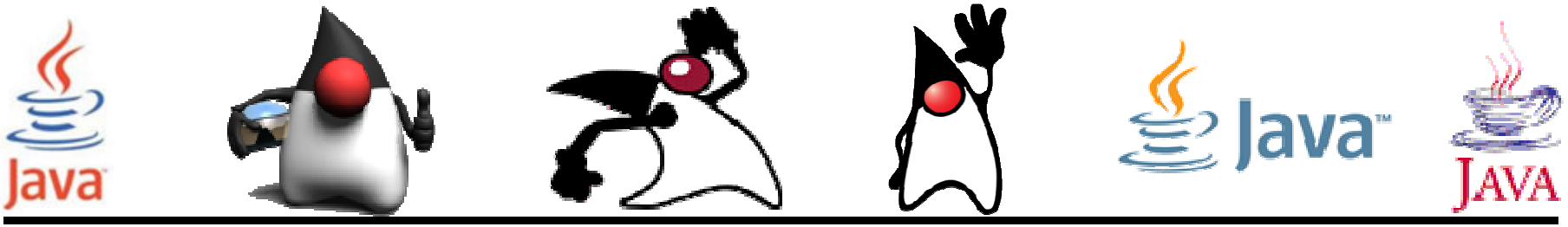

Packages, Classpath y utilidad JAR

Pedro Corcuera

Dpto. Matemática Aplicada y Ciencias de la Computación

Universidad de Cantabria

corcuerp@unican.es

Objetivos

- Aprender a crear e importar packages
- Conocer la utilidad jar para la distribución de aplicaciones

Índice

- Packages
- Ejemplos de Packages en la librería Java
- Beneficios del Packaging
- Creación de un Package
- Nombres de Packages
- Gestión de ficheros fuentes y de clases
- Usando clases de otros paquetes
- Importando paquetes
- Configuración de CLASSPATH
- Utilidad JAR

Packages

- Los programas Java son una colección de clases
  - Cuando los programas son más complejos es útil agrupar conjuntos de clases en paquetes (packages)
- Cada clase pertenece a un package y las clases en un mismo package sirven un mismo propósito
- Los packages son como directorios y las clases en otros packages necesitan ser importados
- La API de Java, por ejemplo, está estructurada como un conjunto de paquetes
  - Se ha usado paquetes mediante el uso de `import`

Importando Packages

- Para usar clases de otro paquete se puede:
  1) Usar su 'nombre completo' (`java.util.Scanner`)
  2) Usar `import` y el nombre de la clase (`Scanner`)

```
public String getInput()
{
  java.util.Scanner keyboard = new java.util.Scanner(System.in);
   . . .
}
```

```
import java.util.*;   // Permite usar todas las clases de java.util

public String getInput()
{
   Scanner keyboard = new Scanner(System.in);
  . . .
}
```

Ejemplos de Packages en la librería Java

Important Packages in the Java Library		
Package	Purpose	Sample Class
java.lang	Language support	Math
java.util	Utilities	Scanner
java.io	Input and output	PrintStream
java.awt	Abstract Windowing Toolkit	Color
java.applet	Applets	Applet
java.net	Networking	Socket
java.sql	Database access through Structured Query Language	ResultSet
javax.swing	Swing user interface	JButton
omg.w3c.dom	Document Object Model for XML documents	Document

Beneficios del Packaging

- Los programadores pueden determinar fácilmente que esas clases e interfaces están relacionadas
- Los programadores conocen dónde encontrar clases e interfaces que pueden ofrecer funciones relacionadas
- Los nombres de las clases e interfaces no entrarán en conflicto con los nombres en otros paquetes porque el paquete crea un nuevo espacio de nombres
- Se puede permitir clases dentro del paquete con acceso sin restricciones a otra

Creación de un Package

- En cada fichero fuente de la clase perteneciente a un package debe colocarse como primera sentencia `package` seguido del nombre del paquete
- Las clases se deben almacenar en una carpeta (directorio) con el mismo nombre del paquete
- Sintaxis:

```
package <nombrePaquete>;
```

- Ejemplo: `package mipropiopackage;`

Nombres de Packages

- Los nombres de los packages se escriben todo en minúscula
- Los nombres de packages se usan para estructurar los espacios de nombres y prevenir colisión de nombres
- Los packages de uso amplio pueden usar el nombre de dominio de Internet del desarrollador al revés

```
package es.unican.macc;
```

Gestión de ficheros fuentes y de clases

- En Java se usa una organización jerárquica para gestionar los ficheros fuentes y de clases
- El código fuente de una clase se coloca en un fichero con nombre igual al de la clase. Ej. `Point.java`

```
package geometry;
public class Point {
    public double x, y;
    . . .
}
```

- El fichero se coloca en un directorio (carpeta) con nombre igual al package

`...\geometry\Point.java`

Gestión de ficheros fuentes y de clases

- El nombre del fichero de clase y el nombre del camino (pathname) al fichero fuente son paralelos.
- Parecido a los ficheros fuente .java, los ficheros .class deben estar en una serie de directorios que reflejan el nombre del package.
- Ejemplo:

pathname al fichero fuente: `geometry\Point.java`

pathname al fichero clase: `geometry\Point.class`

nombre de clase: `geometry.Point`

Gestión de ficheros fuentes y de clases

- En el caso de usar el nombre de dominio de Internet al revés, cada componente del nombre del package corresponde a un subdirectorio.
- Ejemplo:

Package: `package es.unican.macc.geometry;`

pathname al fichero fuente:

`...\es\unican\macc\geometry\Point.java`

pathname al fichero clase:

`...\es\unican\macc\geometry\Point.class`

Gestión de ficheros fuentes y de clases

- El path a los ficheros .class no tienen que ser los mismos que los ficheros .java
- Se pueden organizar los directorios de fuentes y clases de forma separada

  `<dirsrc>\sources\geometry\Point.java`

  `<dirclass>\classes\geometry\Point.class`
- De esta manera se puede entregar el directorio de clases a terceros sin revelar los fuentes
- El path al directorio de clases se llama *class path* y se configura con la variable de entorno CLASSPATH

Gestión de ficheros fuentes y de clases

- Tanto el compilador como la JVM construyen el path a los ficheros .class añadiendo el nombre del package al class path.
- Ejemplo: si el class path es `<dirclass>\classes`

  y el nombre del paquete es:
  `es.unican.macc.geometry`

  entonces el compilador y la JVM buscan los ficheros .class en

`<dirclass>\classes\es\unican\macc\geometry`

Gestión de ficheros fuentes y de clases

- Si el nombre del paquete es: `geometry`

  el directorio de fuentes `<dirsrc>\sources`

  el directorio de clases `<dircls>\classes`

- Para compilar se debe cambiar al directorio de fuentes y ejecutar el comando de compilación:

```
<dirsrc>\sources> javac geometry/Point.java
```

- Para ejecutar se debe cambiar al directorio de clases y ejecutar el comando de ejecución:

```
<dircls>\classes> java geometry.Point
```

Gestión de ficheros fuentes y de clases

- Para evitar cambiar de directorios se puede utilizar los siguientes parámetros en el comando de compilación

  `-cp <path>` para especificar el directorio de clases

  `-d <dir>` para indicar el direct. de los ficheros de clases

- Para ejecutar una clase que requiere otras clases usar el parámetro cp en el comando de ejecución

  `-cp <path>` para especificar el directorio y ficheros zip/jar de clases

- También se puede asignar *`<dirclass>`* en la variable de entorno `CLASSPATH`

Usando clases de otros paquetes

- Para usar un miembro (clases e interfaces) público de otro paquete de forma externa al paquete, se puede usar una de los siguientes opciones:
  - Importar el miembro del paquete usando la sentencia `import`
  - Importar todos los miembros del paquete usando la sentencia `import`
  - Usar el nombre completo del miembro (sin usar la sentencia import)

Usando clases de otros paquetes

- Para usar un miembro (clases e interfaces) público de otro paquete de forma externa al paquete, se puede usar una de los siguientes opciones:
  - Importar el miembro del paquete usando la sentencia `import NombrePaquete.NombreClase;`
  - Importar todos los miembros del paquete usando la sentencia: `import NombrePaquete.*;`
  - Usar el nombre completo del miembro (sin usar la sentencia import):

    `NombrePaquete.NombreClase`

Importando paquetes

- Las clases fundamentales y más usadas están definidas en el paquete `java.lang` que es importado implícitamente por todos los programas Java (no es necesario importarlo explícitamente).
- Ejemplos de importación y uso de clases:

```
import java.util.Date; //Importando una clase
import java.util.*; //Importando todas las
                    //clases del paquete
...
public static void main(String[] args) {
    java.util.Date x = new java.util.Date();
}
```

Configuración de CLASSPATH

- La variable de sistema CLASSPATH puede contener una lista de directorios y/o jar files. El compilador y la JVM también buscará las clases en esos directorios.
- Configuración de CLASSPATH en Windows (DOS):
  - Para mostrar el contenido de la variable CLASSPATH

```
> set CLASSPATH
```

  - Para borrar el contenido de la variable CLASSPATH

```
> set CLASSPATH=
```

  - Para asignar la variable CLASSPATH

```
> set CLASSPATH=c:\users\libs\java\classes
```

Configuración de CLASSPATH

- Configuración de CLASSPATH en Windows (DOS):
  - Para asignar la variable CLASSPATH con varios paths

```
> set CLASSPATH=path1;path2...
```

- También se puede utilizar: Panel de Control → Sistema → Opciones Avanzadas → Variables de Entorno
- Los comandos javac y java tienen una opción (-cp) para especificar el classpath durante su ejecución

> java –cp c:\java\myclasses utility.myapp.Cool

para ejecutar la clase Cool.class en el package utility.myapp que se encuentra en el directorio c:\java\myclasses

Utilidad JAR

- La utilidad JAR (Java Archive) permite agrupar varios ficheros en uno de forma comprimida.
- Adicionalmente JAR proporciona varios beneficios:
  - Seguridad: se puede firmar digitalmente el contenido del fichero JAR
  - Tiempo de descarga bajo: al descargarse todos los recursos necesarios de una aplicación
  - Compresión: JAR comprime los ficheros
  - Portabilidad: los ficheros JAR son un estándar del API
  - Empaquetado para extensiones, sellado y versionado

Utilidad JAR

- Los ficheros JAR se empaquetan con el formato ZIP
- Se utiliza el comando jar

Operación	Comando
Crear un fichero JAR	`jar cf` *`jar-file input-file(s)`*
Ver contenido de un fichero JAR	`jar tf` *`jar-file`*
Extraer contenido de un fichero JAR	`jar xf` *`jar-file`*
Extraer ficheros específicos de un fichero JAR	`jar xf jar-file archived-file(s)`

Utilidad JAR

- Hay otras opciones como u para actualizar
- El manifiesto es un fichero especial que especifica control de versiones, sellado y clase principal

Operación	Comando
Ejecutar una aplicación empaquetada como fichero JAR (requiere cabecero Main-class del manifiesto)	`java -jar app.jar`

Invocar un applet empaquetado como fichero JAR	`<applet code=AppClassName.class`
	`archive="JarFileName.jar"`
	`width=width height=height>`
	`</applet>`