CHAPTER 9

Platform Independent Development with Java

In previous chapters most of the discussion centered around GNU development tools and programming in C and C++ languages. This chapter is devoted to Java development and describes how to use some Java tools and a Java virtual machine.

As many of the readers already know, Java is a different language compared to other compile-and-execute languages in that it needs a Java virtual machine (JVM) to execute what is called *byte code*. Each platform has its own implementation of the Java virtual machine. There are many JVMs available on Linux platform in free and open source as well as commercial domains. Some virtual machines are also available for embedded Linux platforms that are very small in size and need little memory to execute. You can select a virtual machine based upon your requirements. Java virtual machines are built on Java Language Specifications (JLS). Information about JLS and books in downloadable format can be found at http://java.sun.com/docs/books/jls/.

This chapter provides some information about how to use different development tools on Linux. The most commonly available tools are Sun SDK and GNU `gcj` compiler. Kaffe and Jboss are other very popular open source development tools on Linux and these are introduced in this chapter. However please note that this chapter is not a tutorial on Java language and if you are new to the language, you get help from many other resources on the Internet as well as in print media. Some of the most popular include Sun Java SDK, IBM Java SDK, Kaffe, and `gcj`.

9.1 How Java Applications Work

Java is a sort of interpreted language. Any program you write in Java is first compiled to generate byte code. Depending upon type of output, the byte code may be in the form of a *Java application* or a *Java applet*. In the case of Java applications, you can directly invoke the Java virtual machine to execute it. If the output type is an applet, it is usually loaded into a browser for execution.

9.1.1 Java Compiler

The Java compiler is used to build Java applications or applets from source code files. In most of the cases, the compiler is invoked as `javac` command. It also uses standard class libraries that are usually part of the Java distributions.

9.1.2 Java Virtual Machine

The purpose of the Java virtual machine is to execute Java code. In most cases, the virtual machine is available as the command `java`. To run any application, you can just invoke the `java` command with the application name as the command line argument.

9.2 Kaffe

Kaffe is one of the popular freely available implementations of the Java virtual machine. You can download it from its web site http://www.kaffee.org. At the time of writing this book, version 1.0.6 is available from the web site. I am using RedHat 7.1 for the development of this book and this version is also part of the distribution. Most of the binaries are installed in `/usr/bin` directory. The following command displays version information for Kaffe:

```
[root@desktop Java]# java -version
Kaffe Virtual Machine
Copyright (c) 1996-2000
Transvirtual Technologies, Inc.  All rights reserved
Engine: Just-in-time v3   Version: 1.0.6
Java Version: 1.1
[root@desktop Java]#
```

Note that the actual Kaffe Virtual machine is present as `/usr/bin/kaffe` and `/usr/bin/java` is simply a shell script like the following to invoke Kaffe.

```
#! /bin/sh
# Pretend Kaffe is Java
prefix=/usr
exec_prefix=/usr
exec /usr/bin/kaffe ${1+"$@"}
```

Kaffe also include class libraries associated with the Virtual machine. However, Kaffe is not official Java, which is a registered trademark of Sun Microsystems. The open source nature of Kaffe provides you with information about the internal implementation and workings of the Java virtual machine.

Basic help on using the Kaffe virtual machine is available using the `java` or `kaffe` command without any argument. The `kaffe` command produces the following output.

```
[root@desktop /root]# kaffe
usage: kaffe [-options] class
Options are:
        -help                   Print this message
        -version                Print version number
        -fullversion            Print verbose version info
        -ss <size>              Maximum native stack size
        -mx <size>              Maximum heap size
        -ms <size>              Initial heap size
        -as <size>              Heap increment
        -classpath <path>       Set classpath
        -verify *               Verify all bytecode
        -verifyremote *         Verify bytecode loaded from
                                network
        -noverify               Do not verify any bytecode
        -D<property>=<value>    Set a property
        -verbosegc              Print message during
                                garbage collection
        -noclassgc              Disable class garbage
                                collection
        -v, -verbose            Be verbose
        -verbosejit             Print message during JIT
                                code generation
        -verbosemem             Print detailed memory
                                allcation statistics
        -nodeadlock             Disable deadlock detection
        -prof                   Enable profiling of Java
                                methods
        -debug *                Trace method calls
        -noasyncgc *            Do not garbage collect
                                asynchronously
        -cs, -checksource *     Check source against class
                                files
        -oss <size> *           Maximum java stack size
        -jar                    Executable is a JAR
  * Option currently ignored.
[root@desktop /root]#
```

9.3 The Jboss Java Development System

The Jboss is probably the most popular open source Java development products suite and it provides services similar to Sun's Java 2 Enterprise Edition (J2EE). You can download it from free from http://www.jboss.org. The best thing about Jboss is that it is open source and free for use. At the time of writing this book Jboss version 2.4.4 is the stable release. Since J2EE is a big platform and this is not a programming tutorial text, you will learn only how to install the Jboss package here.

