

UNIDAD 8

PROGRAMACIÓN ORIENTADA A OBJETOS I


Programación
CFGS DAW


Paco Aldarias
paco.aldarias@ceedcv.es
2020/2021
17/01/21 20:54:30

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

 Importante

 Atención

 Interesante


ÍNDICE DE CONTENIDO


1. Introducción.....	3
2. Fundamentos de una clase.....	4
3. Objetos.....	6
3.1 Instanciación de un objeto (el operador new).....	6
3.2 Asignación de variables de referencia a objetos.....	7
3.3 Recolector de basura.....	7
4. Tipos de acceso a los miembros de una clase (VISIBILIDAD).....	8
5. Métodos.....	9
6. Constructores.....	10
7. Constantes de clase y de objeto.....	11
8. Arrays de objetos.....	12
9. Ejemplos.....	13
9.1 Ejemplo 1.....	13
9.2 Ejemplo 2.....	15
9.3 Ejemplo 3.....	16
9.4 Ejemplo 4.....	20
9.5 Ejemplo 5.....	21
9.6 Ejemplo 6.....	21
10. Agradecimientos.....	23


UD08. PROGRAMACIÓN ORIENTADA A OBJETOS I

1. INTRODUCCIÓN

La Programación Orientada a Objetos (POO) hace que los problemas sean más sencillos, al permitir dividir el problema. Esta división se hace en objetos, de forma que cada objeto funcione de forma totalmente independiente.

 Un **objeto** es un elemento del programa que posee sus propios datos y su propio funcionamiento.

 Una **clase** describe un grupo de objetos que contienen una información similar (atributos) y un comportamiento común (métodos).

 Antes de poder utilizar un objeto, se debe definir su clase. La clase es la definición de un tipo de objeto.

Al definir una clase lo que se hace es indicar cómo funciona un determinado tipo de objetos. Luego, a partir de la clase, podremos crear objetos de esa clase.

Las propiedades de la POO son las siguientes:

- **Encapsulamiento.** Una clase se compone tanto de variables (atributos) como de funciones y procedimientos (métodos). De hecho **no se pueden definir variables (ni funciones) fuera de una clase** (es decir no hay variables globales).
- **Ocultación.** Hay una zona oculta al definir las clases (zona privada) que sólo es utilizada por esa clase y por alguna clase relacionada. Hay una zona pública (llamada también interfaz de la clase) que puede ser utilizada por cualquier parte del código.
- **Polimorfismo.** Cada método de una clase puede tener varias definiciones distintas. En el caso del juego parchís: partida.empezar(4) empieza una partida para cuatro jugadores, partida.empezar(rojo, azul) empieza una partida de dos jugadores para los colores rojo y azul; estas son dos formas distintas de emplear el método empezar, que es polimórfico.
- **Herencia.** Una clase puede heredar propiedades (atributos y métodos) de otra.

2. FUNDAMENTOS DE UNA CLASE

Una clase describe un grupo de objetos que contienen una información similar (atributos) y un comportamiento común (métodos).

Las definiciones comunes (nombre de la clase, los nombres de los atributos, y los métodos) **se almacenan una única vez en cada clase**, independientemente de cuántos objetos de esa clase estén presentes en el sistema.

Una clase es como un molde. A partir de ella se pueden crear objetos.

Es decir antes de poder utilizar un objeto se debe definir la clase a la que pertenece, esa definición incluye:

- **Atributos.** Las variables miembro de la clase. Pueden ser *public* (accesibles desde otra clase), *private* (sólo accesibles por código de su propia clase) o *protected* (accesibles por las subclases).
- **Métodos.** Las funciones miembro de la clase. **Son las acciones u operaciones que puede realizar la clase.** Al igual que **los atributos pueden ser *public*, *private* o *protected*.**

En notación UML¹ la **estructura de una clase** se define así:

Nombre de la clase
Atributos
Métodos

Y la sintaxis de una clase en Java es la siguiente:

```
[acceso] class nombreDeClase {  
    [acceso] [static] tipo atributo1;  
    [acceso] [static] tipo atributo2;  
    //...más atributos...  
  
    [acceso] [static] tipo método1(listaDeParametros) {  
        //...código del método...  
    }  
    [acceso] [static] tipo método2(listaDeParametros) {  
        //...código del método...  
    }  
    //... más métodos...  
}
```

