

Similar Platforms

The success of Java and its write once, run anywhere concept has also led to other similar efforts. The most notable of these is the Microsoft .NET platform, which borrows many of the concepts and innovations of Java; in fact, it has an implementation of a Java-like language called Visual J# (formerly known as J++). (It is *Java-like* in that J# is not the Java language. Instead, J# contains non-standard extensions of the language.)

Later Microsoft stopped, withdrew its J# support, and created a new language called C#. C# is very similar to J# and Java, but not compatible with them. The differences between Java and C# can be read at [w:Comparison of C Sharp and Java](#).

Java Programming Environment

To compile Java programs, you will need to download and install the Java Development Kit (JDK). This is available from Sun's website ^[9]. Other hardware and operating system vendors also supply Java Development Kits for their platforms, although they may change the name of the kit. Sun produces JDKs for Windows, Linux, and Solaris.

There are numerous environments you can use to develop your Java programs. You can choose to write your programs in a text editor and then compile them using the command line, or you can use an integrated development environment (IDE). IDEs like NetBeans ^[10] and Eclipse ^[11] provide many useful functions such as syntax error checking, code completion, automatic compilation and debugging, which you may find useful, especially if this is your first foray into programming.

The Java Compiler

The JDK consists of a set of tools necessary to construct Java programs. The most notable tool in the JDK is the Java compiler, also known as `javac`.

`javac` compiles Java source files into executable Java class files. Source files are text files with a `.java` file name extension. You can create such files with a text editor, like Notepad, or an IDE. `javac` then compiles these files into loadable and executable class files, using the `.class` extension. For example, if you create a Java class `org.wikibooks.util.PrintDate.java`

```
package org.wikibooks.util;
import java.util.Date;
public class PrintDate
{
    public static void main(String[] args)
    {
        System.out.println(new Date());
    }
}
```

you can compile it by entering the following command in a command shell:

```
javac org/wikibooks/util/PrintDate.java
```

You would normally invoke this in a shell whose working directory is the root directory containing all of your source files resides. (The Java package statement, `package org.wikibooks.util`, corresponds to the directory structure `org/wikibooks/util` see Java packages.)

`javac` will create the file `PrintDate.class` in the same directory where the source file is located. If there are syntax errors, `javac` will print those to the shell. The `PrintDate.class` file contains the byte code, that will run under all hardware where the Java runtime is installed. Even if the `PrintDate.class` file was created in Windows operating

system, you can copy this file to unix and it will be executed fine.

Usually there is more than one class file created and those files are packaged to a `application_name.jar` file to distribute to run in any hardware.

IDEs may manage this process automatically. For example, Eclipse contains its own Java compiler and thus does not use `javac` directly. It automatically compiles the Java source files when you save them. The use of IDE's such as Eclipse are beyond the scope of this module, however, so consult the IDE's tutorials and help to see how they provide a Java programming environment.

The bytecode

It should be clear from the above paragraph that the Java compiler compiles source code text files with the extension `.java` into executable code usually confined into a class file with the `.class` extension. Such code is called Bytecode.

In many languages prior to Java, the source code would generally compile into the machine-code for the particular machine the program was compiled upon. Therefore, if a program was compiled on an X86 machine, it would run only on an X86 machine and no other. Java, on the other hand, produces a *bytecode* - an intermediate binary form of code that is a portable representation of the Java class. Any Java Virtual Machine on any hardware/operating system platform can then execute this same bytecode. There are some restrictions to this portability. For example, a Java ME system cannot execute all programs compiled for the Java SE environment because Java ME is pared down to small devices. But in general, a Java SE program can run unmodified on any Java SE virtual machine.

The JIT compiler

Being compiled halfway through, it is the job of the Java Virtual Machine to compile the rest of the program to native code at the time of its execution making Java code follow the "Write Once, Run Anywhere" (WORA) policy. The compiler used to compile bytecode into machine-code at runtime is called the *Just-In-Time* or *JIT compiler*. Once a piece of code is compiled by the JVM to execution code, the code is used and re-used again and again, to speed up execution.

The Java Runtime Environment

The Java Runtime Environment, or JRE, is responsible for the execution of Java programs. The Sun JDK also includes the JRE. The JRE however can also be installed and used without installing the JDK which is useful if you wish to execute Java programs but not build them. The JRE helps load Java programs into the memory and executes them.

Main entry point

In Java programming, classes are used to define objects and entities that hold particular data. To execute a Java program, a special class is required to assist in loading the program into the computer's memory. Such a class contains a method with the following signature.

```
public static void main(String[] args) {...}
```

The class is hence said to have a *main entry point* defined and is usually called a *Java program*. The method described above as the main entry point is usually nicknamed the *main method*. Java classes without main methods are simply classes, although they may be part of a program.

Each Java class can have a *main* method, as opposed to C, and C++ where there can only be one, for the whole program. This feature is good in a sense that the *main* method can be implemented to do some testing on the class. The program must define one class's main method as the entry point for the program.

Executing a command-line Java program

If a class which you compile with `javac` has a main entry point, you can execute the class by specifying the class name as an argument to the `java` program.

```
java org.wikibooks.util.PrintDate
```

This will run the main method in the `PrintDate` class in the `org.wikibooks.util` package. (Packages provide a convenient way to provide namespaces and organization of Java classes. We'll use `org.wikibooks.util` as a parent package in this module. More on packages.)

Your program will begin execution in the shell window. Inputs and outputs will be gathered from and to your shell window. On a Windows platform, you can use a DOS command window as the command shell program for the execution of Java programs. The JRE is normally called `java` because the `java` program is the most widely used program to execute Java programs.

Executing a Graphical User Interface Java program

On the Windows platform, there is an alternate JRE executable called `javaw.exe` or `javaw` which runs Java programs as a Windows native application - that is, with no console for standard input or output.

```
javaw org.wikibooks.util.ViewDate
```

Rather than executing in the console, the Java program would be executed in a separate Windows native process. This is typically done for Java applications which create their own graphical user interface (GUI) windows. The above-mentioned `org.wikibooks.util.PrintDate` program, which prints output to the standard output stream, is not appropriate for use with `javaw` as there is no console output. Instead, `org.wikibooks.util.ViewDate` would be a program which creates its own windows to display the date.

As with `java`, IDE's also manage the execution of Java programs in slightly different ways. They may provide shortcuts for running programs and windows for capturing the output.

On UNIX/Linux this does not matter. If the program is launched graphically (by file association in a file manager) a console is not shown. GUI programs will have the titlebars following the look and feel of your desktop (KDE, GNOME, Fluxbox, XFCE) theme.

Other JDK tools

Apart from the tools specified above in detail, the JDK has matured over the years and has included in itself several other tools. Where some of these tools are no longer used, others offer a far greater deal of capability to the Java Development Kit. Below is a list of some of the tools available for the JDK.

The *apt* tool

In Java 1.5 (alias Java 5.0) Sun added a mechanism called *annotations*. Annotations allow the addition of meta-data to Java source code, and even provide mechanisms to carry that meta-data forth into a compiled class files.

Also starting with Java 1.5 Sun added the *apt* tool to the JDK. *apt* works on Java source code. It is an *annotation processing tool* which digs through source code, finds annotation statements in the source code and executes actions if it finds known annotations. The most common task is to generate some particular source code.

The actions *apt* performs when finding annotations in the source code are not hard-coded into *apt*. Instead, one has to code particular annotation handlers (in Java). These handlers are called *annotation processors*.

The most difficult thing with *apt* is that Sun decided to use a whole set of new terminology. *apt* can simply be seen as a source code preprocessor framework, and *annotation processors* are typically just code generators.