Supplement: Java Packages
For Introduction to Java Programming
By Y. Daniel Liang

Packages are used to group classes. So far, all the classes in this book are grouped into a default package. You can explicitly specify a package for each class. There are four reasons for using packages.

- **To locate classes.** Classes with similar functions can be placed in the same package to make them easy to locate.

- **To avoid naming conflicts.** When you develop reusable classes to be shared by other programmers, naming conflicts often occur, i.e., two classes with the same name. To prevent this, put your classes into packages so that they can be referenced through package names.

- **To distribute software conveniently.** Packages group related classes so that they can be easily distributed.

- **To protect classes.** Packages provide protection so that the protected members of the classes are accessible to the classes in the same package, but not to the external classes.

1 Package-Naming Conventions

Packages are hierarchical, and you can have packages within packages. For example, **java.lang.Math** indicates that **Math** is a class in the package **lang** and that **lang** is a package in the package **java**. Levels of nesting can be used to ensure the uniqueness of package names.

Choosing a unique name is important because your package may be used on the Internet by other programs. Java designers recommend that you use your Internet domain name in reverse order as a package prefix. Since Internet domain names are unique, this prevents naming conflicts. Suppose you want to create a package named **mypackage** on a host machine with the Internet domain name **prenhall.com**. To follow the naming convention, you would name the entire package **com.prenhall.mypackage**. By convention, package names are all in lowercase.

2 Package Directories

Java expects one-to-one mapping of the package name and the file system directory structure. For the package named **com.prenhall.mypackage**, you must create a directory, as shown in Figure 1(a). In other words, a package is actually a directory that contains the bytecode of the classes.

(a)   (b)

Figure 1

The package com.prenhall.mypackage is mapped to a directory structure in the file system.

The **com** directory does not have to be the root directory. In order for Java to know where your package is in the file system, you must modify the environment variable **classpath** so that it points to the directory in which your package resides. Such a directory is known as the *classpath* for the class. Suppose the **com** directory is under c:\book, as shown in Figure 5.1(b). The following line adds c:\book into the **classpath**:

    set classpath=.;c:\book;

The period (.) indicating the current directory is always in **classpath**. The directory **c:\book** is in **classpath** so that you can use the package **com.prenhall.mypackage** in the program.

You can add as many directories as necessary in **classpath**. The order in which the directories are specified is the order in which the classes are searched. If you have two classes of the same name in different directories, Java uses the first one it finds.

<Side Remark: classpath>
The **classpath** variable is set differently in Windows and UNIX, as outlined below.

> **Windows 98**: Edit autoexec.bat using a text editor, such as Microsoft Notepad.

Windows NT/2000/XP: Go to the Start button and choose Control Panel, select the System icon, then modify **classpath** in the Environment Variables.

UNIX: Use the **setenv** command to set **classpath**, such as

```
setenv classpath .:/home/book
```

If you insert this line into the .cshrc file, the **classpath** variable will be set automatically when you log on.

NOTE

On Windows 95 and Window 98, you must restart the system in order for the **classpath** variable to take effect. On Windows NT/2000/ME/XP, however, the settings are effective immediately. They affect any new command windows, but not the existing command windows.

3 Putting Classes into Packages

Every class in Java belongs to a package. The class is added to a package when it is compiled. All the classes that you have used so far in this book were placed in the current directory (a default package) when the Java source programs were compiled. To put a class in a specific package, you need to add the following line as the first noncomment and nonblank statement in the program:

package packagename;

Let us create a class named **Format** and place it in the package **com.prenhall.mypackage**. The **Format** class contains the **format(number, numberOfDecimalDigits)** method, which returns a new number with the specified number of digits after the decimal point. For example, **format(10.3422345, 2)** returns 10.34, and format(-0.343434, 3) returns -0.343.

1. Create Format.java in Listing 5.10 and save it into c:\book\com\prenhall\mypackage.

Listing 1 Format.java

```
package com.prenhall.mypackage;

public class Format {
```

```
    public static double format(
        double number, int numberOfDecimalDigits) {
      return Math.round(number * Math.pow(10,
  numberOfDecimalDigits)) /
        Math.pow(10, numberOfDecimalDigits);
    }
  }
```

        2. Compile Format.java and place it in
        c:\book\com\prenhall\mypackage.

A class must be defined as public in order to be accessed
by other programs. If you want to put several classes into
the package, you have to create separate source files for
them because each file can have only one public class.

