Object Oriented Programming using Java 7

Packages

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Name Space Management

Name Space Management

- Classes written so far all belong to a single name space: a unique name has to be chosen for each class to avoid name collision
- Java provides a mechanism for partitioning the class name space into more manageable chunks. This mechanism is a package

Package

- A package is both a naming and a visibility control mechanism:
  - divides the name space into disjoint subsets:
    - It is possible to define classes within a package that are not accessible by code outside the package
  - controls the visibility of classes and their members:
    - It is possible to define class members that are only exposed to other members of the same package
- Same-package classes may have an intimate knowledge of each other, but not expose that knowledge to other packages

Package Definition

Package Definition

- Creating a package is quite easy:
  package myPackage;
  class MyClass1 {...}
  class MyClass2 {...}

- The package statement creates a name space where such classes are stored
- When the package statement is omitted, class names are put into the default package which has no name
  *package myPackage; /*file1.java*/*
  class MyClass1 {...}
  class MyClass2 {...}

  *package myPackage; /*file2.java*/*
  class MyClass3{...}

Packages and Directories

- Java uses file system directories to store packages:

  package myPackage;
  class MyClass1 {...}
  class MyClass2 {...}

- The bytecode files MyClass1.class and MyClass2.class must be stored in a directory myPackage

- ***Case is significant! Directory names must match package names exactly***

Package Hierarchy & Package Finding

Package Hierarchy & Package Finding

- To create a package hierarchy, separate each package name with a dot:
  package myPackage1.myPackage2.myPackage3;
- A package hierarchy must be reflected in the file system of your java development system
- ***You cannot rename a package without renaming its directory!***

- As packages are stored in directories, how does the Java run-time system know where to look for packages?
  - **The current directory is the default start point** - if packages are stored in the current directory or sub-directories, they will be found
  - **Specify a directory path or paths** by setting the CLASSPATH environment variable

Package Hierarchy & Package Finding...

Package Hierarchy & Package Finding...

- CLASSPATH - environment variable that points to the root directory of the system's package hierarchy
- A package hierarchy must be reflected in the file system of your java development system
- Several root directories may be specified in CLASSPATH, e.g. the current directory and the C:\ myJava directory: *;C:\ myJava*
- Java will search for the required packages by looking up subsequent directories described in the CLASSPATH variable
- In order for a program to find myPackage, one of the following must be true:
  - *program is executed from the directory immediately above myPackage (the parent of myPackage directory)*
  - *CLASSPATH must be set to include the path to myPackage*

Package Hierarchy & Package Finding...

```
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 Tel ", 157.02);
                current[2] = new Balance("Tom Jackson", -12.3 );
                for (int i=0; i<3; i+ )
                        current[i].show();
                }
        }
```

Package Hierarchy & Package Finding...

Package Hierarchy & Package Finding...

- **Save, compile and execute**:
  - call the file AccountBalance.java
  - save the file in the directory MyPack
  - compile; AccountBalance.class should be also in MyPack
  - set access to MyPack in CLASSPATH variable, or make the parent of MyPack your current directory
  - run: java MyPack.AccountBalance
  - You need to be in the directory above MyPack when executing this command
  - *The .class filename must be qualified with its package name*
  - ***javac -d . AccountBalance.java***

Importing of Packages

Importing of Packages

- Since classes within packages must be fully-qualified with their package names, it would be tedious to always type long dot-separated names
- The import statement allows to use classes or whole packages directly
- **Importing of a concrete class**:
  import myPackage1.myPackage2.myClass;
- **Importing of all classes within a package**:
  import myPackage1.myPackage2.;*

Access Control

Access Control

- Classes and packages are both means of encapsulating and containing the name space and scope of classes, variables and methods
  - packages act as a container for classes and other packages
  - classes act as a container for data and code
- Access control is set separately for classes and class members
- **Access Control: Classes**
  - A class available in the whole program:
    public class MyClass{...}
  - A class available within the same package only:
    class MyClass{...}

Access Control...

Access Control...