Following is a step-by-step outline how to build Jboss on Linux system.

- Download the latest version `jboss-all.tgz` from its web site.
- Unpack it using the `tar zxvf jboss-all.tgz command`
- You must have Java SDK installed to build it and JAVA_HOME variable properly set. Use the "`export JAVA_HOME=/usr/java/j2sdk1.4.0/`" command if you have SDK version 1.4 installed. If you have any other environment, use appropriate command.
- Go to `jboss-all` directory.
- Run the following command and it should display a result like the one shown.

```
[root@conformix jboss-all]# ./build/build.sh -version
Ant version 1.4 compiled on September 3 2001
[root@conformix jboss-all]#
```

- Execute the `build/build.sh init` command.
- Run the `build/build.sh` command.
- Run Jboss using the following command. It will show a list of message as it starts different processes.

```
[root@conformix jboss-all]# ./build/build.sh run-jboss
Searching for build.xml ...
Buildfile: /opt/jboss-all/build/build.xml

_buildmagic:init:

_buildmagic:init:local-properties:

_buildmagic:init:buildlog:

configure:
     [echo] groups:  default
     [echo] modules:
jmx,common,system,j2ee,naming,management,server,security,messa
ging,pool,connector,cluster,admin,jetty,varia,jboss.net,iiop

init:

run-jboss:
     [echo] Starting JBoss (redirected /opt/jboss-all/build/
run.log)

run-jboss-check-os:

_buildmagic:init:

_buildmagic:init:local-properties:

_buildmagic:init:buildlog:

configure:
     [echo] groups:  default
     [echo] modules:
jmx,common,system,j2ee,naming,management,server,security,messa
ging,pool,connector,cluster,admin,jetty,varia,jboss.net,iiop

init:

run-jboss-unix:
```

If everything goes well, the startup will be successful. You can see a log of messages in the `build/run.log` file. You can also run Jboss using the `run.sh` script.

9.4 Java 2 SDK

Java SDK is distributed by Sun Microsystems. There are two main versions of the SDK. The *Standard Edition* or J2SE is introduced in this book. The *Enterprise Edition* or J2EE is the premier product and it is comparable to Jboss discussed in this book. Please note that neither of J2SE nor J2EE are open source products. You have a limited license to use these. Please go through the license terms and conditions if you are inclined towards using Java SDK.

At the time of writing this book, Version 1.4 of the SDK is available.

9.4.1 Java 2 SDK Standard Edition

The latest version of Java SDK released by Sun Microsystems is 1.4 at the time of writing this book. There are some new features added to this version. Some important features are listed here.

9.4.1.1 Non-blocking I/O

After a long waiting period, a non-blocking I/O library is finally added to the SDK as `java.nio` package. Non-blocking I/O is important when you don't want to create threads for each connection in large network application. Creation of threads is very expensive compared to non-blocking I/O where you can open many I/O streams and use mechanisms similar to `poll()` and `select()` which are available with C language since long. Using this mechanism one thread may handle several connections, for example in the case of a web server.

It is interesting to note that a non-blocking I/O library was available in the open source before it is included in SDK version 1.4. You can find more information about this open source implementation at http://www.cs.berkeley.edu/~mdw/proj/java-nbio/.

9.4.1.2 Security Enhancements

Many new features have been added for security enhancements to the SDK and some existing features are refined. These features include Java Certification Path API, Cryptography, Java Authentication and Authorization Service security extensions, GSS-API with Kerberos support and so on.

9.4.1.3 Web Services

J2SE version 1.4 lays foundation for XML-based web services. Now the XML is part of the core platform which allows building interoperable web applications and services.

Different building blocks for Java platform standard edition version 1.4 are shown in Figure 9-1 taken from the Sun Microsystems web site.