¹ [Unified Modeling Language](#)

Veamos un ejemplo:

La clase `Persona` contiene dos atributos (variables) para almacenar datos sobre una persona (nombre y edad) además de varios métodos (funciones) que hacen cosas con dichos datos.

```
public class Persona {  
    String nombre;  
    int edad;  
  
    // Establece el nombre de la persona  
    void setNombre(String n) {  
        nombre = n;  
    }  
  
    // Establece la edad de la persona  
    void setEdad(int e) {  
        edad = e;  
    }  
  
    // Devuelve el nombre de la persona  
    String getNombre() {  
        return nombre;  
    }  
  
    // Devuelve la edad de la persona  
    int getEdad() {  
        return edad;  
    }  
  
    // Muestra su nombre por pantalla  
    void imprimeNombre() {  
        System.out.println(nombre);  
    }  
  
    // Devuelve true si es mayor de edad, false en caso contrario  
    boolean esMayorEdad() {  
        return (edad >= 18)  
    }  
}
```

Hay que tener en cuenta que **la clase Persona nos servirá para crear tantos objetos Persona como necesitemos, cada uno con su nombre y edad**. Los métodos nos permitirán manipular los datos de cada objeto. Esto se explica en más detalle en el siguiente apartado.

También es importante entender que **cada clase se crea en un archivo Java diferente (con el mismo nombre de la clase), y se utilizan fuera de la clase**.

Por ejemplo, podríamos tener un archivo Persona.java (con la clase Persona del ejemplo anterior) además de un archivo Programa.java (que solo tendrá la función principal `public static void main` a la que estamos acostumbrados). **Desde la función main de Programa podremos crear objetos de tipo Persona además de cualquier otro código que necesitemos**. Esto se verá en ejemplos posteriores.

Los atributos y métodos de una clase se llaman *miembros de una clase*.

Los atributos (variables) de una clase se llaman *variables de instancia* porque cada instancia de la clase (es decir, cada objeto de la clase), contiene sus propias variable atributo. Por lo tanto, los datos de cada objeto son individuales e independiente de los demás objetos.

La palabra opcional `static` sirve para hacer que el método o el atributo a la que precede se pueda *utilizar de manera genérica*² (más adelante se hablará de clases genéricas), los atributos y métodos así definidos se llaman *atributos de clase y métodos de clase* respectivamente.

Ver [ejemplo1](#).

3. OBJETOS

3.1 Instanciación de un objeto (el operador new)

Cuando creamos una clase definimos un nuevo tipo de datos que puede utilizarse para instanciar (crear) objetos de esa clase. La instanciación se hace de la siguiente manera:

- En primer lugar, hay que **declarar una variable del tipo de la clase**.
- En segundo lugar, se necesitará una **copia física del objeto y asignarla a esa variable**. Esto se hace utilizando el operador `new` que asigna dinámicamente (es decir, durante tiempo de ejecución) memoria a un objeto y devuelve una referencia. Esta referencia se almacena en una variable.

Por ejemplo, suponiendo que ya hemos creado la clase Cubo:

```
Cubo c1;           // Crea una variable referencia llamada c1 de tipo Cubo
c1 = new Cubo();    // Crea un objeto Cubo y lo asigna a c1
```

También puede hacerse todo en una sola línea de código:

```
Cubo c1 = new Cubo();
```

² Es decir, se puede hacer uso de una llamada estática sin necesidad de crear una instancia para acceder, por ejemplo, a algún método de la clase.

3.2 Asignación de variables de referencia a objetos

Las variables de referencia a objetos permiten acceder al objeto (son una referencia al objeto, no el objeto). Por ello, actúan de forma diferente a lo que cabría esperar cuando tiene lugar una asignación. Por ejemplo, ¿qué hace el siguiente fragmento de código?

```
Cubo c1 = new Cubo();
```

```
Cubo c2 = c1;
```

Podríamos pensar que a c2 se le asigna una copia del objeto c1, pero no es así. Lo que sucede es que la referencia c1 se copia en c2, por lo que c2 permitirá acceder al mismo objeto referenciado por c1. Por lo tanto cualquier cambio que se haga al objeto referenciado a través de c2 afectará al objeto al que referencia c1.

