APPENDIX H

Packages

NOTE: To use this appendix you must understand how your operating system uses directories, or folders. In addition, you must know how to set the value of an environment variable. The process of setting an environment variable differs among operating systems.

So far, you have been storing the classes you have created in the same folder or directory as the program that uses them. That is where the compiler looks for classes by default. In real-world application development, however, this approach is less than ideal. If you are developing more than one application that uses the same set of classes, you would have to make separate copies of the classes and store them in the same disk location as each application. A better approach is to have the classes stored in a central location available to all applications. Only one copy of the classes is needed, regardless of the number of applications you develop using them. In Java, this can be accomplished by using packages.

A *package*, which is also called a *library*, is a named group of related classes. Packages are stored in their own folder or directory on the computer's disk. The compiler is informed of the package's location, so it can find it regardless of where the application that uses the package may be stored.

Let's look at a simple example. Suppose we create two classes: `Car` and `Truck`. Both classes are part of a package named `vehicles`. Code Listing H-1 shows the listing for the `Car` class, and Code Listing H-2 shows the listing for the `Truck` class.

Code Listing H-1 (Car.java)

```
 1  package vehicles;
 2
 3  /**
 4   *  This class is in the vehicles package.
 5   */
 6
 7  public class Car
 8  {
 9     private int passengers;   // Number of passengers
10     private double topSpeed;  // Top speed in miles per hour
11
12     /**
13      *  Constructor
14      */
15
16     public Car(int passengers, int topSpeed)
17     {
18        this.passengers = passengers;
19        this.topSpeed = topSpeed;
20     }
21
22     /**
23      *  toString method
24      */
25
26     public String toString()
27     {
28        return "Passengers: " + passengers
29             + "\nTop Speed: " + topSpeed + " miles per hour";
30     }
31  }
```

Code Listing H-2 (Truck.java)

```
 1  package vehicles;
 2
 3  /**
 4   *  This class is in the vehicles package.
 5   */
 6
 7  public class Truck
 8  {
 9     private double mpg;  // Fuel economy in miles-per-gallon
10     private double tons; // Hauling capacity in tons
11
12     /**
13      * Constructor
14      */
15
16     public Truck(double mpg, double tons)
17     {
18        this.mpg = mpg;
19        this.tons = tons;
20     }
21
22     /**
23      * toString method
24      */
25
26     public String toString()
27     {
28        return "Fuel economy: " + mpg + " miles per gallon"
29             + "\nHauling capacity: " + tons + " tons";
30     }
31  }
```

Notice that the first line of each file reads:

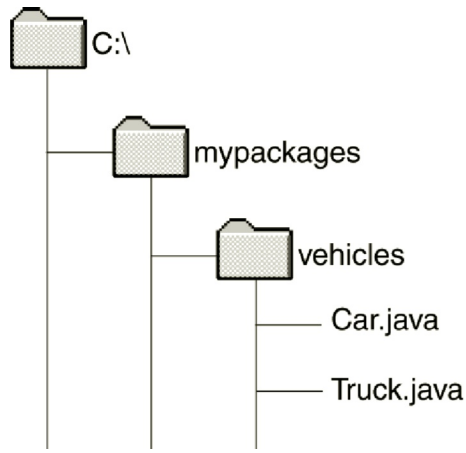
```
package vehicles;
```

The word `package` is a key word in Java, and `vehicles` is the name of the package to which the class belongs. Notice the statement is terminated by a semicolon. This statement informs the compiler that the contents of the file belong to the `vehicles` package.

NOTE: The compiler expects the package statement to be the first statement in a file. Only comments and blank lines may be placed before it.

When a class is part of a package, the compiler expects the class file to be stored in a directory with the same name as the package. For example, the `Car` and `Truck` classes are part of the `vehicles` package, so they must be stored in a directory named *vehicles*. Typically, the *vehicles* directory would be stored under another directory that acts as the base directory for packages. Figure H-1 depicts an example directory structure on a Windows system. The *mypackages* directory is the base directory for packages. Under *mypackages*, the *vehicles* directory appears, which holds the *Car.java* and *Truck.java* files.