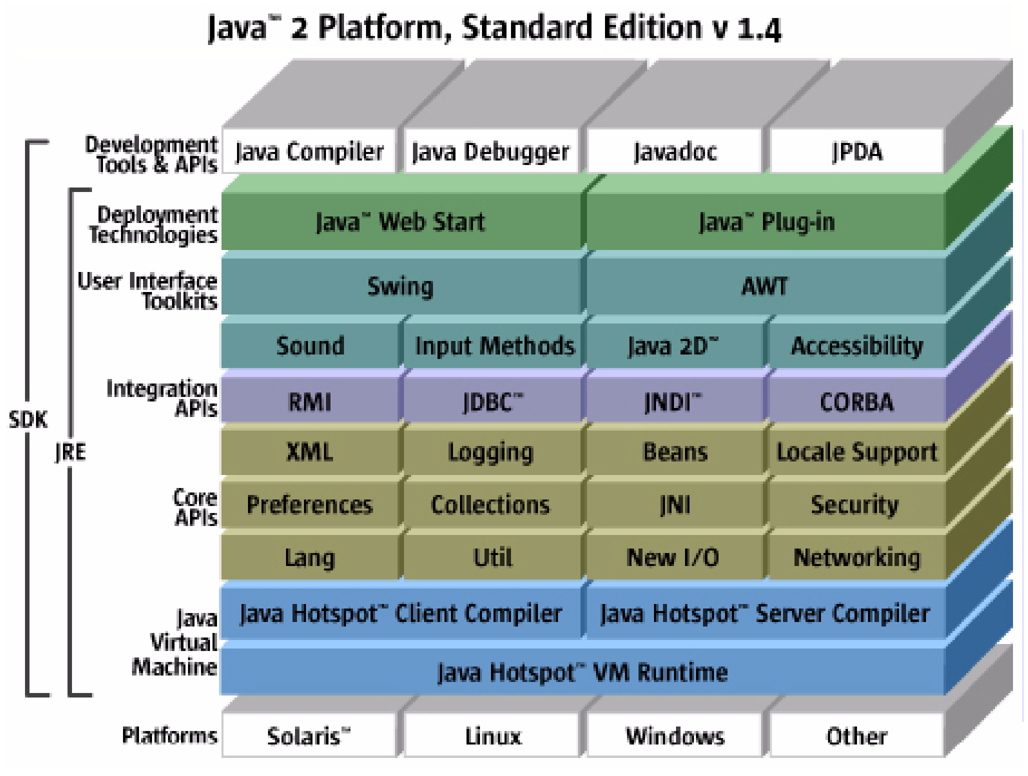

Figure 9-1 Building blocks of J2SE version 1.4.

9.4.2 Getting and Installing Java SDK from Sun

You can download the current version from http://java.sun.com. The file is available as `j2sdk-1_4_0-linux-i386-rpm.bin`. The best way is to copy this file to a temporary directory and then install it from there. I have copied this file into the `/download` directory where the first step is to make it executable using the following command:

```
chmod a+x j2sdk-1_4_0-linux-i386-rpm.bin
```

Then you can execute this file just by typing in the following command:

```
./j2sdk-1_4_0-linux-i386-rpm.bin
```

It will display the license agreement and you have to type in "yes" to agree to this license agreement. After that the self-extracting file will extract `j2sdk-1_4_0-fcs-linux-i386.rpm` file that you can install using the rpm package with the help of the following command:

```
rpm --install j2sdk-1_4_0-fcs-linux-i386.rpm
```

If you have any previous version of SDK installed on your system, you should un-install it first. To list files installed by the rpm package, you can use the following command.

```
rpm --query -l j2sdk-1.4.0-fcs|more
```

By default, files are installed under the `/usr/java/j2sdk1.4.0/` directory. The `/usr/java/j2sdk1.4.0/bin` directory contains all binary files for the SDK.

9.4.3 Creating jar Files

The `jar` utility is used to create Java-class archives. It can take multiple class files, compress these and put them into a single `.jar` file. The syntax for using the `jar` utility is similar to the `tar` program widely used on Linux and other UNIX systems. The following command creates a file `myclasses.jar` from two class file `first.class` and `second.class`.

```
jar cvf myclasses.jar first.class second.class
```

You can also add or extract classes from the existing `.jar` files. The following command adds `third.class` to `myclasses.jar` archive.

```
jar uf myclasses.jar third.class
```

The following command will extract `second.class` from the `myclasses.jar` file.

```
jar x second.class
```

You can also use wild cards with the `jar` command. If you use directory names, all files in that directory are added to the `.jar` file.

To list all files in a `.jar` file, use t command line option. The following command lists all files in `myclasses.jar` file.

```
jar tf myclasses.jar
```

9.5 Building Java Applications

A Java application is a compiled Java program that can be executed using a Java virtual machine. There are three basic steps to create a Java application. These steps are:

1. Create source code.
2. Compile it.
3. Run it using the virtual machine.

Let's see how you can go through these steps.

