Objects and Classes

Part 8 – Packages

Chapter 4, Core Java, Volume I

Contents

- Packages
- Class Importation
- Static Imports
- Package Declaration
- Package Test
- Compile & Run
- Package Scope
- Class Path
- How to locate the class files
- Modules

Packages

- Related classes can be organized into packages
  - to separate your work from other codes (to manage complexity)
  - to guarantee the uniqueness of class names
- Avoids name conflict:

  java.util.Date ≠ java.sql.Date

- Java standard library contains:

java.lang	javax.swing
java.util	javax.sql
java.time	javax.xml
java.net	…
. . .	

- For uniqueness of package names
  - Use reverse domain name for your own packages: **com.horstmann.corejava**
- No relationship between nested packages (e.g. java.util and java.util.jar)

Class Importation

- A class can use all classes from its own package and all public classes from other packages
- Two ways to access classes from another package
  - with the fully qualified name:

```
java.time.LocalDate today = java.time.LocalDate.now();
```

  - to use import statements (to avoid tedious repetition)
- Two ways of importation
  - to import whole package:

```
import java.time.*;
LocalDate today = LocalDate.now();
```

  - to import single class:

```
import java.time.LocalDate;
```

Class Importation

- Use the * notation to import a single package
  - Cannot use java.* to import multiple packages
  - Cannot have multiple wildcards (e.g. import java.*.*)
- If two packages import the same class, you get a compile-time error:

```
import java.util.*;
import java.sql.*;
. . .
Date today; // Error--java.util.Date or java.sql.Date?
```

- You can solve the problem by adding a specific import to the wildcard imports:

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

- If you need both Date classes you need to use fully qualified names:

```
var deadline = new java.util.Date();
var today = new java.sql.Date(...);
```

Static Imports

- Imports static fields and methods:

```
import static java.lang.System.*;
...
out.println("Goodbye, World!"); // i.e., System.out
exit(0); // i.e., System.exit
```

- To import a specific method or field:

```
import static java.lang.System.out;
```

- Can be handy for mathematical functions:

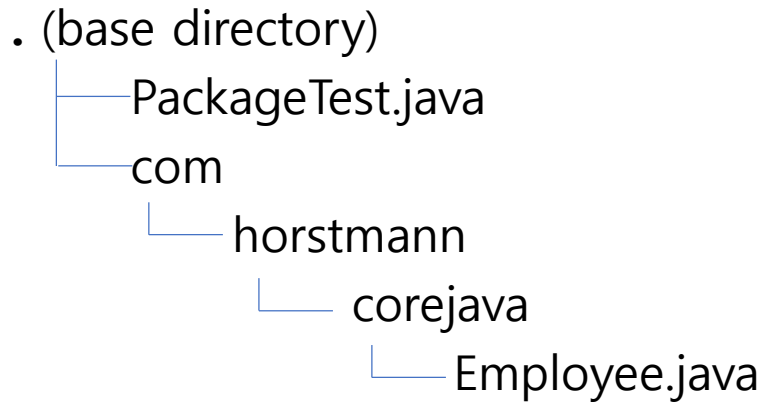
```
import static java.lang.Math.*;
. . .
r = sqrt(pow(x, 2) + pow(y, 2));  (cf. r  = Math.sqrt(Math.pow(x,2)+Math.pow(y,2));
```

Package Declaration

- Put a package declaration at the top of the file:

```
package com.horstmann.corejava;
public class Employee
{
   . . .
}
```

- A class without a package declaration is in the *default package (unnamed package)*.
- Place the source file into a subdirectory that matches the package name.



Package Test

```
package com.horstmann.corejava;
// the classes in this file are part of this package

import java.time.*;
// import statements come after the package statement

/**
 * @version 1.11 2015-05-08
 * @author Cay Horstmann
 */
public class Employee
{
   private String name;
   private double salary;
   private LocalDate hireDay;
   …
}
```

Package Test

```
import com.horstmann.corejava.*;
// the Employee class is defined in that package

import static java.lang.System.*; // static import

public class PackageTest
{
   public static void main(String[] args)
   {
      // because of the import statement, we don't have to use
      // com.horstmann.corejava.Employee here
      Employee harry = new Employee("Harry Hacker", 50000, 1989, 10, 1);

      harry.raiseSalary(5);

      // because of the static import statement, we don't have to use System.out here
      out.println("name=" + harry.getName() + ",salary=" + harry.getSalary());
   }
}
```