Figure H-1 Example directory structure for storing a package



Once a package has been created, the Java compiler must be informed of its location. The exact process that you use depends on your Java compiler and your operating system. You should consult your instructor for precise details, but we will give a general overview of what needs to be done when using the Sun JDK under Windows and UNIX or Linux.

The Sun JDK uses an operating system environment variable named `CLASSPATH` to determine where the base directory for your packages is located. The `CLASSPATH` environment variable holds a string that consists of one or more pathnames, separated by a delimiting character. To inform the compiler of your base directory's location, simply add its pathname to the contents of the `CLASSPATH` environment variable.

Setting CLASSPATH under Windows

The following is an example command that can be used at the command line in Windows to set the CLASSPATH environment variable.

```
set CLASSPATH=C:\mypackages;
```

There are no spaces before or after the = sign. In Windows, multiple pathnames stored in the variable are separated by a semicolon. This command stores two pathnames in the CLASSPATH environment variable: C:\mypackages and the . symbol, which represents the current directory. This tells Java to search two locations for packages: C:\mypackages and the current directory. Here is another example:

```
set CLASSPATH=C:\mypackages;C:\project;
```

This command stores three pathnames in the CLASSPATH environment variable: C:\myPackages, C:\project, and the . symbol, which represents the current directory. This tells Java to search these three locations when looking for packages.

Setting CLASSPATH under UNIX or Linux

NOTE: The UNIX and Linux instructions in this appendix assume you are using the bash shell.

The following is an example command that can be used in UNIX or Linux to set the CLASSPATH environment variable.

```
export CLASSPATH=/home/tsmith/mypackages:
```

In UNIX and Linux, multiple pathnames stored in the variable are separated by a colon. This command stores two pathnames in the CLASSPATH environment variable: /home/tsmith/mypackages and the . symbol, which represents the current directory. This tells Java to search two locations for packages: /home/tsmith/mypackages and the current directory. Here is another example:

```
export CLASSPATH=/home/tsmith/mypackages:/home/tsmith/project;
```

This command stores three pathnames in the CLASSPATH environment variable: /home/tsmith/mypackages, /home/tsmith/project, and the . symbol, which represents the current directory. This tells Java to search these three locations when looking for packages.

NOTE: You should always include the dot symbol, which represents the current directory, as a pathname in the CLASSPATH variable. If you do not, your programs will not run correctly.

Using the `import` Statement

When a package has been created and stored on the disk as previously described and the `CLASSPATH` variable has been set with the pathname of the base directory for your packages, you are ready to create programs that use your package. Code Listing H-3 uses both the `Car` and `Truck` classes.

Code Listing H-3 (CarTruckDemo.java)

```
 1  import vehicles.*;
 2
 3  /**
 4   *  This program demonstrates the Car and Truck
 5   *  classes which are part of the vehicles package.
 6   */
 7
 8  public class CarTruckDemo
 9  {
10     public static void main(String[] args)
11     {
12        Car roadster = new Car(2, 155);
13        Truck pickUp = new Truck(18, 2);
14
15        System.out.println("Here's information " +
16                           "about the car:");
17        System.out.println(roadster);
18        System.out.println();
19        System.out.println"Here's information " +
20                          "about the truck:");
21        System.out.println(pickUp);
22     }
23  }
```

Program Output

```
Here's information about the car:
Passengers: 2
Top speed: 155.0 miles per hour

Here's information about the truck:
Fuel economy: 18.0 miles per gallon
Hauling capacity: 2.0 tons
```

Notice that line 1 contains an `import` statement. The word `import` is a key word in Java. The name that follows `import` is the name of a package the program intends to use. The `.*` that follows the package name means to import all the classes that are part of that pack-

age. This statement tells the compiler to make all the classes that are part of the `vehicles` package available to the program. If we only wanted to make the `Truck` class available, we could have used the following `import` statement:

```
import vehicles.Truck;
```

In this case the compiler would only make the `Truck` class available. Any references to the `Car` class would cause an error. Likewise, we could use the following statement to make only the `Car` class available:

```
import vehicles.Car;
```