3.3 Recolector de basura

En Java hay un recolector de basura (*garbage collector*) que se encarga de gestionar los objetos que se dejan de usar y de eliminarlos de memoria. Este proceso es automático e impredecible y trabaja en un hilo (*thread*) de baja prioridad. Por lo general ese proceso de recolección de basura, trabaja cuando detecta que un objeto hace demasiado tiempo que no se utiliza en un programa. Esta eliminación depende de la máquina virtual de Java, en casi todas las recolecciones se realiza periódicamente en un determinado lapso de tiempo.

Ver [ejemplo2](#).

4. TIPOS DE ACCESO A LOS MIEMBROS DE UNA CLASE (VISIBILIDAD)

Los miembros de una clase (atributos y métodos, es decir, sus variables y funciones) pueden definirse como públicos, privados o protegidos. Es importante entender la diferencia:

- **public**: Se puede utilizar desde cualquier clase.
- **private**: Solo puede utilizarlo la propia clase.
- **protected**: Puede utilizarlo la propia clase y también las subclases heredadas (por ahora no la utilizaremos, la herencia de clases se verá en la siguiente unidad).

Se le llama **interfaz** a los miembros de una clase (atributos y métodos) que son **public**, porque son los que permiten interactuar con la clase desde fuera de ella.

Los principios de la programación orientada a objetos dicen que para mantener la encapsulación en los objetos debemos aplicar el especificador **public** a las funciones miembro que formen la interfaz pública y denegar el acceso a los datos miembro usados por esas funciones mediante el especificador **private**.

⚡ En un **paquete**, que es un agrupamiento lógico de clases en un mismo directorio, los atributos y los métodos de estas clases son **públicos** por defecto para el resto de clases existentes en el mismo paquete, y **privados** para cualquier clase que se encuentre fuera (a no ser que se especifique lo contrario).

Cuando un paquete no está definido, se dice que la clase pertenece al paquete por defecto, por lo tanto se aplicará el calificador **public** al resto de las clases cuyos ficheros se encuentren en el mismo directorio.

(Ver [ejemplo 3](#)).

5. MÉTODOS

Se llama método a una función de una clase. Su sintaxis general es la siguiente:

```
tipo nombre_del_método(lista de parámetros) {  
    // cuerpo del método  
}
```

Donde:

- el **tipo** especifica el tipo de datos que devuelve el método. Puede tratarse de cualquier tipo válido, incluyendo los tipos de clases que crea el programador. Si el método no devuelve un valor, el tipo devuelto será *void*.
- el **nombre** del método lo especifica *nombre_del_método*, que puede ser cualquier identificador válido diferente a aquellos ya usados por otros elementos del programa.
- la **lista de parámetros** es una secuencia de pares de tipo e identificador separados por comas. Los parámetros son variables que reciben el valor de los argumentos que se pasa al método cuando se llama. Si el método no tiene parámetros, la lista de parámetros estará vacía.

Cono en cualquier otra función, los métodos devuelven un valor con la sentencia **return** a no ser que el tipo devuelto se defina como void:

```
return valor;
```

Aunque sería perfectamente válido crear una clase que únicamente contenga datos, esto raramente ocurre. La mayoría de las veces es conveniente crear métodos para acceder a los atributos de la clase.

Además de definir métodos que proporcionen acceso a los datos (se oculta o abstrae la estructura interna de los datos) pueden definirse métodos para ser utilizados internamente por la clase.

6. CONSTRUCTORES

Puede resultar una tarea bastante pesada inicializar todas las variables de una clase cada vez que se crea una instancia. Incluso cuando se añaden funciones adecuadas, es más sencillo y preciso realizar todas las inicializaciones cuando el objeto se crea por primera vez. Como el proceso de inicialización es tan común, Java permite que los objetos se inicialicen cuando se crean. Esta inicialización automática se lleva a cabo mediante un constructor.



Un **constructor** es un método especial que no devuelve nunca un valor, siempre devuelve una referencia a una instancia de la clase y es llamado automáticamente al crear un objeto de una clase, es decir al usar la instrucción *new*.

