Programming Language II
CSE-215

Prof. Dr. Mohammad Abu Yousuf
yousuf@juniv.edu

Java Packages

What is Packages?

- Packages are used in Java in order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces etc.
- *Packages are containers for classes. They 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.**
- You can define classes inside a package that are not accessible by code outside that package.
- You can also define class members that are exposed only to other members of the same package.

Defining a Package

- Include a **package command as the first** statement in a Java source file.

- There can be only one package statement in each source file, and it applies to all types in the 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. (This is why you haven't had to worry about packages before now.)

Defining a Package

- 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;
- More than one file can include the same **package statement.**

Defining a Package

- 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 in a Windows environment.**

A Short 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();
  }
}
```

A Short Package Example

- Call this file **AccountBalance.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 **AccountBalance class, using the following command** line:

  java MyPack.AccountBalance
- Remember, you will need to be in the directory above **MyPack when you execute this** command.
- As explained, **AccountBalance is now part of the package MyPack. This means that** it cannot be executed by itself. That is, you cannot use this command line:

  java AccountBalance
- **AccountBalance must be qualified with its package name.**

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.

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

Access Protection

While Java's access control mechanism:

- 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. This is the 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

	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

Class Member Access

Example of Access Protection

- This example has two packages (**p1 and p2**) and five classes.

- The source for the first package defines three classes: **Protection, Derived, and SamePackage.**

- In the first class**,** The variable **n is declared with the default protection, n_pri is private, n_pro is protected, and n_pub is public.**

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);
   }
}
```

- The second class, **Derived, is a subclass of Protection in the same package, p1. This** grants **Derived access to every variable in Protection except for n_pri, the private one.**

- The third class, **SamePackage, is not a subclass of Protection, but is in the same** package and also has access to all but **n_pri.**

This is file **Derived.java**

```
package p1;

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

// class only
// System.out.println("n_pri = "4 + n_pri);

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

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);
  }
}
```

- Following is the source code for the other package, **p2. The two classes defined in p2** cover the other two conditions that are affected by access control.

- The first class, **Protection2, is a subclass of p1.Protection. This grants access to all of p1.Protection's** variables except for **n_pri (because it is private) and n, the variable declared with the** default protection. Remember, the default only allows access from within the class or the package, not extra-package subclasses.

- Finally, the class **OtherPackage has access to** only one variable, **n_pub, which was declared public.**

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);
  }
}
```

- Try these two packages, here are two test files you can use. The one for package **p1 is shown here:**

```
// Demo package p1.
package p1;

// Instantiate the various classes in p1.
public class Demo {
  public static void main(String args[]) {
    Protection ob1 = new Protection();

    Derived ob2 = new Derived();
    SamePackage ob3 = new SamePackage();
  }
}
```

- The test file for **p2 is shown**

```
// Demo package p2.
package p2;

// Instantiate the various classes in p2.
public class Demo {
  public static void main(String args[]) {
    Protection2 ob1 = new Protection2();
    OtherPackage ob2 = new OtherPackage();
  }
}
```

Importing Packages

- 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.

- 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.

- In a Java sourcefile, **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.

- It must be emphasized that 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.*

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

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

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

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

- For example, if you want the **Balance class of the package MyPack shown earlier to be** available as a stand-alone class for general use outside of **MyPack, then you will need to** declare it as **public and put it into its own file, as shown here:**

```
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);
  }
}
```

- As you can see, the **Balance class is now public. Also, its constructor and its show()** method are **public, too. This means that they can be accessed by any type of code** outside the **MyPack package. For example, here TestBalance imports MyPack and is** then able to make use of the **Balance class:**

```
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(); // you may also call show()
  }
}
```

Thank you