

Sesión 12: Módulos

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

¹ *TDD (Test-Driven Development)* o “desarrollo dirigido por las pruebas”
está considerado como una buena práctica

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 y métodos públicos		javadocumentación y tests unitarios (los tests lo primero) ¹


🏠 *SIMs* y “Listas”

¹ *TDD* (*Test-Driven Development*) o “desarrollo dirigido por las pruebas” está considerado como una buena práctica

En el capítulo de hoy

Tema 3: Programación Modular

En el capítulo de hoy



 Tema 3: Programación Modular 

En el capítulo de hoy



- 🕒 Tema 3: Programación Modular ❓
 - Organización de mi código (**paquetes**)
 - Uso del código de otros (**bibliotecas**)
 - Más **buenas prácticas**

Usando bibliotecas

Segundos y meses

-  ¿Cuántos segundos quedan de año?
-  Escribir el mes actual en español y en inglés

Segundos y meses


-  ¿Cuántos segundos quedan de año?
-  Escribir el mes actual en español y en inglés
 - Biblioteca: Joda-Time
 - ¿Cómo puedo usarla?

Usando una biblioteca i

1. Necesitas **descargar** la biblioteca
2. Necesitas **importar** las clases que vayas a usar
3. Necesitas decirle **al compilador** javac dónde está la biblioteca
4. Necesitas decirle a **la máquina abstracta** java dónde está la biblioteca

Descargar

- Normalmente las bibliotecas están disponibles en **formato** `.jar`
- ¿Qué es un `.jar`?
- Un **archivador** con el `.class` de todas las clases de la biblioteca
- El `.jar` puede empaquetar también las fuentes y la documentación pero no es habitual

 Descarga la biblioteca Joda-Time

<https://www.joda.org/joda-time>

Importar

- Para poder usar la biblioteca hay que entenderla
- Normalmente las bibliotecas están documentadas con ejemplos, tutoriales o guías y con javadoc

Q Explorar Joda-Time para buscar ejemplos y documentación

<https://www.joda.org/joda-time>

Importar

- Para poder usar la biblioteca hay que entenderla
- Normalmente las bibliotecas están documentadas con ejemplos, tutoriales o guías y con javadoc

Q Explorar Joda-Time para buscar ejemplos y documentación

<https://www.joda.org/joda-time>

- ¿Difícil? Pues gran parte de vuestro tiempo vais a tener que dedicarlo a buscar bibliotecas de calidad y a aprender a usarlas

Programando: segundos y meses

```
import java.util.Locale;

import org.joda.time.DateTime;
import org.joda.time.Seconds;

public class SegundosYMeses {
    public static void main(String args[]) {
        DateTime ahora = new DateTime();
        DateTime nuevo = ahora.plusYears(1).withDayOfYear(1).withTime(0,0,0,0);
        System.out.println(Seconds.secondsBetween(ahora, nuevo).getSeconds());
        System.out.println(ahora.monthOfYear().getAsText(new Locale("en")));
        System.out.println(ahora.monthOfYear().getAsText(new Locale("es")));
    }
}
```

Compilando

- javac: ¿Dónde está la biblioteca?

Añadir la biblioteca al *CLASSPATH*

- El *CLASSPATH* es un *path*²
- El compilador consulta la variable de entorno
- Pero se puede informar al ejecutar el compilador con la opción *-cp* (*-classpath*)³

```
javac -cp joda-time-2.10.1.jar SegundosYMeses.java
```

²*Path*: lista de directorios o ficheros. Recuerda que en los *paths* el separador es “:” en Unix y “;” en Windows.

³Merece la pena mirar también *-sourcepath*

Ejecutando

Añadir la biblioteca al *CLASSPATH*

- Esta vez, en el *CLASSPATH*, hay que incluir las bibliotecas a usar y los directorios donde están nuestros ficheros `.class` compilados⁴

```
java -cp .:joda-time-2.10.1.jar SegundosYMeses
```

⁴En nuestro caso, el directorio *actual*: `"."`

Usando una biblioteca

- Cuando usas un IDE avanzado tipo Eclipse o tipo IntelliJ IDEA el IDE se encarga de casi todo
- Incluso algunos editores de texto *se encargan de casi todo*
- Cada IDE tiene una forma de decirle dónde están las bibliotecas