Compile and Run

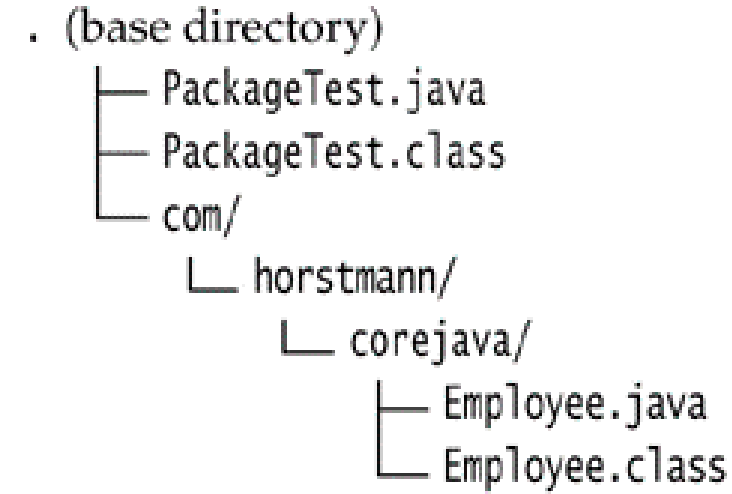
- Compile and run form the base directory:

```
javac com\horstmann\corejava\Employee.java
javac PackageTest.java
java PackageTest
```

- A simpler solution:

```
javac PackageTest.java
java PackageTest
```

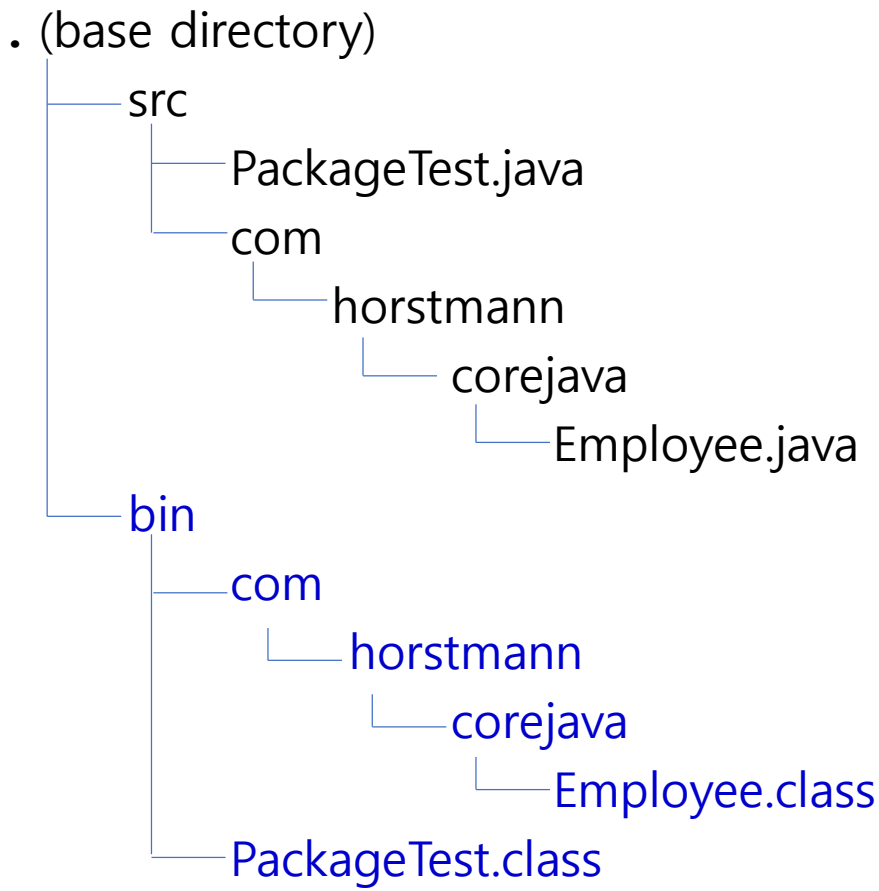
➔ The compiler automatically finds Employee.java



