

Sesión 13: *Packages*

Programación 2

3. Programación modular

Ángel Herranz

2019-2020

Universidad Politécnica de Madrid

En capítulos anteriores

- 👍 Tema 1: Clases y Objetos
- 👍 Tema 2: Colecciones acotadas de Objetos
- 👍 Tema 4: Tipos Abstractos de Datos

API

+

Semántica

nombre de la clase

javadococumentación

y métodos públicos

y tests unitarios

- 🕒 Tema 3: Programación Modular




- Uso del código de otros (bibliotecas)
- Organización de mi código (paquetes)

En el capítulo de hoy




Tema 3: Programación Modular

- Organización de mi código (**paquetes**)
- Más **buenas prácticas**

package geometria;

-  Implementar las clases Punto2D (constructor, observadores de cartesianas y distancia), Circulo y Cuadrado (constructores y observador área) y **empaquetarlas** en un paquete geometria
-  Hacer un programa principal para probar las clases anteriores
-  Compilar y ejecutar

package geometria;

-  Implementar las clases Punto2D (constructor, observadores de cartesianas y distancia), Circulo y Cuadrado (constructores y observador área) y empaquetarlas en un paquete geometria
-  Hacer un programa principal para probar las clases anteriores
-  Compilar y ejecutar ¿Problemas?



Opciones de javac

```
javac -d lib -cp .:lib -sourcepath src TestGeometria.java
```



Opciones de javac

```
javac -d lib -cp .:lib -sourcepath src TestGeometria.java
```

```
javac
```

```
[-d DIRECTORIO]
```

```
[-cp PATH]
```

```
[-sourcepath PATH]
```

```
[FILE] ...
```



Opciones de `javac`

```
javac -d lib -cp .:lib -sourcepath src TestGeometria.java
```

`javac`

`[-d DIRECTORIO]` # Donde se dejan los `.class`

`[-cp PATH]` # Donde se buscan los `.class`

`[-sourcepath PATH]` # Donde se buscan los `.java`

`[FILE] ...` # Ficheros a compilar



Opciones de `javac`

```
javac -d lib -cp .:lib -sourcepath src TestGeometria.java
```

`javac`

`[-d DIRECTORIO]` # Donde se dejan los `.class`

`[-cp PATH]` # Donde se buscan los `.class`

`[-sourcepath PATH]` # Donde se buscan los `.java`

`[FILE] ...` # Ficheros a compilar

Nota: Recuerda que el separador de ficheros en el `PATH` en windows es “;” y que la línea en Windows debería ser:

```
javac -d lib -cp .;lib -sourcepath src TestGeometria.java
```



Opciones de java

```
java -cp .:lib TestGeometria
```



Opciones de java

```
java -cp .:lib TestGeometria
```

```
java  
  [-cp PATH]  
  [CLASS]
```



Opciones de java

```
java -cp .:lib TestGeometria
```

```
java
```

```
[-cp PATH] # Donde se buscan los .class
```

```
[CLASS] # Nombre de la clase principal
```

Organización

```
./
src/
    geometria/
        Circulo.java
        Cuadrado.java
        Punto2D.java
    TestGeometria.java
lib/
```

Organización y compilación

```
javac -d lib src/geometria/Punto2D.java
```

```
./
src/
  geometria/
    Circulo.java
    Cuadrado.java
    Punto2D.java
TestGeometria.java
lib/
```

Organización y compilación

```
javac -d lib src/geometria/Punto2D.java
```

./	./
src/	lib/
geometria/	geometria/
Circulo.java	
Cuadrado.java	
Punto2D.java	Punto2D.class
TestGeometria.java	

Organización y compilación

```
javac -d lib -cp lib src/geometria/Circulo.java
```

./	./
src/	lib/
geometria/	geometria/
Circulo.java	
Cuadrado.java	
Punto2D.java	Punto2D.class
TestGeometria.java	

Organización y compilación

```
javac -d lib -cp lib src/geometria/Circulo.java
```

./	./
src/	lib/
geometria/	geometria/
Circulo.java	Circulo.class
Cuadrado.java	
Punto2D.java	Punto2D.class
TestGeometria.java	

Organización y compilación

```
javac -d lib -cp lib src/geometria/Cuadrado.java
```

