calling any of the Runtime.exec() methods
- amically load native code libraries with the load() or loadLibrary() hods of Runtime or System

d code cannot make use of certain AWT facilities. One major restric- hat all windows created by untrusted code display a prominent visual n that they have been created by untrusted code and are "insecure."

This is to prevent untrusted code from spoofing the on-screen appearance of trusted code. Additionally, untrusted code cannot:

- Initiate a print job
- Access the system clipboard
- Access the system event queue

- Untrusted code has restricted access to system properties. It cannot call System.getProperties(), so it cannot modify or insert properties into the system properties list. It can call System.getProperty() to read individual properties but can read only system properties to which it has been explicitly granted access. By default, *appletviewer* grants access to only the following 10 properties. Note that user.home and user.dir are excluded:
  - java.version
  - java.class.version
  - java.vendor
  - java.vendor.url
  - os.name
  - os.version
  - os.arch
  - file.separator
  - path.separator
  - line.separator
- Untrusted code cannot create or access threads or thread groups outside of the thread group in which the untrusted code is running.
- Untrusted code has restrictions on the classes it can load and define. It cannot:
  - Explicitly load classes from the sun.* packages
  - Define classes in any of the java.* or sun.* packages
  - Create a ClassLoader object or call any ClassLoader methods
- Untrusted code cannot use the java.lang.Class reflection methods to obtain information about nonpublic members of a class, unless the class was loaded from the same host as the untrusted code.

l code has restrictions on its use of the `java.security` package. It

pulate security identities in any way

r read security properties

look up, insert, or remove security providers

ly, to prevent untrusted code from circumventing all of these restric-
, it is not allowed to create or register a `SecurityManager` object.

ets

let is loaded from the local filesystem, instead of through a network
browsers and applet viewers may relax some, or even many, of the
trictions. The reason for this is that local applets are assumed to be
thy than anonymous applets from the network.

applet security policies are also possible. For example, an applet
written so that it places fewer restrictions on applets loaded from an
rate network than on those loaded from the Internet.

plets

l the ability to attach a digital signature to a JAR file that contains an
ignature securely identifies the author or origin of an applet. If you
or or originating organization, you can configure your web browser
ver to run applets bearing that signature as trusted code, rather than
code. Such an applet runs without the onerous security restrictions
rusted applets. Java 1.2 platform actually allows the security policy to
l based on the origin of an applet. This means that an end user or
istrator may define multiple levels of trust, allowing fully trusted
n with all the privileges of a standalone application, while partially
s run with a reduced list of security restrictions.

of attaching a digital signature to an applet's JAR file is platform
Java 1.1, you use the *javakey* program. In Java 1.2, this program has
led by *jarsigner*. Netscape and Microsoft also provide their own digi-
rograms that are customized for use with their browsers.

f telling your web browser or applet viewer which digital signatures
vendor dependent, of course. In Java 1.1, you use *javakey* to spec-
atures are trusted. In Java 1.2, you use a different tool, *policytool*, to
l signatures and the security policies associated with them. See *Java*
or further details.

PART II

API Quick Reference

Part II is the real heart of this book: quick-reference material for the APIs that comprise the Java Foundation Classes. Please read the following section, *How To Use This Quick Reference*, to learn how to get the most out of this material.