**"javac"**:

Commonly used options are:

* "-help" - Displays a short help text.
* "-verbose" - Generates verbose output to standard output.
* "-classpath classpath" - Specifies a list of path names where the compiler will search for compiled type definitions. If "-classpath" is not specified, the current directory will be used as the class path.
* "-sourcepath sourcepath" - Specifies a list of path names where the compiler will search for source type definitions. If "-source" is not specified, the current directory will be used as the source path.
* "-d directory" - Specifies the directory where the compiler will store the generated class files. If "-d" is not specified, the class files will be stored in the same places as the source files.
* "-g | -g:none" - Asks the compiler to generate debugging information into the class files or generates no debugging information at all.
* The most commonly used "javac" option is "-classpath", which specifies a list of path names where the compiler will search for compiled type definitions. If "-classpath" is not specified, the current directory will be used as the class path.
* When compiler encounters an unknown type in the source file, it will try to find the type definition by searching class files in the class path.

**Option '-sourcepath' - Specifying Source Path**

If you use a new type, and you don't have the class definition of that type, but you have its source definition, you can use the "-sourcepath sourcepath" option to tell compiler to get that source definition.

**Option '-d' - Specifying Output Directory**

By default, "javac" will output the class file in the same directory as the source file. But you can change this default behavior by using the "-d" option. It will make "javac" to output the class files into the specified directory.

To test this option, I wrote the following Java source file: PackagedHello.java

/\*\*

\* PackagedHello.java

\* Copyright (c) 2006 by Dr. Herong Yang, http://www.herongyang.com/

\*/

package com.herong.util;

public class PackagedHello {

public static void main(String[] a) {

System.out.println("Packaged: Hello world!");

}

}

Test 1 - Storing PackagedHello.java in the wrong directory:

C:\herong>dir PackagedHell.\*

264 PackagedHello.java

C:\herong>javac PackagedHello.java

C:\herong>dir PackagedHell.\*

459 PackagedHello.class

264 PackagedHello.java

C:\herong>java -cp . PackagedHello

Exception in thread "main" java.lang.NoClassDefFoundError:

PackagedHello (wrong name: com/herong/util/PackagedHello)

This proves that packaged classes can not be stored in any directory.

Test 2 - Storing PackagedHello.java in the right directory:

C:\herong>mkdir .\com

C:\herong>mkdir .\com\herong

C:\herong>mkdir .\com\herong\util

C:\herong>copy PackagedHello.java .\com\herong\util

C:\herong>del PackagedHell.\*

C:\herong>javac .\com\herong\util\PackagedHello.java

C:\herong>dir .\com\herong\util

459 PackagedHello.class

264 PackagedHello.java

C:\herong>java -cp . com.herong.util.PackagedHello

Packaged: Hello world!

As you can see, the compiler outputs the class file in the same directory as the source file by default.

Test 3 - Outputing PackagedHello.class to a new directory:

C:\herong>mkdir .\cls

C:\herong>javac -d .\cls .\com\herong\util\PackagedHello.java

C:\herong>dir .\cls\com\herong\util

459 PackagedHello.class

C:\herong>java -cp .\cls com.herong.util.PackagedHello

Packaged: Hello world!

This time, the compiler outputs the class file under a directory path: .\cls.