You will recall that you have previously used the following `import` statement in programs that use the `DecimalFormat` class:

```
import java.text.DecimalFormat;
```

This statement tells the compiler to use the `DecimalFormat` class, which is part of the `java.text` package. The `java.text` package is part of the Java standard class library. You have also used the following `import` statement in programs using the `Scanner` class:

```
import java.util.Scanner;
```

This statement tells the compiler to use the `Scanner` class, which is part of the `java.util` package. The `java.util` package is also part of the Java standard class library. At this point, it might be helpful to summarize the steps necessary to create and use a package:

1. Place an appropriate `package` statement in each class file that is to be part of the package. The `package` statement must be the first line of the file or preceded only by comments and/or blank lines.
2. You should have a base directory on your system for storing your packages. Under this directory, create a directory that bears the same name as the package. Store the package's class files in this folder.
3. Add the pathname of the package base directory to the `CLASSPATH` environment variable.
4. Place an appropriate `import` statement in each program that intends to use the package.

Using Fully Qualified Class Names

It is not required that you use the `import` statement to access classes in a package. Without the `import` statement, however, you must use the classes' fully qualified names. The fully qualified names of the `Car` and `Truck` classes in the `vehicle` package are `vehicles.Car` and `vehicles.Truck`. Code Listing H-4 is a version of Code Listing H-3, which uses fully qualified class names instead of the `import` statement.

Code Listing H-4 **(CarTruckDemo2.java)**

```
1  /**
2   *  This program demonstrates the Car and Truck
3   *  classes by using their fully-qualified names
```

```
 4    *  instead of an import statement.
 5    */
 6
 7  public class CarTruckDemo2
 8  {
 9     public static void main(String[] args)
10     {
11        vehicles.Car roadster = new vehicles.Car(2, 155);
12        vehicles.Truck pickUp = new vehicles.Truck(18, 2);
13
14        System.out.println("Here's information " +
15                           "about the car:");
16        System.out.println(roadster);
17        System.out.println();
18        System.out.println("Here's information " +
19                           "about the truck:");
20        System.out.println(pickUp);
21     }
22  }
```

Program Output

Same as Code Listing H-3.

Notice that every occurrence of the `Car` and `Truck` class names must be written as `vehicles.Car` and `vehicles.Truck`. Obviously it is more convenient to use the `import` statement.

Using Package Subdirectories

You have learned that the Java compiler locates packages by searching the directories listed by the `CLASSPATH` environment variable. It is possible to create subdirectories in these locations and store packages there as well. For example, suppose we create a set of classes for calculating the pay of employees. The first step would be to create a folder to hold the packages. Figure H-2 shows the `employees` directory has been added to the `mypackages` directory.

Under the `employees` directory are two other directories: `hourly` and `salaried`. Figure H-3 shows the contents of the `hourly` directory.

The files *Clerical.java* and *AssemblyLine.java* contain code for classes that calculate the hourly pay for clerical staff and assembly line workers. If we looked at the *Clerical.java* file, we would see something similar to the following:

```
package employees.hourly;

/**
 *  This class is part of the employees.hourly package.
 */

public class Clerical
{
      Details of this class are omitted.
}
```

Figure H-2 `employees` directory added to the `mypackages` directory

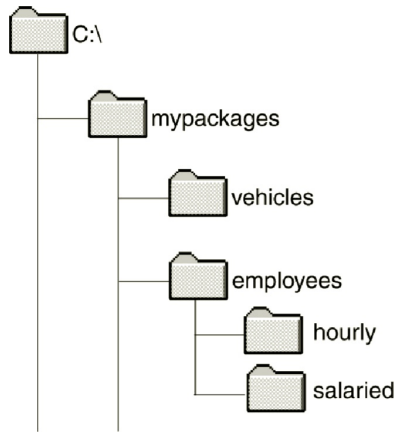
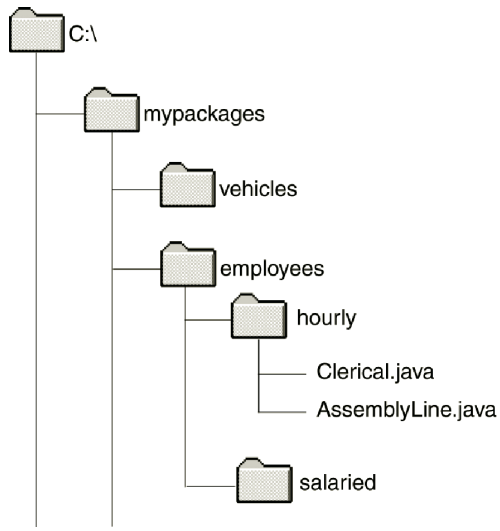