Un *constructor* no es más que un método que tiene el mismo nombre que la clase. En general, cuando se crea una instancia de una clase, no siempre se desea pasar los parámetros de inicialización al construirla, por ello existe un tipo especial de constructores, que son los llamados **constructores por defecto**.

Estos constructores **no llevan parámetros asociados**, e **inician los datos** asignándoles valores por defecto.

Cuando no se llama a un constructor de forma explícita, Java crea un constructor por defecto y lo llama cuando se crea un objeto. Este constructor por defecto asigna a las variables de instancia un valor inicial igual a **cero a las variables numéricas** y *null* a todas las referencias a objetos.

Una vez se defina un constructor para la clase se deja de utilizar el constructor por defecto.



Existe la referencia *this*, que apunta a la instancia que llama al método, y siempre esta accesible, es utilizado por las funciones miembro para acceder a los datos del objeto.

(Ver [ejemplo 4](#)).

7. CONSTANTES DE CLASE Y DE OBJETO

Los miembros constantes se definen en Java a través de la palabra reservada *final*, mientras que los miembros de clase se definen mediante la palabra reservada *static*. De esta manera, si consideramos que los miembros son atributos, tenemos cuatro posibles combinaciones en Java para indicar si un atributo es constante:

- Atributos ***static***: toman valores comunes a todos los objetos existentes y potencialmente variables. Pueden utilizarse aunque no exista ningún objeto instanciado.
- Atributos ***final***: son valores constantes, pero potencialmente distintos en cada una de las instancias. Su valor se inicializa en la fase de construcción del objeto y no pueden ser modificados durante el tiempo de vida de éste.
- Atributos ***static final***: combinan las características de *static* y *final*.
- Resto de atributos (sin *static* ni *final*): atributos variables y diferentes para cada objeto.

⚡ Los atributos ***static*** de la clase deben ser **limitados**, ya que pueden dar lugar a errores de muy difícil depuración, además de ir contra el concepto de la programación orientada a objetos.

Las constantes de objeto se definen mediante la palabra reservada *final*. Se trata de atributos que toman el valor en el constructor del objeto, un valor que no puede ser modificado en el resto del programa. De esta manera, a cada objeto corresponde un atributo de ese tipo, pero invariable para él. Este tipo de atributos sirve para identificar cada objeto de forma única.

Ver [ejemplo 5](#).

8. ARRAYS DE OBJETOS

En Java hay dos sintaxis posibles para definir un **array de objetos**:

```
NombreDeLaClase [] Objetos;  
NombreDeLaClase Objetos[];
```

De esta manera se crea la referencia al array de Objetos.

Para crear la instancia del objeto array, debemos especificar el tamaño deseado con la sintaxis:

```
Objetos = new NombreDeLaClase[n]; // n es el nº de referencias a Objetos
```

Podemos resumir estas dos sentencias en una sola:

```
NombreDeLaClase [] Objetos = new NombreDeLaClase[n];  
NombreDeLaClase Objetos[] = new NombreDeLaClase[n];
```

Hay que tener en cuenta que, cuando se crea el array, el compilador genera una referencia para cada uno de los elementos que lo integran, aunque no existan todavía las instancias de ninguno de ellos.

⚡ Para crear los objetos hay que **llamar** explícitamente al **constructor por cada elemento creado**:

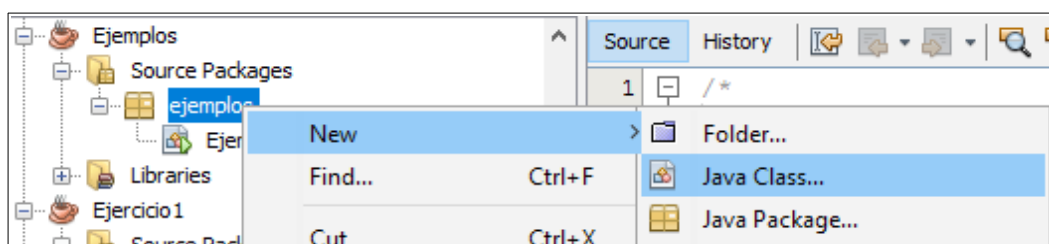
```
Objeto[i] = new NombreDeLaClase();
```

(Ver [ejemplo 6](#)).

