Packages & Classpath (Importance scale: *****, Very important)

Sang Shin Michèle Garoche www.javapassion.com "Learn with Passion!"



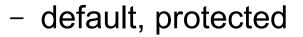
What are Packages?

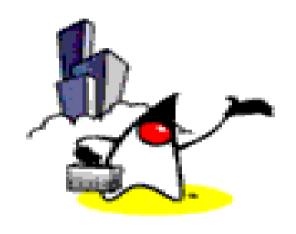
- A package is a grouping of related types providing access protection and name space management
 - Note that types refers to classes, interfaces, enumerations, and annotation types.
 - Types are often referred to simply as classes and interfaces since enumerations and annotation types are special kinds of classes and interfaces, respectively, so types are often referred to simply as classes and interfaces (or simply classes)



Benefits of Packaging

- You and other programmers can easily determine that the classes and interfaces in a package are related.
 - You and other programmers know where to find classes and interfaces that can provide graphics-related functions, for example.
- The names of your classes and interfaces won't conflict with the names in other packages because the package creates a new namespace.
- You can allow classes within a package to have unrestricted access to one another yet still restrict access for types outside the package through access modifiers





Creating a Package

Creating a Package

- To create a package, you choose a name for the package and put a package statement with that name at the top of every source file that contains the types (classes, interfaces, enumerations, and annotation types) that you want to include in the package
- If you do not use a package statement, your type ends up in an unnamed package
 - Use an unnamed package only for small or temporary applications



Placing a Class in a Package

 To place a class in a package, we write the following as the first line of the code (except comments)

```
package <packageName>;
package myownpackage;
```

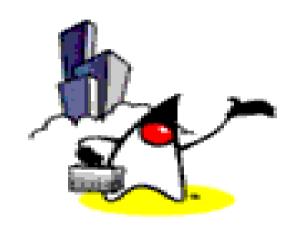


Example: Placing StudentRecord class in SchoolClasses pacakge

```
package SchoolClasses;

public class StudentRecord {
   private String name;
   private String address;
   private int age;
   :
```





Importing and Using classes from other Packages

Importing Classes

- To be able to use classes outside of the package you are currently working in, you need to import those classes.
- By default, all your Java programs import the java.lang.*
 classes, that is why you can use classes like String and
 Integer inside the program even though you haven't
 imported any of them.
- The syntax for importing all classes of a package is as following:

```
import <nameOfPackage>.*;
```



Using Classes from Other Packages

- To use a public package member (classes and interfaces) from outside its package, you must do one of the following
 - Import the package member using import statement
 - Import the member's entire package (all classes under the package) using import statement
 - Refer to the member by its fully qualified name (without using import statement)



Using Classes of other packages with import

```
// "import" statement is required since the Date class is
// used without full qualified path
import java.util.Date;

public static void main(String[] args) {
    Date x = new Date();
}
```



Using Classes of other packages with import

```
// "import" all classes under java.util package
import java.util.*;

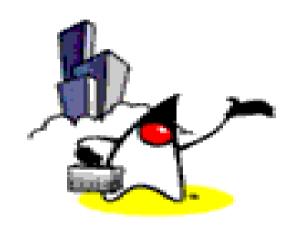
public static void main(String[] args) {
    Date x = new Date();
}
```



Using Classes of other packages via fully qualified path

```
// No import of java.util.Date is required since you
// are using fully qualified path
public static void main(String[] args) {
    java.util.Date x = new java.util.Date();
}
```





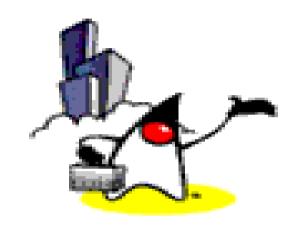
Package Directory Structure

Package & Directory Structure

- Packages can also be nested. In this case, the Java interpreter expects the directory structure containing the classes to match the package hierarchy.
- There should have same directory structure, ./myowndir/myownsubdir/myownpackage directory for the following package statement in a classpath

```
package myowndir.myownsubdir.myownpackage;
```





