Programming in Java

Lecture 12: Packages

Mahesh Kumar
Assistant Professor (Adhoc)

Department of Computer Science
Acharya Narendra Dev College
University of Delhi

Course webpage
[http://www.mkbhandari.com/mkwiki]

Outline

1. Introduction to Packages
2. Access Protection
3. Importing Packages

Packages

- A package (act as *Container*) is a collection of related java entities (such as classes, interfaces, exceptions, errors and enums), a great way to achieve reusability, can be considered as means to achieve data encapsulation.
- Packages in Java provides a mechanism for partitioning the class name space into more manageable chunks.
- The Package is both a naming and a visibility control mechanism.
- The advantages of packages are:
  - *Removes naming collision:* by prefixing the class name with a package name.
  - *Provides access control:* Besides *public* and *private*, Java has two access control modifiers- *protected* and *default (*that are related to package).
  - *Categorize the classes and interfaces so that they can be easily maintained.*
- Packages are stored in a hierarchical manner and are explicitly imported into new class definitions.

Packages

- Package in java can be categorized in two form, built-in package and user-defined package.

- Built-in packages: standard packages which are part of JRE or Java API. Some of the commonly used built-in packages are:

java.lang	Contains language support classes (for e.g classes which defines primitive data types, math operations, etc.) . This package is automatically imported.
java.io	Contains classes for supporting input / output operations.
java.util	Contains utility classes which implement data structures like Linked List, Hash Table, Dictionary, etc and support for Date / Time operations.
java.applet	Contains classes for creating Applets.
java.awt	Contains classes for implementing the components of graphical user interface (like buttons, menus, etc.).
java.net	Contains classes for supporting networking operations.

Defining a Package

- To create a package is quite easy: simply include a package statement 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, and suitable for short, sample programs but inadequate for real applications.
- Most of the time, for real applications, you will define a package for your code, using the general form:

```
package pkg;          // for example,  package MyPackage;
```

Defining a Package

- *package MyPackage;* // creates a package called MyPackage;

1. Java uses file system directories to store packages.

2. 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.

3. 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.

4. 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 is:

   package pkg1[.pkg2[.pkg3]];

Defining a Package

- *package MyPackage;* // creates a package called MyPackage;

5. 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 in a UNIX environment.

6. Be sure to choose your package names carefully. You cannot rename a package without renaming the directory in which the classes are stored.

Package Example

```
// A simple package
package MyPack;
class Balance {
        String name;
        double bal;
        Balance(String n, double b) {
                name = n;
                bal = b;
        }
        void show( ) {
                if(bal < 0 )
                        System.out.print("--> ");
                System.out.println(name + ": $" + bal);
        }
}
class AccountBalance {
        public static void main(String args[ ]) {
                Balance current[ ] = new Balance[3];
                current[0] = new Balance("K. J. Fielding", 123.23);
                current[1] = new Balance("Will Tell", 157.02);
                current[2] = new Balance("Tom Jackson", -12.33);
                for(int i=0; i<3; i++) {      current[i].show( );         }
        }
}
```

1. How to compile Java Package program
   - *Syntax:* javac -d directory javafilename
   - *Example:* javac -d . AccountBalance.java

     //creates package MyPack in current directory (.) and saves the generated .class files(Balance.class and AccountBalance.class) in it. *This step can be performed manually as well.*

2. How to run Java Package program.
   - java MyPack.AccountBalance

This program displays the following output :

```
K. J. Fielding: $123.23
Will Tell: $157.02
--> Tom Jackson: $-12.33
```

** 3 ways are there to locate/run Java Packages program (other two are discussed in next slide)*

Finding Packages and CLASSPATH

- How does the Java run-time system know where to look for packages that you create?