./

src/

geometria/

Circulo.java

Cuadrado.java

Punto2D.java

TestGeometria.java

./

lib/

geometria/

Circulo.class

Punto2D.class

Organización y compilación

```
javac -d lib -cp lib src/geometria/Cuadrado.java
```

./

src/

geometria/

Circulo.java

Cuadrado.java

Punto2D.java

TestGeometria.java

./

lib/

geometria/

Circulo.class

Cuadrado.class

Punto2D.class

Organización y compilación

```
javac -d lib -cp lib TestGeometria.java
```

./

src/

geometria/

Circulo.java

Cuadrado.java

Punto2D.java

TestGeometria.java

./

lib/

geometria/

Circulo.class

Cuadrado.class

Punto2D.class

Organización y compilación

```
javac -d lib -cp lib TestGeometria.java
```

./

src/

geometria/

Circulo.java

Cuadrado.java

Punto2D.java

TestGeometria.java

./

lib/

geometria/

Circulo.class

Cuadrado.class

Punto2D.class

TestGeometria.class

javac es muy listo

```
./
src/
    geometria/
        Circulo.java
        Cuadrado.java
        Punto2D.java
    TestGeometria.java
lib/
```

javac es muy listo

```
javac -d lib -cp lib -sourcepath src TestGeometria.java
```

```
./
```

```
./
```

```
src/
```

```
lib/
```

```
    geometria/
```

```
        Circulo.java
```

```
        Cuadrado.java
```

```
        Punto2D.java
```

```
TestGeometria.java
```

javac es muy listo

```
javac -d lib -cp lib -sourcepath src TestGeometria.java
```

./

./

src/

lib/

geometria/

geometria/

Circulo.java

Circulo.class

Cuadrado.java

Cuadrado.class

Punto2D.java

Punto2D.class

TestGeometria.java

TestGeometria.class

Ejecución

```
C:\ Sesion13> java -cp lib TestGeometria  
(0.0, 0.0)  
(2.0, 2.0)  
Distancia: 2.8284271247461903  
Área círculo: 3.141592653589793  
Área cuadrado: 4.0
```

¿Puedo construir mi propia biblioteca?

- `jar`: programa para *manipular* ficheros `.jar`
- Para construir una biblioteca `geometria.jar`:

```
jar cvf geometria.jar -C lib/ .
```

 - c para **crear** un fichero `.jar`
 - v hacerlo **verboso** (mostrando lo que hace)
 - f `geometria.jar` para indicar el **nombre del fichero** `.jar` a generar
 - -C lib/ para indicar el **directorio** donde está lo que se quiere *empaquetar*
 - . para indicar los **ficheros a empaquetar** (el punto indica **todo**)

Construyendo la biblioteca

```
C:\ Sesion13> jar cvf geometria.jar -C lib/ .  
added manifest  
adding: TestGeometria.class(in = 1126) (out= 623)(deflated 44%)  
adding: geometria/(in = 0) (out= 0)(stored 0%)  
adding: geometria/Cuadrado.class(in = 963) (out= 616)(deflated 36%)  
adding: geometria/Circulo.class(in = 419) (out= 298)(deflated 28%)  
adding: geometria/Punto2D.class(in = 815) (out= 486)(deflated 40%)
```

Distribuir la biblioteca

- Ahora ya podéis distribuir vuestra biblioteca
- Y otro programador o usuario puede usarla
- Por ejemplo:

```
C:\ Sesion13> java -cp geometria.jar TestGeometria  
(0.0, 0.0)  
(2.0, 2.0)  
Distancia: 2.8284271247461903  
Área círculo: 3.141592653589793  
Área cuadrado: 4.0
```

Sobre los **packages** de Java

Creando un paquete

- Basta con que una clase tenga **package** como primera clausula

```
package geometria;  
public class Punto2D {  
    ...  
}
```

- Eso es suficiente para que exista el paquete
geometria

Usando las clases de un paquete i

El paquete debe estar *disponible* (biblioteca en el *classpath*) y entonces hay tres opciones:

1. Se **importa** la clase y se usa su nombre (*unqualified*):

```
import geometria.Punto2D;
import geometria.Circulo;
public class TestGeometria {
    ...
    Punto2D p = new Punto2D(2.0 , 1.0);
    Circulo c = new Circulo(p, 1.0);
    ...
}
```

