Tutorial Link https://course.testpad.chitkara.edu.in/tutorials/Java : Introduction to Package/5b1e49377becc0459deadd67

TUTORIAL

Java : Introduction to Package

Topics

Packages are containers for classes that are used to keep the class name space compartmentalized. For example, a package allows you to create a class named List, which you can store in your own package without concern that it will collide with some other class named List stored elsewhere. Packages are stored in a hierarchical manner and are explicitly imported into new class definitions.

Packages

Till now, the name of each example class was taken from the same name space. This means that a unique name had to be used for each class to avoid name collisions. After a while, without some way to manage the name space, you could run out of convenient, descriptive names for individual classes. You also need some way to be assured that the name you choose for a class will be reasonably unique and not collide with class names chosen by other programmers. Java provides a mechanism for partitioning the class name space into more manageable chunks. This mechanism is the package. The package is both a naming and a visibility control mechanism. You can define classes inside a package that are not accessible by code outside that package. You can also define class members that are only exposed to other members of the same package. This allows your classes to have intimate knowledge of each other, but not expose that knowledge to the rest of the world.

Defining a Package

To create a package is quite easy: simply include a package command as the first statement in a Java source file. Any classes declared within that file will belong to the specified package. The package statement defines a name space in which classes are stored. If you omit the package statement, the class names are put into the default package, which has no name. While the default package is fine for short, sample programs, it is inadequate for real applications. Most of the time, you will define a package for your code.

This is the general form of the package statement:

```
package pkg;
```

Here, pkg is the name of the package. For example, the following statement creates a package called MyPackage.

```
package MyPackage;
```

Java uses file system directories to store packages. For example, the .class files for any classes you declare to be part of MyPackage must be stored in a directory called MyPackage. Remember that case is significant, and the directory name must match the package name exactly.

More than one file can include the same package statement. The package statement simply specifies to which package the classes defined in a file belong. It does not exclude other classes in other files from being part of that same package. Most real-world packages are spread across many files. You can create a hierarchy of packages. To do so, simply separate each package name from the one above it by use of a period. The general form of a multileveled package statement is shown here:

```
package pkg1[.pkg2[.pkg3]];
```

A package hierarchy must be reflected in the file system of your Java development system. For example, a package declared as

```
package java.awt.image;
```

needs to be stored in java/awt/image, java\awt\image, or java:awt:image on your UNIX, Windows, or Macintosh file system, respectively. Be sure to choose your package names carefully. You cannot rename a package without renaming the directory in which the classes are stored. A short example to show package building is following:

Java

```
// A simple package
package MyPack;
class Account
{
  String name;
  double bal;
  Account(String n, double b)
  {
    name = n;
    bal = b;
  }
  void show()
  {
    if(bal<0)
      System.out.print("--> ");
    System.out.println(name + ": $" + bal);
  }
}
```

Call this file Account.java, and put it in a directory called MyPack. Next, compile the file. Make sure that the resulting .class file is also in the MyPack directory. Then try executing the Account class, using the following command line:

```
java MyPack.Account
```

It will not work as Account class does not have a main method defined. But it is a valid Java class.

Importing a Package

Given that packages exist and are a good mechanism for compartmentalizing diverse classes from each other, it is easy to see why all of the built-in Java classes are stored in packages. There are no core Java classes in the unnamed default package; all of the standard classes are stored in some named package. Since classes within packages must be fully qualified with their package name or names, it could become tedious to type in the long dot-separated package path name for every class you want to use. For this reason, Java includes the import statement to bring certain classes, or entire packages, into visibility. Once imported, a class can be referred to directly, using only its name. The import statement is a convenience to the programmer and is not technically needed to write a complete Java program. If you are going to refer to a few dozen classes in your application, however, the import statement will save a lot of typing.

In a Java source file, import statements occur immediately following the package statement (if it exists) and before any class definitions. This is the general form of the import statement:

```
import pkg1[.pkg2].(classname|*);
```

Here, pkg1 is the name of a top-level package, and pkg2 is the name of a subordinate package inside the outer package separated by a dot (.). There is no practical limit on the depth of a package hierarchy, except that imposed by the file system. Finally, you specify either an explicit classname or a star (*), which indicates that the Java compiler should import the entire package. This code fragment shows both forms in use:

```
import java.util.Date;
import java.io.*;
```

All of the standard Java classes included with Java are stored in a package called java. The basic language functions are stored in a package inside of the java package called java.lang. Normally, you have to import every package or class that you want to use, but

since Java is useless without much of the functionality in java.lang, it is implicitly imported by the compiler for all programs. This is equivalent to the following line being at the top of all of your programs:

```
import java.lang.*;
```

If a class with the same name exists in two different packages that you import using the star form, the compiler will remain silent, unless you try to use one of the classes. In that case, you will get a compile-time error and have to explicitly name the class specifying its package.

Any place you use a class name, you can use its fully qualified name, which includes its full package hierarchy. For example, this fragment uses an import statement:

```
import java.util.*;
class MyDate extends Date {   }
```

The same example without the import statement looks like this:

```
class MyDate extends java.util.Date {  }
```