Figure H-3 Directory structure showing the contents of the `hourly` directory



If we looked in the *AssemblyLine.java* file, we would see something similar to the following:

```
package employees.hourly;
/**
 *  This class is part of the employees.hourly package.
 */

public class AssemblyLine
{
     Details of this class are omitted.
}
```

Notice the first line of both files is the following `package` statement:

```
package employees.hourly;
```

The files are stored in the `hourly` directory. Because `hourly` is a subdirectory of `employees`, the two names are separated by a period. This makes the package name `employees.hourly`. The following `import` statement could then be added to any program that needs to use the `Clerical` or `AssemblyLine` classes:

```
import employees.hourly.*;
```

This statement makes all of the classes that are part of the `employees.hourly` package accessible. As before, individual classes can be specifically named in the `import` statement. For example, the following statement will make only the `Clerical` class accessible:

```
import employees.hourly.Clerical;
```

Access Specifiers and Packages

So far you have learned that class members may be declared as either `public` or `private`. Members that are declared as `public` may be accessed by statements outside the class as well as inside the class. Members that are declared as `private`, however, may only be accessed by statements inside the class.

In addition to `public` and `private`, Java also allows class members to be declared with no access specifier at all. When a class member is declared with no access specifier, it is accessible by any statement within the same package. For example, consider the following scenario. Class `A` and class `B` are in the same package. The variable `x` is a member of class `A` and is declared with no access specifier. Because `x` is declared with no access specifier, the methods in class `B` have access to it (as if it were `public`). Statements that are outside the package, however, cannot access `x`. Table H-1 summarizes the effect of access specifiers, as they pertain to what you have learned so far.

Table H-1 The effect of access specifiers

Access Attribute	Description
No attribute	May be applied to classes and class members (variables or methods). This makes a class or class member accessible within the package.
`public`	May be applied to classes and class members (variables or methods). This makes a class or class member accessible to all statements in the program (inside or outside the package).
`private`	May only be applied to member variables and member methods. A private variable or method is only accessible by statements in the same class.

The Standard Java Packages

The standard Java classes that make up the API are organized into packages. Table H-2 lists a few of them.

Table H-2 A few of the standard Java packages

Package	Description
`java.applet`	Provides the classes necessary to create an applet.
`java.awt`	Provides classes for the Abstract Windowing Toolkit. These classes are used in drawing images and creating graphical user interfaces.
`java.io`	Provides classes that perform various types of input and output.
`java.lang`	Provides general classes for the Java language. This package is automatically imported.
`java.net`	Provides classes for network communications.
`java.security`	Provides classes that implement security features.
`java.sql`	Provides classes for accessing databases using structured query language.
`java.text`	Provides various classes for formatting text.
`java.util`	Provides various utility classes.
`javax.swing`	Provides classes for creating graphical user interfaces.

To use a class from a Java package, you must have an appropriate `import` statement in your program. For example, you have used the `DecimalFormat` class, which is in the `java.text` package. This class requires the following `import` statement:

```
import java.text.DecimalFormat;
```

You have also used various classes from the `java.io` package to perform file input and

output. To import all of the classes from the `java.io` package, you use the following `import` statement:

```
import java.io.*;
```

The `java.lang` package is the only package that does not require an `import` statement. This package contains general classes, such as `String` and `System`, that are fundamental to the Java programming language. The `java.lang` package is automatically imported by all Java programs.