Compile and Run

- Separating the class files from the source directory:
  - make a bin directory

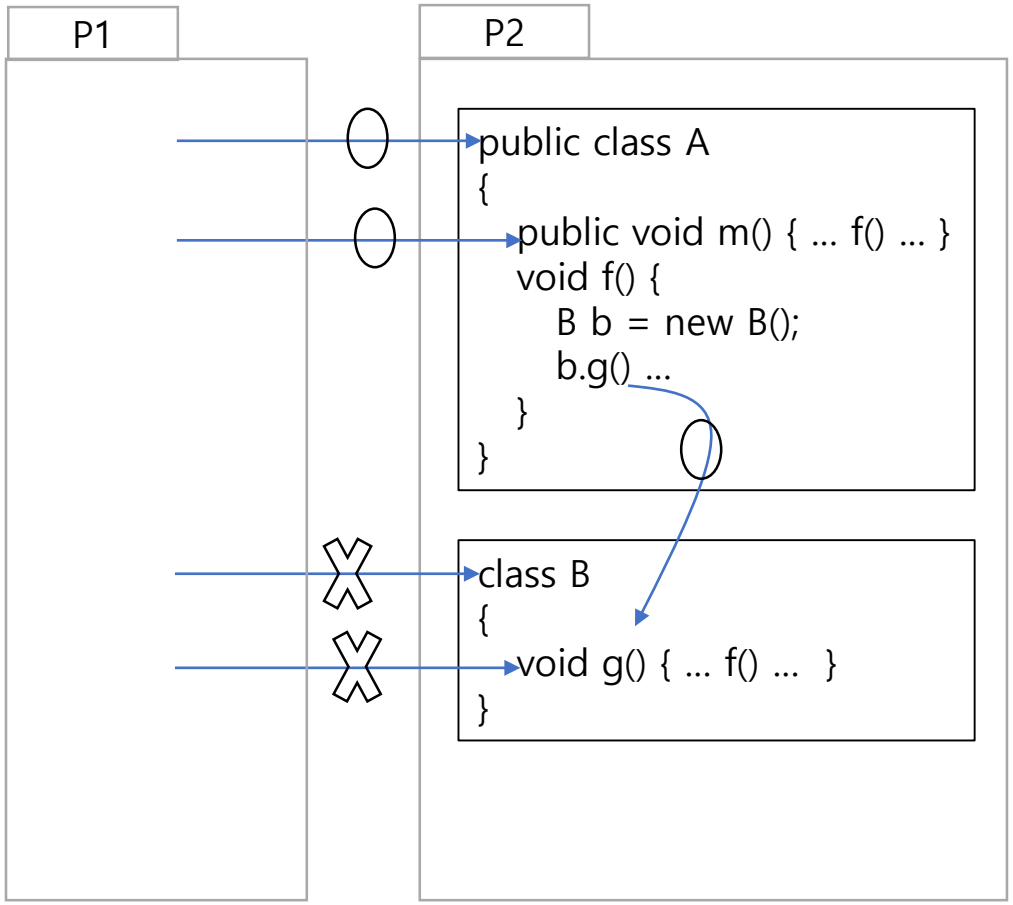
```
javac -d bin -sourcepath src src\PackageTest.java
java -cp bin PackageTest

-d  : create package directories
-sourcepath : source directory
-cp  : class path
```



Package (Default) Scope

- A feature that is not public, private, or protected (see Chapter 5) has package scope.
- The features with package scope can be accessed by all methods in the same package.

	private	package	protected	public
same class	O	O	O	O
same package		O	O	O
subclasses			O	O
any other classes				O



Default Package (Unnamed Package)

- Classes without a package declaration are included in the default (unnamed) package.

```
// EmployeeTest.java
import java.time.*;
class EmployeeTest
{
   public static void main(String[] args)
   {
      // fill the staff array with three Employee objects
      Employee[] staff = new Employee[3];
      staff[0] =
            new Employee("Carl Cracker", 75000, 1987, 12, 15);
      staff[1] =
            new Employee("Harry Hacker", 50000, 1989, 10, 1);
      staff[2] =
            new Employee("Tony Tester", 40000, 1990, 3, 15);
      …
}
```

```
// Employee.java
class Employee
{
   private String name;
   private double salary;
   private LocalDate hireDay;

   Employee(String n, double s, int year, int month,
      int day)
   {   … }
   String getName()  {  …  }
   double getSalary() { … }
   LocalDate getHireDay() { … }
   void raiseSalary() { … }
}
```

The Class Path

- Class path=list of directories and JAR files in which class files or packages are located
  - JAR files are zip files containing class files.
  - Directories are base directories(such as `C:\classdir`), containing package directories (such as `com\horstmann\corejava`).
- Class path elements are separated by : (Unix) or ; (Windows).
- Can include current directory as . (dot)
- Setting the class path (Windows)
  - With `-classpath` option in a command line:

    ```
    java -classpath C:\home\classdir;.;C:\archives/archive.jar MyProg
    ```
  - To set the environment variable `CLASSPATH` permanently

How to locate the class files

- Let's consider the sample class path:
  C:\classdir;.;C:\archives\archives.jar
- Suppose the JVM searches for the class file com.horstmann.corejava.Employee.class
- The order of searches:
  - in system class files (jre\lib, \jre\lib\ext)
  - C:\classdir\com\horstmann\corejava\Employee.class
  - .\com\horstmann\corejava\Employee.class
  - com\horstmann\corejava\Employee.class inside C;\archives\archives.jar
- Let's consider the source file including the following imports:
  - import java.util.*;
  - import com.horstmann.corejava.*;
- What does the java compiler do to find a referred class e.g. Employee without a specifying a package:
  - java.lang.Employee (java.lang is imported by default)
  - java.util.Employee
  - com.horstmann.corejava.Employee

  - It searches the class for each of these classes in all of the locations in the class path.

Modules

- Modules are collections of packages.
- Implementation packages can be encapsulated.
- “Package private” features are not visible outside the module.
- An important feature for programming in the large.