- **Access Control: Members**
  - A member is available in the whole program:
    public int variable;
    public int method(...) {...}
  - A member is only available within the same class:
    private int variable;
    private int method(...) {...}
  - A member is available within the same package (default access):
    int variable;
    int method(...) {...}
  - A member is available within the same package as the current class, or within its sub-classes:
    protected int variable;
    protected int method(...) {...}
- *The sub-class may be located inside or outside the current package*

Access Control Summary

Access Control Summary

- Member declared public can be accessed from anywhere
- Member declared private cannot be seen outside its class
- When a member does not have any access specification (default access), it is visible to all classes within the same package
- To make a member visible outside the current package, but only to sub-classes of the current class, declare this member protected

Most Restrictive ← → Least Restrictive

Access Modifiers ->	private	Default/no-access	protected	public
Inside class	Y	Y	Y	Y
Same Package Class	N	Y	Y	Y
Same Package Sub-Class	N	Y	Y	Y
Other Package Class	N	N	N	Y
Other Package Sub-Class	N	N	Y	Y

Access Control...

```
package p1;                                    /* Protection.java */
    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);
                    }
            }
```

Access Control...

```
package p1;                                  /*Derived.java*/
    class Derived extends Protection {
        Derived() {
                System.out.println("derived constructor");
                System.out.println("n = " + n);
                System.out.println("n_pro = " + n_pro);
                System.out.println("n_pub = " + n_pub);
                }
        }
package p1;                                  /*SamePackage.java*/
    class SamePackage {
        SamePackage() {
                Protection p = new Protection();
                System.out.println("same package constructor");
                System.out.println("n = " + p.n);
                System.out.println("n_pro = " + p.n_pro);
                System.out.println("n_pub = " + p.n_pub);
                }
        }
```

Access Control...

```
package p2;                                  /*Protection2.java*/
    class Protection2 extends p1.Protection {
            Protection2() {
                    System.out.println("derived other package");
                    System.out.println("n_pro = " + n_pro);
                    System.out.println("n_pub = " + n_pub);
                    }
    }
package p2;
class OtherPackage {                         /*OtherPackage.java*/
            OtherPackage() {
                    p1.Protection p = new p1.Protection();
                    System.out.println("other package constructor");
                    System.out.println("n_pub = " + p.n_pub);
                    }
            }
```

Access Control...

```
package p1;
        public class Demo {
                public static void main(String args[]) {
                        Protection ob1 = new Protection();
                        Derived ob2 = new Derived();
                        SamePackage ob3 = new SamePackage();
                        }
        }
package p2;
        public class Demo {
                public static void main(String args[]) {
                        Protection2 ob1 = new Protection2();
                        OtherPackage ob2 = new OtherPackage();
                        }
        }
```

Import Statement

Import Statement

- The import statement occurs immediately after the package statement and before the class statement:

  package myPackage;
  import otherPackage1.otherPackage2.otherClass;
  class myClass{...}

- The Java system accepts this import statement by default: *import java.lang.*;*

- This package includes the basic language functions. Without such functions, Java is of no much use

Name Conflict

```
package otherPackage1;
class otherClass{ ... }
```

```
package otherPackage2;
class otherClass{ ... }
```

```
import otherPackage1.*;
import otherPackage2.*;
class myClass{
        ...
        otherClass
        ...
        }
```

Name Conflict...

Name Conflict...

- Compiler will remain silent, unless we try to use otherClass. Then it will display an error message
- In this situation we should use the full name

```
import otherPackage1.*;
import otherPackage2.*;
class myClass{
        ...
        otherPackage1.otherClass
        ...
        otherPackage2.otherClass
        ...
}
```

Short vs. Full References

Short vs. Full References

- **Short reference**:
  import java.util.;*
  class MyClass extends Date {...}

- **Full reference**:
  class MyClass extends java.util.Date {...}

- *Only the public components in imported package are accessible for non-sub-classes in the importing code!*

Static Import

Static Import

- It facilitates the programmer to access any static member of a class directly. There is no need to qualify it by the class name
- Less coding is required if you have access any static member of a class oftenly. But, if static import feature is overused, it makes the program unreadable and unmaintainable
- **Importing a particular member**:
  import static packageName.ClassName.memberName;
- **Importing all static members**:
  import static packageName.ClassName.;*

```
import static java.lang.System.*;
class Test{
  public static void main(String []args){
    out.println("Welcome");
  }
}
```