9. EJEMPLOS

9.1 Ejemplo 1

En este ejemplo vamos a implementar la clase *Articulos*. Esta clase representa cada objeto con los siguientes atributos: *codigo_articulo*, *titulo*, *formato* y *precio_alquiler*. También define tres métodos que permiten calcular, respectivamente, el precio de alquiler de un día, de dos días y una semana. Lo primero que necesitaremos, y esto lo haremos por cada clase que necesitemos en todos los ejemplos, será crearnos una nueva clase en Java. Para ello pincharemos con el botón derecho sobre el paquete donde vayamos a tener las clases y luego en *NEW > Java Class*. Crearemos una clase llamada *Articulo*.



```
12 public class Articulo {
13     // Atributos de la clase
14     String cod;
15     String titulo;
16     String formato;
17     float precio_alquiler;
18
19     // Métodos de la clase
20     float precio1() {
21         return (precio_alquiler);
22     }
23
24     float precio2() {
25         float precio_total;
26
27         precio_total = precio_alquiler * 1.80f;
28
29         return (precio_total);
30     }
31
32     float precio_semana() {
33         float precio_total;
34
35         precio_total = precio_alquiler * 5;
36
37         return (precio_total);
38     }
39 }
```

El método *precio1* devuelve el valor del precio de alquiler del artículo. El método *precio2* calcula el precio de alquiler de dos días haciendo un descuento del 20% (por eso se multiplica por 1,8). Por último el método *precio_semana* calcula el precio de una semana multiplicando por 5 el precio de alquiler.

Vamos a crear una clase nueva llamada Ejemplos, con la función public static void main, donde vamos a instanciar (crear) objetos de Artículo y utilizarlos.

Con el operador *new* crearemos una instancia de la clase *Articulo*. Como por el momento la clase no tiene constructores se invocará al constructor por defecto de la clase. RECUERDA: Cuando instanciamos una clase estamos creando un objeto de esta clase.

```
12 public class Ejemplos {
13
14     public static void main(String[] args) {
15         // Creamos dos artículos
16         Articulo articulo1 = new Articulo();
17         Articulo articulo2 = new Articulo();
18
19         // Le damos valores a sus atributos
20         articulo1.cod = "001";
21         articulo1.titulo = "Titulo1";
22         articulo1.formato = "DVD";
23         articulo1.precio_alquiler = 2.50f;
24
25         articulo2.cod = "002";
26         articulo2.titulo = "Titulo2";
27         articulo2.formato = "DVD";
28         articulo2.precio_alquiler = 3;
29
30         // Utilizamos sus métodos
31         System.out.println("Alquiler Art. " + articulo1.cod + ", 1 dia:" + articulo1.precio1());
32         System.out.println("Alquiler Art. " + articulo1.cod + ", 2 dias:" + articulo1.precio2());
33         System.out.println("Alquiler Art. " + articulo1.cod + ", 1 semana:" + articulo1.precio_semana());
34         System.out.println("Alquiler Art. " + articulo2.cod + ", 1 dia:" + articulo2.precio1());
35         System.out.println("Alquiler Art. " + articulo2.cod + ", 2 dias:" + articulo2.precio2());
36         System.out.println("Alquiler Art. " + articulo2.cod + ", 1 semana:" + articulo2.precio_semana());
37     }
38 }
39 }
```

