

External Links

- Sun's website ^[15]
- Java.Net JDK repository ^[16]

Compilation

Navigate **Getting Started** topic:

- Installing Java on Your Computer
- Compiling programs
- Running Java programs
- Developing the first Java program
- Understanding a Java program

We have already discussed compilation basics. Here's a recap of the concepts we'd seen earlier and some additional details.

Compiling to bytecode

In Java, programs are not compiled into executable files; they are compiled into Bytecode (as discussed earlier), which the JVM then executes at runtime. Java source code is compiled into bytecode when we use the `javac` compiler. The bytecode gets saved on the disk with the file extension `.class`. When the program is to be run, the bytecode is converted, using the Just-In-Time(JIT) compiler. The result is machine code which is then fed to the memory and is executed.

So Java has two step compilation:

- Step one to create byte-code
- Step two to create machine level code

The Java classes/Byte Codes are compiled to machine code and loaded into memory by the JVM when needed the first time. This is different than other languages like C/C++ where the whole program had to be compiled to machine code and linked to create an executable file, before the program could start.

JIT compilers compile byte-code once and the compiled machine code are re-used again and again, to speed up execution. Early Java compilers compiled the byte-code to machine code each time it was used, but more modern compilers cache this machine code for reuse on the machine. Even then, java's JIT compiling was still faster than an "interpreter-language", where code is compiled from **high level language**, instead of from byte-code each time it was used.

Automatic Compilation of Dependent Classes

In Java, if you have used any reference to any other java object, then the class for that object will be automatically compiled, if that was not compiled already. These automatic compilations are nested, and this continues until all classes are compiled that are needed to run the program. It is usually enough to compile only the high level class, since all the dependent classes will be automatically compiled.

```
javac ... MainClass.java
```

However, you can't rely on this feature if your program is using reflection to create objects, or you are compiling for servlets or a "jar" package. In these cases you should list these classes for explicit compilation.

```
javac ... MainClass.java, ServletOne.java, ...
```

The best way is to use a build tool to build your application. The build tool would check all the needed dependencies and compile only the needed class for the build. The Ant tool is the best and the most popular build tool currently available. Using Ant you would build your application from the command line by typing:

```
ant build.xml
```

The xml file contains all the information needed to build the application.

Note: In rare cases, your code may appear to compile correctly but the program behaves as if you were using an old copy of the source code (or otherwise reports errors during runtime.) When this occurs, you may need to clean your compilation folder by either deleting the class files or using the Clean command from the IDE.

The next most popular way to build applications are using an IDE. IDE stands for *Integrated Development Environment*, examples of which are listed below.

Packages, Subdirectories, and Resources

Each Java top level class belongs to a package (covered in the chapter about Packages). This may be declared in a `package` statement at the beginning of the file; if that is missing, the class belongs to the unnamed package.

For compilation, the file must be in the right directory structure. A file containing a class in the unnamed package must be in the current/root directory; if the class belongs to a package, it must be in a directory with the same name as the package.

The convention is that package names and directory names corresponding to the package consist of only lower case letters.



Example:

Top level package. A class with this package declaration `package example;` has to be in a directory named `example`



Example:

Subpackages. A class with this package declaration `package org.wikibooks.en;` has to be in the `org/wikibooks/en` directory.

Java programs often contain non-code files such as images and properties files. These are referred to generally as 'resources' and stored in directories local to the classes in which they're used. For example, if the class `com.example.ExampleApp` uses the `icon.png` file, this file could be stored as `/com/example/resources/icon.png`. These resources present a problem when a program is compiled, because `javac` does not copy them to wherever the `.class` files are being compiled to (see above); it is up to the programmer to move the resource files and directories. *See also the section on how to automate this using ant, below.*

Filename Case

The Java source file name must be the same as the public class name, the file contains. There can be only one public class defined per file. The Java class name is case sensitive, as is the source file name.

The naming convention for the class name is for it to start with a capital letter.

Compiler Options

Additional Tools

IDEs

This section contains a little about the different IDEs available and their strengths and weaknesses.

JBuilder

JBuilder is a IDE with proprietary source code, sold by Borland. One of the advantages in integration with together, a modeling tool.

JCreator

There's info at: <http://www.apcomputerscience.com/ide/jcreator/index.htm>

Eclipse

Eclipse is a free IDE, plus a developer tool framework that can be extended for a particular development need. IBM was behind this free software development and it replaced IBM Visual Age tool. The idea was to create a standard look and feel that can be extended. The extendibility has distinguished Eclipse from other IDE tools. Eclipse also meant to compete with Microsoft Visual Studio tools. Microsoft tools give a standard way of developing code in the Microsoft world. Eclipse gives a similar standard way of developing code in the Java world, with big success so far. With the online error checking only, coding can be speed up by at least 50%(coding does not include programming).

The goal for Eclipse are twofold:

- Give a standard IDE for developing code
- Give a starting point, and the same look and feel for all other more sophisticated tools built on Eclipse

IBM's WSAD, and later IBM Rational Software Development Platform are built on Eclipse.

Standard Eclipse features:

- Standard window management (perspectives, views, browsers, explorers, ...)
- As you type error checking (immediate error indications, ...)
- As you type help window (type ., or <ctrl> space, ...)
- Automatic build (changed source code automatically compiled, ...)
- Built in debugger (full featured GUI debugger)
- Source code generation (getters and setters, ...)
- Searches (for implementation, for references, ...)
- Code refactoring (global reference update, ...)
- Plug-in-based architecture (be able to build tools that integrate seamlessly with the environment and other tools)
- ...

For more information see;

- [Eclipse ^[17]].