9.5.1 Creating Source Code File

Like any other programming language, Java source code files are created using an editor. These source code files contain Java instructions. The simplest source code file contains an instruction to print the message "Hello World", similar to your first C program. Please note that Java examples in this chapter are just to demonstrate how to use development tools.

The program that prints the "Hello World" message is shown below:

```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

The program must have a function `main()`, like C programs. The program may be saved as `HelloWorld.java` text file.

Note that you can use any editor to create the source code file. I would recommend using Emacs which you have already learned in Chapter 2. Emacs understands Java programming language syntax. There are other GUI editors and integrated development environment (IDE) packages that you can use to create Java source code.

9.5.2 Compiling Java Code

The Java compiler will compile the Java code into an executable. The following command creates an output file `HelloWorld.class` from source code file `HelloWorld.java`.

```
/usr/java/j2sdk1.4.0/bin/javac HelloWorld.java
```

Note that the output file name is the same as the name of the class that contains function `main()`.

You can check the type of the output file type by using the `file` command. The following command shows type of the output file as compiled class data.

```
[root@desktop /root]# file HelloWorld.class
HelloWorld.class: compiled Java class data, version 46.0
[root@desktop /root]#
```

Note that in large projects, there may be many source code files, class files and other object code files to generate the output executables. In such cases, you usually don't use the Java compiler on the command line to compile individual files. The best way to carry out such tasks is to use the `make` utility that you have already learned previously in this book.

9.5.3 Running Java Applications

Java applications run on the top of Java virtual machine. The "Hello World" application that you created in the previous section can be executed using the Java virtual machine.

```
[root@desktop /root]# java HelloWorld
Hello World!
[root@desktop /root]#
```

Note that when you run this application, you don't use `HelloWorld.class` on the command line. If you do, the Java virtual machine will not be able to execute the command. I have seen many messages in mailing lists and newsgroups from people who make this mistake and then wonder why they get error messages.

9.5.4 Using gcj to Build Java Applications

The `gcj` compiler is part of the GCC compiler suite. It is able to compile the Java language programs. One benefit of compiling Java programs with `gcj` is that you can create binary executable applications instead of Java byte code. These applications can be executed on Linux without the need of Java virtual machine. The downside is that you can execute applications built this way only on the platform for which you compiled. This is in contrast to the basic Java notion of compile once, run everywhere. Detailed information about using `gcj` is present at its web site http://gcc.gnu.org/java/.

You can use GNU debugger for programs compiled with `gcj`, which is a big advantage for many programmers who are familiar with `gdb`. To be able to compile a program with `gcj`, you must have `libgcj` available on your system. The `libgcj` provides class libraries, byte code interpreter and garbage collector at the runtime. Latest version of `libgcj` is available from ftp://sourceware.cygnus.com/pub/java/ and method of its compilation and installation is discussed in Chapter 3. Refer to Chapter 3 for a description of how to compile programs with `gcj` compiler.

9.6 Building Applets

Like any Java applications, applets are also compiled using the Java compiler. The only difference is that a Java applet is not like a standard application and can be run only inside a browser. Following is a source code for the applet `HelloWorld.java` that is used as an example here. The applet, when compiled with the Java compiler results in `HelloWorld.class` file, which is Java byte code.

```
import java.applet.*;
import java.awt.*;

public class HelloWorld extends java.applet.Applet {
    public void paint (java.awt.Graphics gc) {
  gc.drawString("Hello World", 100, 90);
    }
}
```

The main thing to note here is that compiling Java applets is not different in any way from compiling Java applications. The only difference between a Java application and a Java applet is in the source code.

9.7 Testing Applets with Netscape

As mentioned earlier, a Java applet must run inside a browser. For this you have to write an HTML file. The following HTML file incorporates the applet and can be loaded to a browser like Netscape.

```
<html>
<head>
</head>
<body>
   <applet code=HelloWorld width=400 height=300>
   </applet>
</body>
</html>
```

When you load this file in the browser, the browser will execute the Java code and you can see the “Hello World” message printed in the browser window. Applets can also be viewed by applet viewers.

9.8 Jikes for Java

Jikes is an open source compiler for Java from IBM. It is fully compatible with the Java Language Specifications. You can download it from its web site at http://oss.software.ibm.com/developerworks/opensource/jikes/. Jikes version 1.15 is available right now. You have to download and install it the usual way. Typical steps are:

- Download it from its web site.
- Uncompress it using `tar -zxvf jikes-1.15.tar.gz` command.
- Go to `jikes-1.15` directory.
- Run the configure script using the `./configure` command.
- Run the `make` command.
- Run the `make install` command.

The Jikes FAQ can also be found on the above-mentioned web site. Since Jikes is a command line compiler, it can be used in Makefiles for large projects. The compiler is invoked using the `jikes` command. The following command lists available options with Jikes:

