Packages

Java™ How to Program, 9/e

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

Creating Packages

- Each class in the Java API belongs to a package that contains a group of related classes.
- Packages are defined once, but can be imported into many programs.
- Packages help programmers manage the complexity of application components.
- Packages facilitate software reuse by enabling programs to import classes from other packages, rather than copying the classes into each program that uses them.
- Packages provide a convention for unique class names, which helps prevent class-name conflicts.

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

Creating Packages (Cont.)

- The steps for creating a reusable class:
- Declare a `public` class; otherwise, it can be used only by other classes in the same package.
- Choose a unique package name and add a package declaration to the source-code file for the reusable class declaration.
  - In each Java source-code file there can be only one `package` declaration, and it must precede all other declarations and statements.
- Compile the class so that it's placed in the appropriate package directory.
- Import the reusable class into a program and use the class.

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

Creating Packages (Cont.)

- Placing a `package` declaration at the beginning of a Java source file indicates that the class declared in the file is part of the specified package.
- Only `package` declarations, `import` declarations and comments can appear outside the braces of a class declaration.
- A Java source-code file must have the following order:
  - a `package` declaration (if any),
  - `import` declarations (if any), then
  - class declarations.
- Only one of the class declarations in a particular file can be `public`.
- Other classes in the file are placed in the package and can be used only by the other classes in the package.
- Non-`public` classes are in a package to support the reusable classes in the package.

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

```java
// Fig. 8.15: Time1.java
// Time1 class declaration maintains the time in 24-hour format.
package com.deitel.jhtp.ch08;

public class Time1
{
   private int hour; // 0 - 23
   private int minute; // 0 - 59
   private int second; // 0 - 59

   // set a new time value using universal time; throw an
   // exception if the hour, minute or second is invalid
   public void setTime( int h, int m, int s )
   {
      // validate hour, minute and second
      if ( ( h >= 0 && h < 24 ) && ( m >= 0 && m < 60 ) &&
         ( s >= 0 && s < 60 ) )
      {
         hour = h;
         minute = m;
         second = s;
      } // end if
```

Fig. 8.15 | Packaging class `Time1` for reuse. (Part 1 of 2.)

```java
      else
         throw new IllegalArgumentException(
            "hour, minute and/or second was out of range" );
   } // end method setTime

   // convert to String in universal-time format (HH:MM:SS)
   public String toUniversalString()
   {
      return String.format( "%02d:%02d:%02d", hour, minute, second );
   } // end method toUniversalString

   // convert to String in standard-time format (H:MM:SS AM or PM)
   public String toString()
   {
      return String.format( "%d:%02d:%02d %s",
         ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 ),
         minute, second, ( hour < 12 ? "AM" : "PM" ) );
   } // end method toString
} // end class Time1
```

Fig. 8.15 | Packaging class `Time1` for reuse. (Part 2 of 2.)

Time Class Case Study: Creating Packages (Cont.)

- Every package name should start with your Internet domain name in reverse order.
  - For example, our domain name is `deitel.com`, so our package names begin with `com.deitel`.
  - For the domain name *yourcollege*`.edu`*, the package name should begin with* `edu.`*yourcollege*.
- After the domain name is reversed, you can choose any other names you want for your package.
  - We chose to use `jhtp` as the next name in our package name to indicate that this class is from *Java How to Program*.
  - The last name in our package name specifies that this package is for Chapter 8 (`ch08`).

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

Time Class Case Study: Creating Packages (Cont.)

- Compile the class so that it's stored in the appropriate package.
- When a Java file containing a `package` declaration is compiled, the resulting class file is placed in the directory specified by the declaration.
- The `package` declaration

```
package com.deitel.jhtp.ch08;
```

- indicates that class `Time1` should be placed in the directory

```
com
   deitel
      jhtp
         ch08
```

- The directory names in the `package` declaration specify the exact location of the classes in the package.

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

Time Class Case Study: Creating Packages (Cont.)

- `javac` command-line option `-d` causes the `javac` compiler to create appropriate directories based on the class's `package` declaration.
  - The option also specifies where the directories should be stored.
- Example:

```
javac -d . Time1.java
```

- specifies that the first directory in our package name should be placed in the current directory (`.`).
- The compiled classes are placed into the directory that is named last in the `package` statement.