1. By default, the Java run-time system uses the current working directory as its starting point. Thus, if your package is in a subdirectory of the current directory, it will be found.
   - Already Discussed in previous slide

2. You can specify a directory path or paths by setting the **CLASSPATH** environmental variable. (in Unix/Linux systems)
   - export CLASSPATH=.:/home/UserName/Desktop/MyJavaPrograms;
     // Assuming your packages are saved under Desktop/MyJavaPrograms;

3. You can use the **-classpath** option with **java** and **javac** to specify the path to your classes.
   - java -classpath /home/UserName/Desktop/MyJavaPrograms/ MyPack.AccountBalance

     // Assuming your packages are saved under Desktop/MyJavaPrograms;
     // Save all .class files of your program(AccountBalance.java) in MyPack.

Access Protection

- Packages act as containers for classes and other subordinate packages
- Classes act as containers for data and code
- The class is Java's smallest unit of abstraction
- Because of the interplay between classes and packages, Java addresses four categories of visibility for class members:
  1. Subclasses in the same package
  2. Non-subclasses in the same package
  3. Subclasses in different package
  4. Classes that are neither in the same package nor subclasses

Access Protection

- The three access modifiers, private, public, and protected, provide a variety of ways to produce the many levels of access required by these categories.

- The following applies only to members of classes

	Private	No Modifier	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

[Source: (1)]

Access Protection

- Anything declared public can be accessed from anywhere.
- Anything declared private cannot be seen outside of its class.
- When a member does not have an explicit access specification, it is visible to subclasses as well as to other classes in the same package (default access).
- If you want to allow an element to be seen outside your current package, but only to classes that subclass your class directly, then declare that element protected.

Access Protection

- A non-nested class has only two possible access levels
  - default and public (others are abstract and final)
- When a class is declared as public, it is accessible by any other code.
- If a class has default access, then it can only be accessed by other code within its same package.
- When a class is public, it must be the only public class declared in the file, and the file must have the same name as the class

An Access Example

// Shows all combinations of the access control modifiers.
// This example has **two packages** and **five classes.**

This is file **Protection.java:**

```
package p1;

public class Protection {
        int n = 1;
        private int n_pri = 2;
        protected int n_pro = 3;
        public int n_pub = 4;

        public Protection( ) {
                System.out.println("base constructor");
                System.out.println("n = " + n);
                System.out.println("n_pri = " + n_pri);
                System.out.println("n_pro = " + n_pro);
                System.out.println("n_pub = " + n_pub);
        }
}
```

This is file **Derived.java:**

```
package p1;

class Derived extends Protection {

        Derived( ) {
                System.out.println("derived constructor");
                System.out.println("n = " + n);

                // private member in Protection class
                // System.out.println("n_pri = "+ n_pri);

                System.out.println("n_pro = " + n_pro);
                System.out.println("n_pub = " + n_pub);
        }
}
```

An Access Example

This is file **SamePackage.java:**

```
package p1;

class SamePackage {

        SamePackage( ) {
                Protection p = new Protection( );
                System.out.println("same package constructor");

                System.out.println("n = " + p.n);

                // class only
                // System.out.println("n_pri = " + p.n_pri);

                System.out.println("n_pro = " + p.n_pro);

                System.out.println("n_pub = " + p.n_pub);
        }
}
```

This is test file for package P1, **DemoP1.java:**

```
// Demo package p1.

package p1;

// Instantiate the various classes in p1.
public class DemoP1 {
        public static void main(String args[ ]) {

                Protection ob1 = new Protection( );

                Derived ob2 = new Derived( );

                SamePackage ob3 = new SamePackage( );
        }
}
```

An Access Example

- How to compile?
  1. Compile all classes one by one in sequence:
     ```
     $ javac -d . Protection.java
     $ javac -d . Derived.java
     $ javac -d . SamePackage.java
     $ javac -d . DemoP1.java
     ```

     OR

  2. Compile all classes all together but in sequence
     ```
     $ javac -d . Protection.java Derived.java
         SamePackage.java DemoP1.java
     ```

- How to run?
  - R $ java p1.DemoP1.java

This program displays the following output :

```
base constructor
n = 1
n_pri = 2
n_pro = 3
n_pub = 4
base constructor
n = 1
n_pri = 2
n_pro = 3
n_pub = 4
derived constructor
n = 1
n_pro = 3
n_pub = 4
base constructor
n = 1
n_pri = 2
n_pro = 3
n_pub = 4
same package constructor
n = 1
n_pro = 3
n_pub = 4
```