Format.java can be placed under
anyDir\com\prenhall\mypackage and Format.class in
anyOtherDir\com\prenhall\mypackage, and ***anyDir*** and
anyOtherDir may be the same or different. To make the class
available, add ***anyOtherDir*** in the classpath, using the
command:

        set classpath=%classpath%;anyOtherDir

     NOTE

     Class files can be archived into a single file
     for convenience. For instance, you may compress
     all the class files in the folder mypackage
     into a single zip file named mypackage.zip with
     subfolder information kept as shown in Figure
     2. To make the classes in the zip file
     available for use, add the zip file to the
     classpath like this:
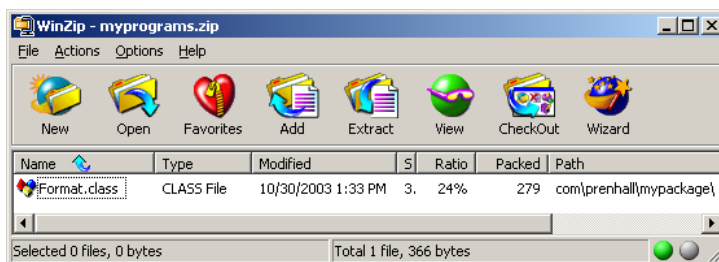
     classpath=%classpath%;c:\mypackage.zip

Figure 2

Class files can be archived into a single compressed file.

NOTE

An IDE such as JBuilder uses the *source directory path* to specify where the source files are stored and uses the *class directory path* to specify where the compiled class files are stored.

A source file must be stored in a package directory under the source directory path. For example, if the source directory is **c:\mysource**, and the package statement in the source code is **package com.prenhall.mypackage**, then the source code file must be stored in c:\mysource\com\prenhall\mypackage.

A class file must be stored in a package directory under the class directory path. For example, if the class directory is **c:\myclass**, and the package statement in the source code is **package com.prenhall.mypackage**, then the class file must be stored in c:\myclass\com\prenhall\mypackage.

4 Using Classes from Packages

There are two ways to use classes from a package. One way is to use the fully qualified name of the class. For example, the fully qualified name for **JOptionPane** is **javax.swing.JOptionPane**. For **Format** in the preceding example, it is **com.prenhall.mypackage.Format**. This is convenient if the class is used only a few times in the program. The other way is to use the **import** statement. For example, to import all the classes in the **javax.swing** package, you can use

> **import javax.swing.*;**

An import that uses an ***** is called *an import on demand* declaration. You can also import a specific class. For example, this statement imports **javax.swing.JOptionPane**:

> **import javax.swing.JOptionPane;**

The information for the classes in an imported package is not read in at compile time nor runtime unless the class is used in the program. The import statement simply tells the

compiler where to locate the classes. There is no performance difference between an import on demand declaration and a specific class import declaration.

Let us write a program that uses the **Format** class in the **com.prenhall.mypackage.mypackage** package.

1. Create TestFormatClass.java in Listing 5.11 and save it into c:\book.

Listing 2 TestFormatClass.java

```java
import com.prenhall.mypackage.Format;

public class TestFormatClass {
  /** Main method */
  public static void main(String[] args) {
    System.out.println(Format.format(10.3422345, 2));
    System.out.println(Format.format(-0.343434, 3));
  }
}
```
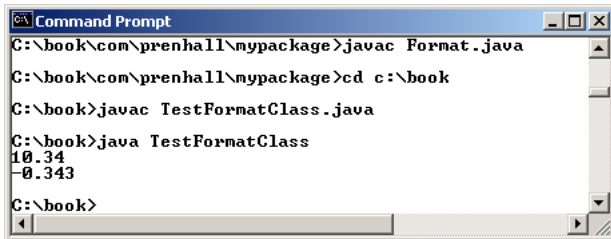
2. Run TestFormatClass, as shown in Figure 3.

Figure 3

TestFormatClass uses *Format* defined in *com.prenhall.mypackage*.

TestFormatClass.java can be placed anywhere as long as c:\book is in the classpath so that the **Format** class can be found. Please note that **Format** is defined as public so that it can be used by classes in other packages.

The program uses an **import** statement to get the class **Format**. You cannot import entire packages, such as **com.prenhall.*.***. Only one asterisk (*) can be used in an **import** statement.

NOTE: The **format** method can be invoked from any class. If you create a new class in the same

package with **Format**, you can invoke the **format** method using **ClassName.methodName** (e.g., **Format.format**). If you create a new class in a different package, you can invoke the **format** method using **packagename.ClassName.methodName** (e.g., **com.prenhall.mypackage.Format.format**).