```
[root@conformix jikes-1.15]# jikes

Jikes Compiler
(C) Copyright IBM Corp. 1997, 2001.
- Licensed Materials - Program Property of IBM - All Rights
Reserved.

use: jikes [-bootclasspath path][-classpath path][-d dir][-
debug][-depend|-Xdepend][-deprecation][-encoding encoding][-
extdirs path][-g][-nowarn][-nowrite][-O][-sourcepath path][-
verbose][-
Xstdout][++][+B][+c][+OLDCSO][+D][+DR=filename][+E][+F][+Kname
=TypeKeyWord][+M][+P][+Td...d][+U][+Z] file.java...

-bootclasspath path prepend path to CLASSPATH
-classpath path     use path for CLASSPATH
-d dir              write class files in directory dir
-debug              no effect (recognized for
                    compatibility)
-depend | -Xdepend  recompile all used classes
-deprecation        report uses of deprecated features
-encoding encoding  use specified encoding to read source
                    files
-extdirs path       prepend all zip files in path to
                    CLASSPATH
-g                  debug (generate LocalVariableTable)
-nowarn             do not issue warning messages
-nowrite            do not write any class files
-O                  do not write LineNumberTable
-sourcepath path    also search for source files in path
-verbose            list files read and written
-Xstdout            redirect output listings to stdout
++                  compile in incremental mode
+B                  do not invoke bytecode generator
+c                  do not discard comments from lexer
                    output
+OLDCSO             perform original Jikes classpath order
                    for compatibility
+D                  report errors immediately in emacs-form
                    without buffering
+DR=filename        generate dependence report in filename
+E                  list errors in emacs-form
+F                  do full dependence check except for Zip
                    and Jar files
```

```
+Kname=TypeKeyWord  map name to type keyword
+M                  generate makefile dependencies
+P                  pedantic compilation - issues lots of
                    warnings
+Td...d             set value of tab d...d spaces; each d
                    is a decimal digit
+U                  do full dependence check including Zip
                    and Jar files
+Z                  treat cautions as errors

Version 1.15 - 26 Sept 2001
Originally written by Philippe Charles and David Shields
of IBM Research, Jikes is now maintained and refined by the
Jikes Project at:
http://ibm.com/developerworks/opensource/jikes
Please consult this URL for more information and to learn
how to report problems.
[root@conformix jikes-1.15]#
```

The following command will compile the `hello.java` program to create the `hello.class` output.

```
jikes hello.java -classpath /usr/java/j2sdk1.4.0/jre/lib/
rt.jar
```

Note that you must have core classes installed on you system to use the compiler. In the above example, the `classpath` is mentioned on the command line.

9.9 Miscellaneous

This section contains a brief summary of information and links to some other fields related to Java. With major industry players like Sun and IBM behind Java technologies, it is being pushed in many different areas.

9.9.1 Embedded Java

Sun has introduced Java Embedded Server, which enables developers to build embedded applications based upon the Java platform. In addition to other things, it includes a Java implementation of SSL. Information about Embedded Java is available at http://java.sun.com/products/embeddedjava/.

9.9.2 Real Time Java

Efforts are being made to implement a real-time Java virtual machine. Specifications for real-time Java can be found at http://www.rtj.org/. The specifications include information about how to manage memory, interrupts, scheduling, garbage collection and so on.

9.9.3 Wireless Applications

Java 2 Micro Edition or J2ME from Sun Microsystems introduces Wireless Java Technology. Information about using Java technology in wireless applications can be found at http://java.sun.com/products/embeddedjava/ .

9.10 References

1. Kaffe is available from its web site http://www.kaffe.org
2. Most of the Sun Java information is available from http://java.sun.com
3. Non-blocking library at http://www.cs.berkeley.edu/~mdw/proj/java-nbio/
4. The Jboss at its web site http://www.jboss.org/
5. The Jboss information at sourceforge.net http://sourceforge.net/projects/jboss/
6. Java Language Specifications at http://java.sun.com/docs/books/jls/
7. The Jikes compiler at http://oss.software.ibm.com/developerworks/opensource/jikes/
8. Information about gcj compiler at http://gcc.gnu.org/java/
9. The libgcj library from ftp://sourceware.cygnus.com/pub/java/
10. Real-time Java specifications at http://www.rtj.org/
11. Embedded Java at http://java.sun.com/products/embeddedjava/