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

Time Class Case Study: Creating Packages (Cont.)

- The `package` name is part of the fully qualified class name.
  - Class `Time1`’s name is actually `com.deitel.jhtp.ch08.Time1`
- Can use the fully qualified name in programs, or `import` the class and use its simple name (the class name by itself).
- If another package contains a class by the same name, the fully qualified class names can be used to distinguish between the classes in the program and prevent a name conflict (also called a name collision).

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

Time Class Case Study: Creating Packages (Cont.)

- Fig. 8.15, line 3 is a single-type-import declaration
  - It specifies one class to import.
- When your program uses multiple classes from the same package, you can import those classes with a type-import-on-demand declaration.
- Example:

```
import java.util.*; // import java.util classes
```

- uses an asterisk (*) at the end of the `import` declaration to inform the compiler that all `public` classes from the `java.util` package are available for use in the program.
  - Only the classes from package `java-.util` that are used in the program are loaded by the JVM.

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

```java
// Fig. 8.16: Time1PackageTest.java
// Time1 object used in an application.
import com.deitel.jhtp.ch08.Time1; // import class Time1

public class Time1PackageTest
{
   public static void main( String[] args )
   {
      // create and initialize a Time1 object
      Time1 time = new Time1(); // invokes Time1 constructor

      // output string representations of the time
      System.out.print( "The initial universal time is: " );
      System.out.println( time.toUniversalString() );
      System.out.print( "The initial standard time is: " );
      System.out.println( time.toString() );
      System.out.println(); // output a blank line

```

Fig. 8.16 | Time1 object used in an application. (Part 1 of 3.)

```
      // change time and output updated time
      time.setTime( 13, 27, 6 );
      System.out.print( "Universal time after setTime is: " );
      System.out.println( time.toUniversalString() );
      System.out.print( "Standard time after setTime is: " );
      System.out.println( time.toString() );
      System.out.println(); // output a blank line

      // attempt to set time with invalid values
      try
      {
         time.setTime( 99, 99, 99 ); // all values out of range
      } // end try
      catch ( IllegalArgumentException e )
      {
         System.out.printf( "Exception: %s\n\n", e.getMessage() );
      } // end catch

```

Fig. 8.16 | Time1 object used in an application. (Part 2 of 3.)

```java
        // display time after attempt to set invalid values
        System.out.println( "After attempting invalid settings:" );
        System.out.print( "Universal time: " );
        System.out.println( time.toUniversalString() );
        System.out.print( "Standard time: " );
        System.out.println( time.toString() );
    } // end main
} // end class Time1PackageTest
```

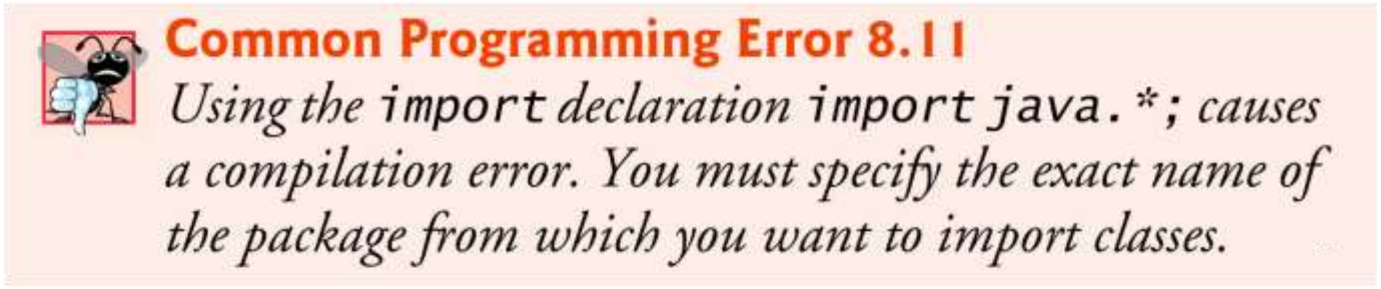
```
The initial universal time is: 00:00:00
The initial standard time is: 12:00:00 AM

Universal time after setTime is: 13:27:06
Standard time after setTime is: 1:27:06 PM

Exception: hour, minute and/or second was out of range

After attempting invalid settings:
Universal time: 13:27:06
Standard time: 1:27:06 PM
```

Fig. 8.16 | `Time1` object used in an application. (Part 3 of 3.)