An Access Example

This is file **Protection2.java:**

```
package p2;

class Protection2 extends p1.Protection {

        Protection2( ) {
                System.out.println("derived other package
                constructor");

                // class or package only
                // System.out.println("n = " + n);

                // class only
                // System.out.println("n_pri = " + n_pri);

                System.out.println("n_pro = " + n_pro);
                System.out.println("n_pub = " + n_pub);
        }
}
```

This is file **OtherPackage.java:**

```
package p2;

class OtherPackage {

        OtherPackage( ) {
                p1.Protection p = new p1.Protection( );
                System.out.println("other package constructor");

                // class or package only
                // System.out.println("n = " + p.n);

                // class only
                // System.out.println("n_pri = " + p.n_pri);

                // class, subclass or package only
                // System.out.println("n_pro = " + p.n_pro);

                 System.out.println("n_pub = " + p.n_pub);
        }
}
```

An Access Example

This is test file for package P2, **DemoP2.java:**

```
// Demo package p2.

package p2;

// Instantiate the various classes in p2.
public class DemoP2 {
        public static void main(String args[ ]) {

                Protection2 ob1 = new Protection2( );

                OtherPackage ob2 = new OtherPackage( );
        }
}
```

This program displays the following output :

```
base constructor
n = 1
n_pri = 2
n_pro = 3
n_pub = 4
derived other package constructor
n_pro = 3
n_pub = 4
base constructor
n = 1
n_pri = 2
n_pro = 3
n_pub = 4
other package constructor
n_pub = 4
```

Importing Packages

- 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. (*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.)*
- The import statement saves a lot of typing. *(If you are going to refer to a few dozen classes in your application)*
- In a Java source file, import statements occur immediately following the packagestatement (if it exists) and before any class definitions.
- The general form of the *import statement:*

  *import pkg1[.pkg2].(classname | *);*
- For example:

```
import java.util.Date;      //Explicit Date class
import java.io.*;           //Entire io package
```

1. Here pkg1 is the top-level package, and pkg2 is the subordinate package inside the outer package separated by a dot (.).
2. There is no practical limit on the depth of a package hierarchy, except that imposed by the file system.

Importing Packages

- 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** (*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.;*
- The import statement is *optional*. Any place you use a class name, you can use its *fully qualified name*, which includes its full package hierarchy.
- For example:

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

- The same example without the import statement looks like this:

```
class MyDate extends java.util.Date {      // fully-qualified name
}
```

Importing Packages

```
/* when a package is imported, only those items within the package declared as public will be available to non-subclasses in the importing code.  */

package MyPack;

/* Now, the Balance class, its constructor, and its show( ) method are public. This means that they can be used by non-subclass code outside their package  */

public class Balance {
        String name;
        double bal;
        public Balance(String n, double b) {
                name = n;
                bal = b;
        }
        public void show( ) {
                if( bal < 0 )
                        System.out.print("--> ");
                System.out.println(name + ": $" + bal);
        }
}
```

```
/* Here TestBalance imports MyPack and is then able to make use of the Balance class:  */

import MyPack.Balance;        //import MyPack.*;

class TestBalance {
        public static void main(String args[ ]) {

                /* Because Balance is public, you may use
                Balance class and call its constructor. */

                Balance test = new Balance("J. J. Jaspers", 99.88);

                test.show( );
        }
}
```

A. Remove the public specifier from the Balance class and then try compiling TestBalance.

References

Reference for this topic

- [Book: Java: The Complete Reference, Ninth Edition: Herbert Schildt]
  https://www.amazon.in/Java-Complete-Reference-Herbert-Schildt/dp/0071808558

- [Web: GeeksforGeeks]
  https://www.geeksforgeeks.org/java/

- [Web: Java T Point tutorial]
  https://www.javatpoint.com/java-tutorial