Managing Sources & Class files

Managing Source and Class Files

Create a *.java file for the class you want to create
 // in the Rectangle.java file
 package graphics;
 public class Rectangle() {

 Place the source file in a directory whose name reflects the name of the package to which the class belongs

-/graphics/Rectangle.java



Directory Structure for the compiled .class files

- The qualified name of the class file and the path name to the Java source file are parallel
- Like the .java source files, the compiled .class files should be in a series of directories that reflect the package name
- Example

```
class name: graphics.Rectangle
```

pathname to source file: graphics/Rectangle.java

pathname to the class file: graphics/Rectangle.class



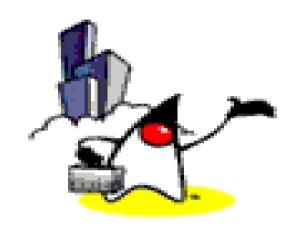
Directory Structure of Java Source Files and Class Files

 However, the path to the .class files does not have to be the same as the path to the .java source files. You can arrange your source and class directories separately, as:

```
<path_one>\sources\com\example\graphics\Rectangle.java
<path_two>\classes\com\example\graphics\Rectangle.class
```

 By doing this, you can give the classes directory to other programmers without revealing your sources





Class path

What is a class path?

- It is a set of directories where Java class files are located
- Java runtime searches for class files in the directories specified in the class path in the order specified



How Java Runtime Find a Class?

- Java runtime tries to find the class files in the following order.
 - In the directories you specify with -classpath or -cp option.
 - n the directories that are specified in the CLASSPATH environment variable (only if you have not specified the -classpath or -cp option)
 - In the currently directory (If you have not specified the -classpath or -cp option and the environment variable CLASSPATH is not set)



- Now, suppose we have a directory schoolClasses representing the package schoolClasses under the C:\ directory - C:\schoolClasses
- We need to set the classpath to point to that directory so that when we try to run it, the JVM will be able to find and load the classes under schoolClasses directory
- Before we discuss how to set the classpath, let us take a look at an example on what will happen if we don't set the classpath.

 Suppose we compile and then run the StudentRecord class we wrote in the last section,

C:\schoolClasses>javac StudentRecord.java

```
C:\schoolClasses>java StudentRecord

Exception in thread "main" java.lang.NoClassDefFoundError: StudentRecord

(wrong name: schoolClasses/StudentRecord)

at java.lang.ClassLoader.defineClass(Native Method)

at java.lang.ClassLoader.defineClass(Unknown Source)

at java.security.SecureClassLoader.defineClass(Unknown Source)

at java.net.URLClassLoader.defineClass(Unknown Source)

at java.net.URLClassLoader.access$100(Unknown Source)

at java.net.URLClassLoader$1.run(Unknown Source)

at java.security.AccessController.doPrivileged(Native Method)

at java.net.URLClassLoader.findClass(Unknown Source)

at java.lang.ClassLoader.loadClass(Unknown Source)

at java.lang.ClassLoader.loadClass(Unknown Source)

at java.lang.ClassLoader.loadClass(Unknown Source)

at java.lang.ClassLoader.loadClass(Unknown Source)
```



 To set the classpath in Windows, we type this at the command prompt,

C:\schoolClasses> set classpath=C:\

- assuming C:\ is the directory in which we have placed the packages meaning there is a directory C:\schoolClasses and there is a C:\schoolClasses\StudentRecord.class
- After setting the classpath, we can now run our program anywhere by typing,

C:\schoolClasses> java
schoolClasses.StudentRecord



 For Unix base systems, suppose we have our classes in the directory /usr/local/myClasses, we write,

export classpath=/usr/local/myClasses

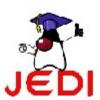


Take note that you can set the classpath anywhere.
 You can also set more than one classpath, we just have to separate them by ;(for windows) and : (for Unix based systems). For example,

```
set classpath=C:\myClasses;D:\;E:\MyPrograms\Java
```

and for Unix based systems,

```
export CLASSPATH=/usr/local/java:/usr/myClasses
```



Thank you!

We do offer Instructor-led Codecamps!
http://www.javapassion.com/codecamps
codecamps@javapassion.com