Common Programming Error 8.11

Using the `import` *declaration* `import java.*;` *causes a compilation error. You must specify the exact name of the package from which you want to import classes.*

Time Class Case Study: Creating Packages (Cont.)

- Specifying the Classpath During **Compilation**
- When compiling a class that uses classes from other packages, `javac` must locate the `.class` files for all other classes being used.
- The compiler uses a special object called a class loader to locate the classes it needs.
  - The class loader begins by searching the standard Java classes that are bundled with the JDK.
  - Then it searches for optional packages.
  - If the class is not found in the standard Java classes or in the extension classes, the class loader searches the classpath, which contains a list of locations in which classes are stored.

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

Time Class Case Study: Creating Packages (Cont.)

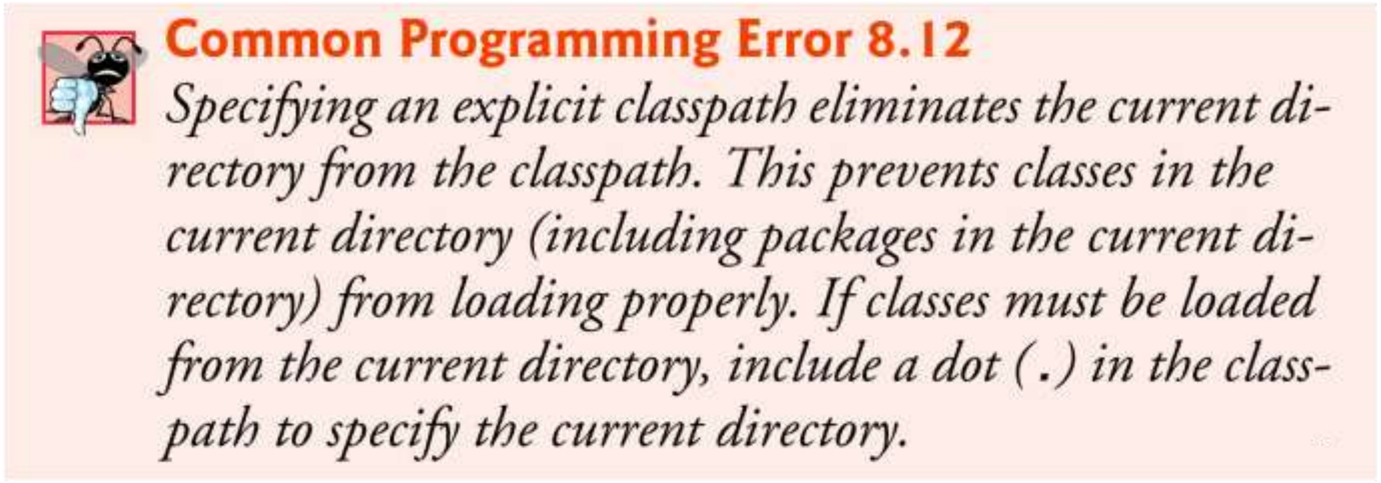
- The classpath consists of a list of directories or archive files, each separated by a directory separator
  - Semicolon (`;`) on Windows or a colon (`:`) on UNIX/Linux/Mac OS X.
- Archive files are individual files that contain directories of other files, typically in a compressed format.
  - Archive files normally end with the `.jar` or `.zip` file-name extensions. **(discussed later)**
- The directories and archive files specified in the classpath contain the classes you wish to make available to the Java compiler and the JVM.

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

Time Class Case Study: Creating Packages (Cont.)