Usando las clases de un paquete ii

2. Se **importan** todas las clases de un paquete y se usan sus nombres (*unqualified*):

```
import geometria.*;  
  
public class TestGeometria {  
    ...  
    Punto2D p = new Punto2D(2.0 , 1.0);  
    Circulo c = new Circulo(p, 1.0);  
    ...  
}
```


Usando las clases de un paquete iii

3. Se usa **su nombre completo** (*qualified*), sin necesidad de importar aunque se podría:

```
public class TestGeometria {  
    ...  
    geometria.Punto2D p =  
        new geometria.Punto2D(2.0 , 1.0);  
    geometria.Circulo c =  
        new geometria.Circulo(p, 1.0);  
    ...  
}
```

Nombrando un paquete i

- Por **convención de código** entre programadores:

minúsculas

- Los nombres de los paquetes pueden contener el caracter “.”:

`org.joda.time`

- Ese caracter ayuda a organizar los ficheros
`.java` y `.class`

`org/joda/time`

Nombrando un paquete ii

- Para **evitar choques de nombres** con otras bibliotecas, suelen usarse los **nombres de los dominios de internet como prefijos** del nombre de los paquetes:

Institución	Paquete
www.joda.org	org.joda.time org.joda.money
babel.upm.es	es.upm.babel.cclib
www.google.com	com.google.common.base

¿Dudas?

- **Módulos**: término **ubicuo** en la programación, existen en **todos los lenguajes** y son una forma de **agrupar y arquitecturar** nuestro código
- **Packages**: cuando hablamos de paquetes en esta asignatura estamos hablando del **sistema de módulos de Java**
- **Bibliotecas**: las bibliotecas son, en general, **una agrupación de módulos**

¿Cómo crear el `.jar`? i

Para ayudaros a organizar vuestro código y contruir un fichero `.jar` que sea usable y que contenga el código fuente os sugiero lo siguiente:

- Cread un directorio `Sesion13`
- Cread un directorio `Sesion13/src` en el que poner todo el código fuente organizado por paquetes, por ejemplo, la clase `es.upm.texas.Texas` con el juego en el fichero `Texas.java` en el directorio

¿Cómo crear el .jar? ii

Sesion13/src/es/upm/texas, la clase
es.upm.texas.cartas.Naipe con el juego en el
fichero Naipe.java en el directorio
Sesion13/src/es/upm/texas/cartas, etc.

- Desde el directorio Sesion13 compilad el código:

```
$ javac -d lib -cp lib -sourcepath src src/es/upm/texas/Texas.java
```

- Desde el directorio Sesion13 cread el fichero texas.jar con las clases y el código fuente:

¿Cómo crear el .jar? iii

```
$ jar cvf texas.jar -C lib . -C . src
added manifest
adding: es/(in = 0) (out= 0)(stored 0%)
adding: es/upm/(in = 0) (out= 0)(stored 0%)
adding: es/upm/texas/(in = 0) (out= 0)(stored 0%)
adding: es/upm/texas/Texas.class(in = 449) (out= 311)(deflated 30%)
adding: es/upm/texas/cartas/(in = 0) (out= 0)(stored 0%)
adding: es/upm/texas/cartas/Naipe.class(in = 204) (out= 170)(deflated 16%)
adding: src/(in = 0) (out= 0)(stored 0%)
adding: src/es/(in = 0) (out= 0)(stored 0%)
adding: src/es/upm/(in = 0) (out= 0)(stored 0%)
adding: src/es/upm/texas/(in = 0) (out= 0)(stored 0%)
adding: src/es/upm/texas/Texas.java(in = 189) (out= 146)(deflated 22%)
adding: src/es/upm/texas/cartas/(in = 0) (out= 0)(stored 0%)
adding: src/es/upm/texas/cartas/Naipe.java(in = 53) (out= 55)(deflated -3%)
```

¿Cómo crear el `.jar`? iv

- Ahora ya puedes entregar el fichero `texas.jar` y cualquier podría usar sus clases o ejecutar el programa principal:

```
$ java -cp texas.jar es.upm.texas.Texas  
Hagan juego, damas y caballeros  
...
```