Usando una biblioteca: Eclipse

- Ejemplo de instrucciones para añadir una biblioteca a Eclipse:
 1. *Right-click this class folder, and select “Properties”*
 2. *Select “Java Build Path” on the left, and then the “Libraries” tab. Now, click the “Add External JARS...” button*
 3. *Locate and select the JAR file you just downloaded, and then click “Open”*
 4. *Finally, click “OK” to close the dialog box. You will see in your project folder an item called “Referenced Libraries” and the library listed*

Usando una biblioteca: herramientas

- Existen herramientas de construcción automática:
- Algunos ejemplos:
Make, Ant, Maven, Gradle
- Muy inteligentes: descargan, compilan, ejecutan los tests, generan JARs, despliegan, ...
- Yo uso `make` pero no es la herramienta más apropiada para Java

Makefile (usar al final de la sesión)

```
geometria: lib/TestGeometria.class
    java -cp lib TestGeometria
```

```
lib/TestGeometria.class: TestGeometria.java src/geometria/Punto2D.java src/geometria/Circulo.java src/geometria/Cuadrado.java
    javac -d lib -cp lib -sourcepath src TestGeometria.java
```

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

```
segundos: SegundosYMeses.class joda-time-2.10.1.jar
    java -cp .:joda-time-2.10.1.jar SegundosYMeses
```

```
joda-time-2.10.1.jar:
    wget https://github.com/JodaOrg/joda-time/releases/download/v2.10.1/joda-time-2.10.1.jar
```


```
SegundosYMeses.class: SegundosYMeses.java
    javac -cp joda-time-2.10.1.jar SegundosYMeses.java
```

```
clean:
    rm -f SegundosYMeses.class
```


```
veryclean: clean
    rm joda-time-2.10.1.jar
```

Usando **consola.Consola**

- Ir a Moodle
- Ir a “Material complementario y de estudio”
- Descargar y entender la biblioteca `ConsoleIO`

 Implementar un programar principal que lea un entero de la entrada estándar usando la biblioteca **ConsoleIO** y lo imprima en la salida estándar

Descargar la bibliotecas de *TADs*

- Ir a Moodle
- Ir a “Material complementario y de estudio”
-  Descargar y entender la biblioteca de *TADs de la asignatura*

Módulos

¿Qué es un módulo?

Un **módulo** es un *compartimento* de código **usable** por otros programadores

- Todos los lenguajes de programación permiten hacer módulos de una u otra forma
- A veces, incluso **de varias formas**

¿Por qué modularizar?

Entender, buscar, repartir trabajo,
reusar, arquitecturar, producir,
distribuir, ...

¿Cómo modularizar correctamente?

- Es muy muy difícil hacerlo bien
- Pero disponemos de buenas prácticas

¿Cómo modularizar correctamente?

- Es muy muy difícil hacerlo bien
- Pero disponemos de buenas prácticas

David Parnas, Niklaus Wirth, Edsger Dijkstra,
Donal Knuth, Barbara Liskov, Bertrand Meyer,
Martin Odersky, Alan Perlis, Robert C. Martin,
Martin Fowler, ...

Gracias

¿Cómo modularizar correctamente?

- Es muy muy difícil hacerlo bien
- Pero disponemos de buenas prácticas

David Parnas, Niklaus Wirth, Edsger Dijkstra,
Donal Knuth, Barbara Liskov, Bertrand Meyer,
Martin Odersky, Alan Perlis, Robert C. Martin,
Martin Fowler, ...

Gracias

- Y ya hemos usado alguna: ocultación

Módulos en Java

Usar una clase como módulo

- Las clases en Java permiten **encapsular** cosas
- Por ejemplo:

`System.out`

`Math.pow`

`SegundosYMeses.main`

 ¿Qué tienen todos esos elementos en común?