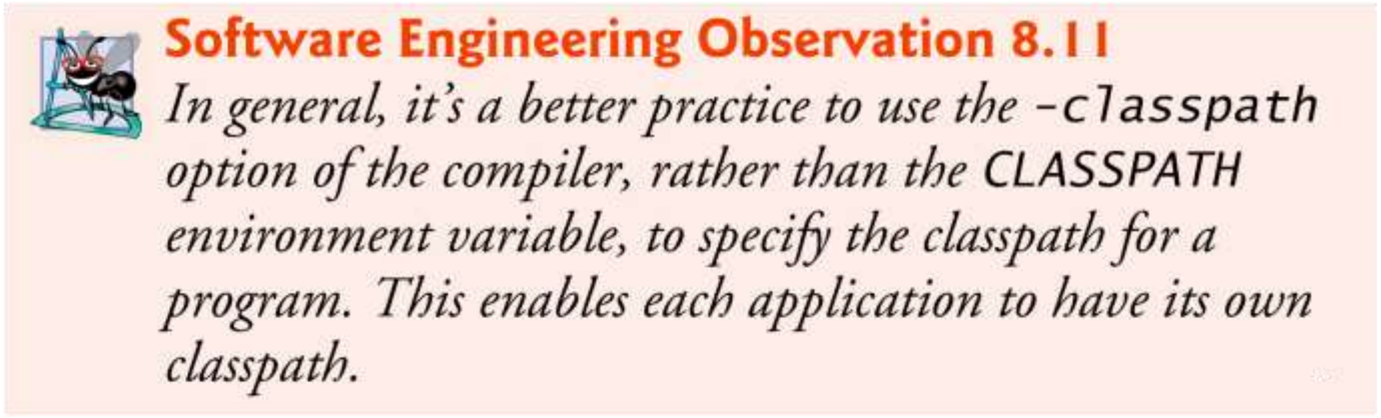
- By default, the classpath consists only of the current directory.
- The classpath can be modified by
  - providing the -classpath option to the `javac` compiler
  - setting the CLASSPATH environment variable (not recommended).
- Classpath
  - `download.oracle.com/javase/6/docs/technotes/tools/index.html#genera`
  - The section entitled "General Information" contains information on setting the classpath for UNIX/Linux and Windows.

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.



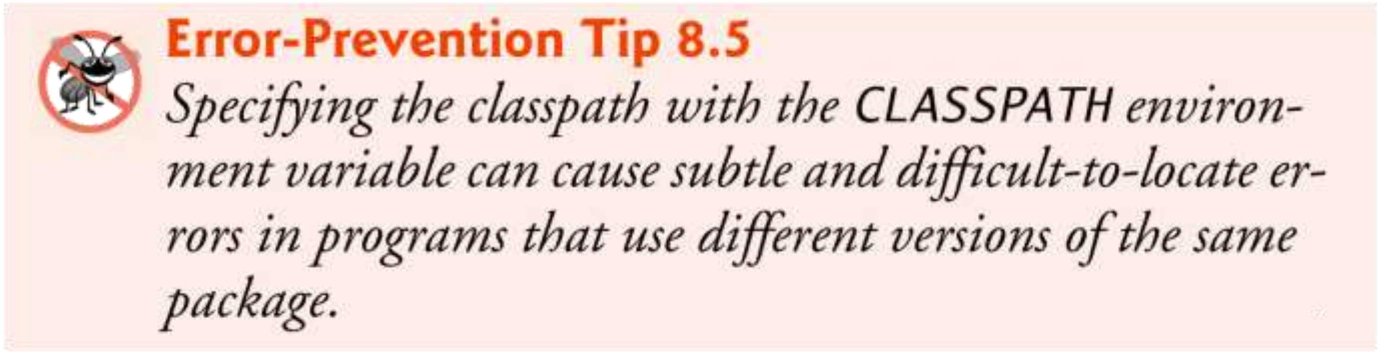
Common Programming Error 8.12

Specifying an explicit classpath eliminates the current directory from the classpath. This prevents classes in the current directory (including packages in the current directory) from loading properly. If classes must be loaded from the current directory, include a dot (.) in the classpath to specify the current directory.



Software Engineering Observation 8.11

In general, it's a better practice to use the `-classpath` *option of the compiler, rather than the* `CLASSPATH` *environment variable, to specify the classpath for a program. This enables each application to have its own classpath.*



Error-Prevention Tip 8.5

Specifying the classpath with the CLASSPATH *environment variable can cause subtle and difficult-to-locate errors in programs that use different versions of the same package.*

Creating Packages (Cont.)

- Specifying the Classpath When **Executing** an Application
- When you execute an application, the JVM must be able to locate the `.class` files of the classes used in that application.
- Like the compiler, the `java` command uses a class loader that searches the standard classes and extension classes first, then searches the classpath (the current directory by default).
- The classpath can be specified explicitly by using either of the techniques discussed for the compiler.
- As with the compiler, it’s better to specify an individual program’s classpath via command-line JVM options.
  - If classes must be loaded from the current directory, be sure to include a dot (`.`) in the classpath to specify the current directory.

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

Package Access

- If no access modifier is specified for a method or variable when it's declared in a class, the method or variable is considered to have package access.
- In a program uses multiple classes from the same package, these classes can access each other's package-access members directly through references to objects of the appropriate classes, or in the case of `static` members through the class name.
- Package access is rarely used.

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

```java
// Fig. 8.17: PackageDataTest.java
// Package-access members of a class are accessible by other classes
// in the same package.

public class PackageDataTest
{
   public static void main( String[] args )
   {
      PackageData packageData = new PackageData();

      // output String representation of packageData
      System.out.printf( "After instantiation:\n%s\n", packageData );

      // change package access data in packageData object
      packageData.number = 77;
      packageData.string = "Goodbye";

      // output String representation of packageData
      System.out.printf( "\nAfter changing values:\n%s\n", packageData );
   } // end main
} // end class PackageDataTest

```

Fig. 8.17 | Package-access members of a class are accessible by other classes in the same package. (Part 1 of 3.)

```java
// class with package access instance variables
class PackageData
{
   int number; // package-access instance variable
   String string; // package-access instance variable

   // constructor
   public PackageData()
   {
      number = 0;
      string = "Hello";
   } // end PackageData constructor

   // return PackageData object String representation
   public String toString()
   {
      return String.format( "number: %d; string: %s", number, string );
   } // end method toString
} // end class PackageData
```

Fig. 8.17 | Package-access members of a class are accessible by other classes in the same package. (Part 2 of 3.)

```
After instantiation:
number: 0; string: Hello

After changing values:
number: 77; string: Goodbye
```

Fig. 8.17 | Package-access members of a class are accessible by other classes in the same package. (Part 3 of 3.)

Access Modifiers in Java

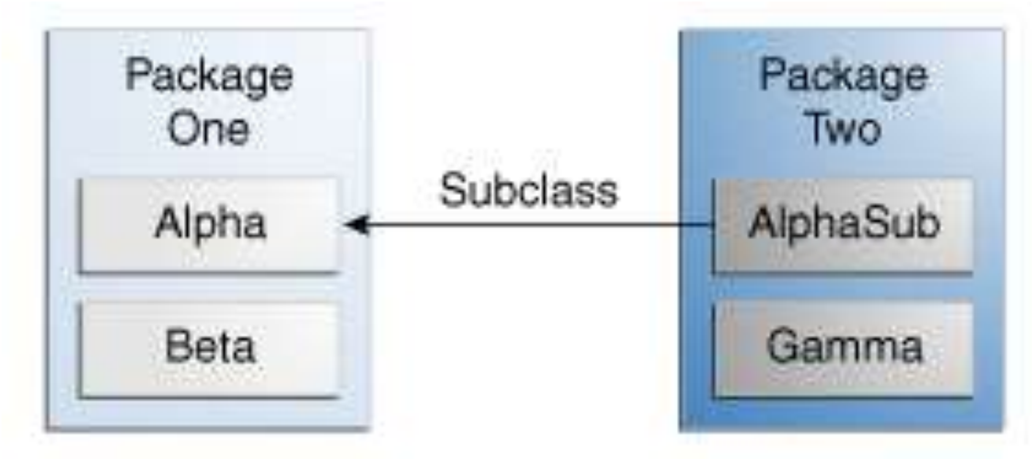
- There are four types of access modifiers available in java:

1. Default – No keyword required
2. Private
3. Protected
4. Public

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.

	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

© Copyright 1992–2012 by Pearson Education, Inc. All Rights Reserved.



The following table shows where the members of the Alpha class are visible for each of the access modifiers that can be applied to them.

Modifier	Alpha	Beta	Alphasub	Gamma
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

© Copyright 1992-2012 by Pearson Education, Inc. All Rights Reserved.