Usar una clase como módulo

- Las clases en Java permiten **encapsular** cosas
- Por ejemplo:

`System.out`

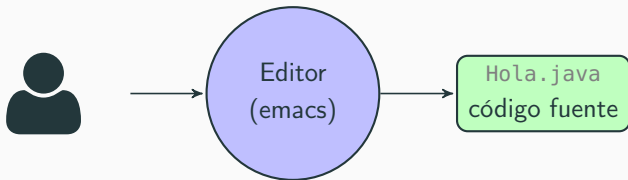
`Math.pow`

`SegundosYMeses.main`

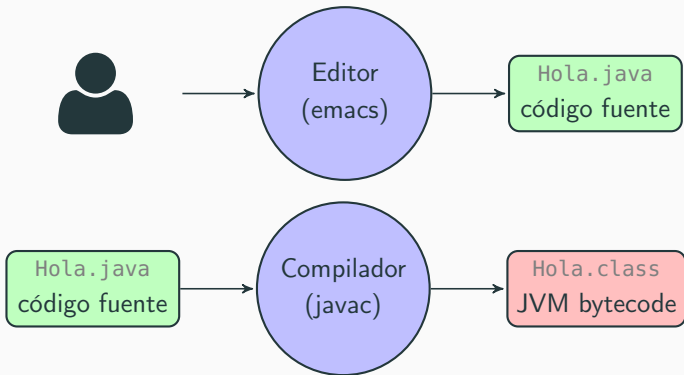
🔊 ¿Qué tienen todos esos elementos en común?

static

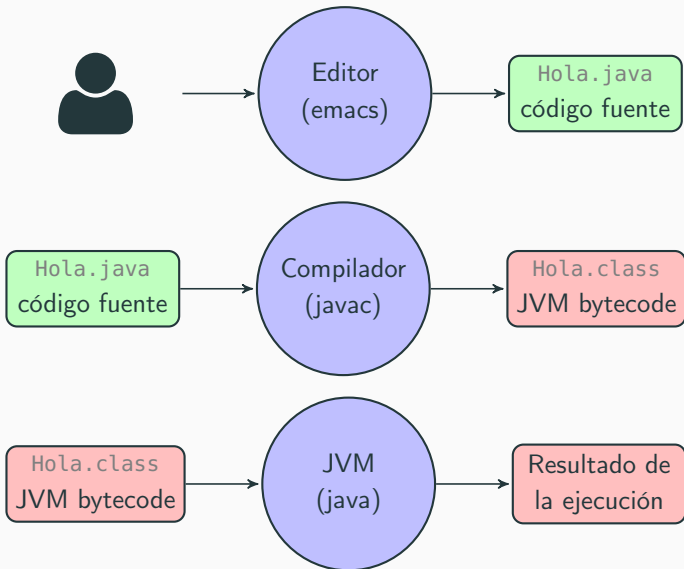
🔔 ¿Cómo funciona Java?






¿Cómo funciona Java?






🔔 ¿Cómo funciona Java?



Módulos “fetén”: **package**


- Los **paquetes** de Java permiten encapsular clases
-  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

Módulos “fetén”: **package**

- Los **paquetes** de Java permiten encapsular clases
-  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?**

Paquetes, directorios y *CLASSPATH*

- Java busca los ficheros `.class` en directorios con los mismos nombres que el nombre del paquete

 Prueba a crear el directorio `geometria` y colocar en él los ficheros `Punto2D.class`, `Cuadrado.class`, y `Circulo.class`
¿Compila ahora?

Paquetes, directorios y *CLASSPATH*

- Java busca los ficheros `.class` en directorios con los mismos nombres que el nombre del paquete



Prueba a crear el directorio `geometria` y colocar en él los ficheros `Punto2D.class`, `Cuadrado.class`, y `Circulo.class`


¿Compila ahora?



¿Te has acordado de *importar* las clases?

Paquetes, directorios y *CLASSPATH*

- Java busca los ficheros `.class` en directorios con los mismos nombres que el nombre del paquete

 Prueba a crear el directorio `geometria` y colocar en él los ficheros `Punto2D.class`, `Cuadrado.class`, y `Circulo.class`

¿Compila ahora?

- ❓ ¿Te has acordado de `importar` las clases?
- ❓ ¿Te has acordado de usar `-cp`?

Paquetes, directorios y *CLASSPATH*

- Java busca los ficheros `.class` en directorios con los mismos nombres que el nombre del paquete



Prueba a crear el directorio `geometria` y colocar en él los ficheros `Punto2D.class`, `Cuadrado.class`, y `Circulo.class`

¿Compila ahora?



¿Te has acordado de **importar** las clases?



¿Te has acordado de usar **-cp**?



Para entender lo que pasa, empezar en un directorio vacío en cada intento

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