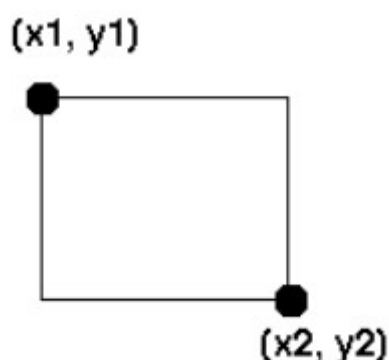
La salida es:

```
run:
Alquiler Art. 001, 1 dia:2.5
Alquiler Art. 001, 2 dias:4.5
Alquiler Art. 001, 1 semana:12.5
Alquiler Art. 002, 1 dia:3.0
Alquiler Art. 002, 2 dias:5.3999996
Alquiler Art. 002, 1 semana:15.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

9.2 Ejemplo 2

En este ejemplo vamos a implementar la clase *Cuadrado*, que representa cuadrados mediante dos coordenadas 2D, y define tres métodos que permiten calcular, respectivamente, la diagonal, el perímetro y el área.

El criterio de representación toma las coordenadas horizontales (x) crecientes de izquierda a derecha, y las verticales (y) crecientes de arriba abajo.



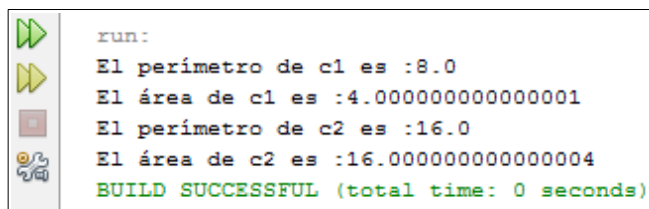
El código de Cuadrado.java sería:

```
12 public class Cuadrado {
13     // Atributos
14     double x1, y1, x2, y2;
15
16     // Métodos
17     double CalcularDiagonal()
18     {
19         return Math.sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
20     }
21
22     double CalcularPerimetro() {
23         double diagonal = CalcularDiagonal();
24         double lado = diagonal/Math.sqrt(2);
25
26         return (4*lado);
27     }
28     double CalcularArea()
29     {
30         double diagonal =CalcularDiagonal();
31
32         return (0.5*diagonal*diagonal);
33     }
34 }
```

El código de Ejemplos.java sería:

```
12 public class Ejemplos {
13
14     public static void main(String[] args) {
15         // Creamos dos cuadrados
16         Cuadrado c1 = new Cuadrado();
17         Cuadrado c2 = new Cuadrado();
18
19         // Les damos valores
20         c1.x1=2; c1.y1=2; c1.x2=4; c1.y2=4;
21         c2.x1=1; c2.y1=1; c2.x2=5; c2.y2=5;
22
23         // Usamos sus métodos
24         System.out.println("El perímetro de c1 es :" + c1.CalcularPerimetro());
25         System.out.println("El área de c1 es :" + c1.CalcularArea());
26         System.out.println("El perímetro de c2 es :" + c2.CalcularPerimetro());
27         System.out.println("El área de c2 es :" + c2.CalcularArea());
28     }
29 }
```

Y la salida:



```
run:
El perímetro de c1 es :8.0
El área de c1 es :4.0000000000000001
El perímetro de c2 es :16.0
El área de c2 es :16.0000000000000004
BUILD SUCCESSFUL (total time: 0 seconds)
```

9.3 Ejemplo 3

En este ejemplo vamos a aplicar el principio de **encapsulamiento** haciendo *private* los atributos de la clase y *public* los métodos. Para ello vamos a modificar la clase Artículo del Ejemplo 1.

Pero ahora no podremos leer ni modificar los atributos de la clase desde fuera de ella (porque son *private*). Así que vamos a definir métodos que nos permitan hacerlo:

- Crearemos métodos *public* (uno por atributo) que nos devuelva el valor de cada atributo. A esto se le llama métodos “**get**” o “**getters**” (del inglés coger).
- Del mismo modo, métodos que nos permitan modificar el valor de los atributos. A esto se le llama métodos “**set**” o “**setters**” (del inglés establecer). En el ejemplo se llama *modificaValores* y permite cambiar todos los valores en una sola llamada.

El código de Artículo.java sería:


```
12 public class Artículo {
13     // Atributos de la clase
14     private String cod;
15     private String titulo;
16     private String formato;
17     private float precio_alquiler;
18
19     // Métodos de la clase
20     public float precio1(){
21         return (getPrecio_alquiler());
22     }
23
24     public float precio2(){
25         float precio_total;
26
27         precio_total = getPrecio_alquiler() * 1.80f;
28
29         return (precio_total);
30     }
31
32     public float precio_semana(){
33         float precio_total;
34
35         precio_total = getPrecio_alquiler() * 5;
36
37         return (precio_total);
38     }
39
40     public void modificaValores(String cod_p, String titulo_p, String formato_p, float precio_p)
41     {
42         cod=cod_p;
43         titulo=titulo_p;
44         formato=formato_p;
45         precio_alquiler=precio_p;
46     }
47
48     public String getCod()
49     {
50         return cod;
51     }
52
53     public String getTitulo()
54     {
55         return titulo;
56     }
57
58     public String getFormato()
59     {
60         return formato;
61     }
62
63     public float getPrecio_alquiler()
64     {
65         return precio_alquiler;
66     }
67 }
```

El código de Ejemplos.java con el main sería:

```

12 public class Ejemplos {
13
14     public static void main(String[] args) {
15         // Creamos dos artículos
16         Artículo articulo1 = new Artículo();
17         Artículo articulo2 = new Artículo();
18
19         // Le damos valores a sus atributos
20         articulo1.modificaValores("001","Titulo1","DVD",2.5f);
21         articulo2.modificaValores("002","Titulo2","DVD",3f);
22
23         // Utilizamos sus métodos
24         System.out.println("Alquiler Art. " + articulo1.getCod() + ", 1 día:" + articulo1.precio1());
25         System.out.println("Alquiler Art. " + articulo1.getCod() + ", 2 días:" + articulo1.precio2());
26         System.out.println("Alquiler Art. " + articulo1.getCod() + ", 1 semana:" + articulo1.precio_semana());
27         System.out.println("Alquiler Art. " + articulo2.getCod() + ", 1 día:" + articulo2.precio1());
28         System.out.println("Alquiler Art. " + articulo2.getCod() + ", 2 días:" + articulo2.precio2());
29         System.out.println("Alquiler Art. " + articulo2.getCod() + ", 1 semana:" + articulo2.precio_semana());
30     }
31
32 }

```

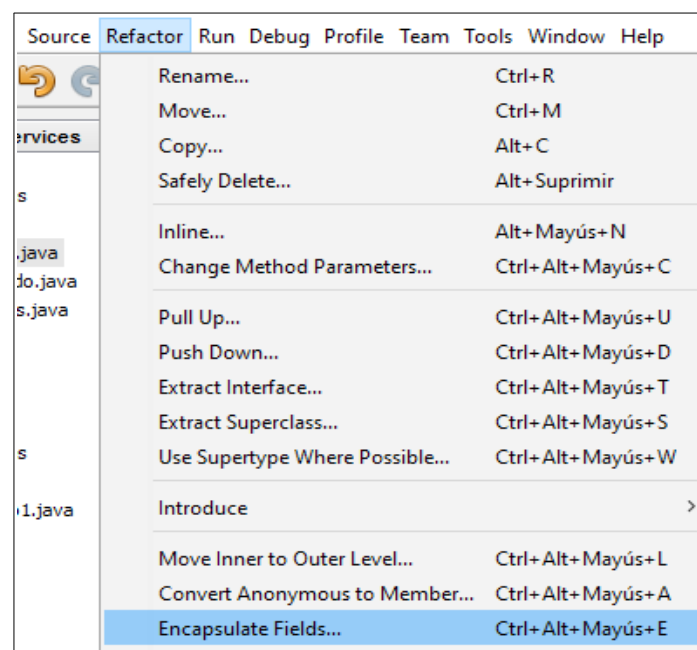
Y la salida:

```

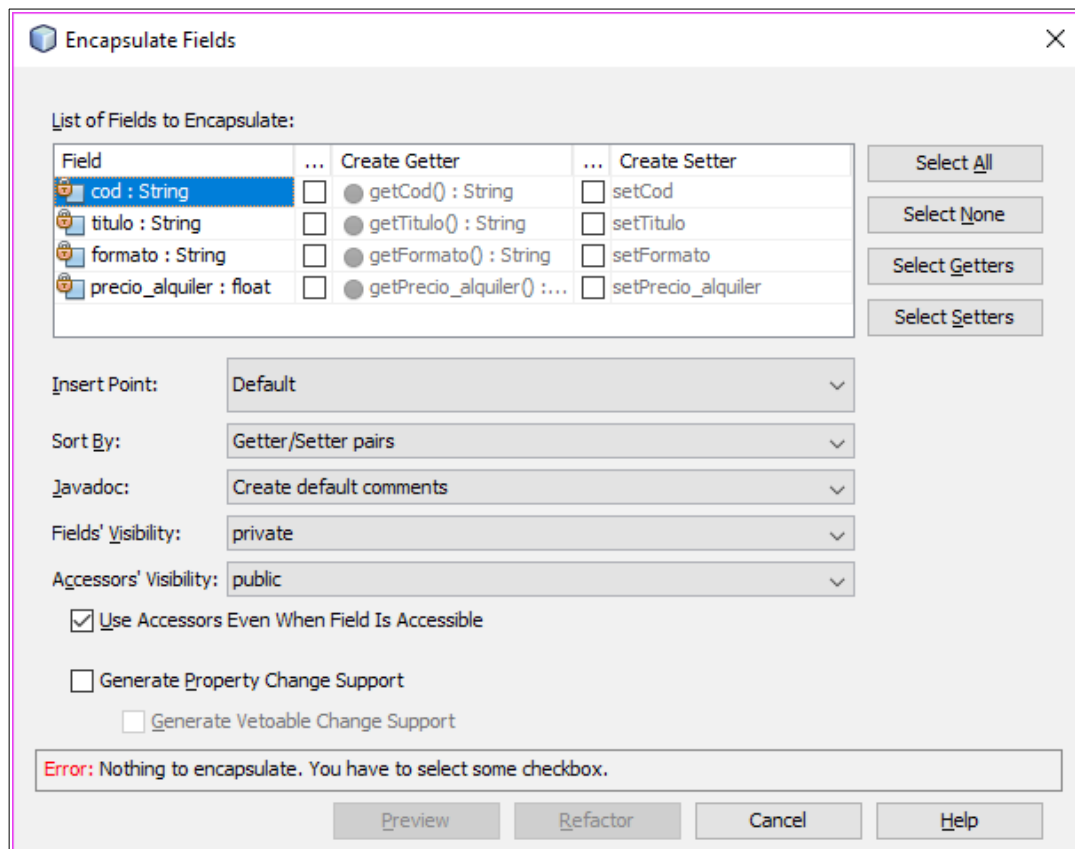
run:
Alquiler Art. 001, 1 día:2.5
Alquiler Art. 001, 2 días:4.5
Alquiler Art. 001, 1 semana:12.5
Alquiler Art. 002, 1 día:3.0
Alquiler Art. 002, 2 días:5.3999996
Alquiler Art. 002, 1 semana:15.0
BUILD SUCCESSFUL (total time: 0 seconds)

```

Otra forma de crear los métodos *getters* y *setters* es utilizando el *Refactor* de Netbeans. Para ello pinchamos en el menú *Refactor* y luego en *Encapsulate Fields*.



Seleccionamos los getters y setters que queramos crear y pinchamos en *Refactor*.



Automáticamente aparecerán los métodos en el código.

9.4 Ejemplo 4

En este ejemplo vamos a ver cómo crear un constructor de clase. Para ello vamos a basarnos en la clase *Articulo* creada anteriormente y le vamos a añadir el constructor. El código de *Articulo.java*:

```
12 public class Articulo {
13     // Atributos de la clase
14     private String cod;
15     private String titulo;
16     private String formato;
17     private float precio_alquiler;
18
19     // Constructor de la clase
20     public Articulo(String cod, String titulo, String formato, float precio_alquiler)
21     {
22         // Con 'this' estamos accediendo al objeto de la clase
23         this.cod = cod;
24         this.titulo = titulo;
25         this.formato = formato;
26         this.precio_alquiler = precio_alquiler;
27     }
28     // Métodos de la clase
29     public float precio1() {
30         return (getPrecio_alquiler());
31     }
```

Ahora en *Ejemplos.java* utilizamos el constructor (al instanciar el objeto con `new`). Esto permite instanciar el objeto y definir atributos en una sola instrucción.

```
12 public class Ejemplos {
13
14     public static void main(String[] args) {
15         // Creamos dos articulos
16         Articulo articulo1 = new Articulo("001", "Titulo1", "DVD", 2.5f);
17         Articulo articulo2 = new Articulo("002", "Titulo2", "DVD", 3f);
18
19         // Utilizamos sus métodos
20         System.out.println("Alquiler Art. " + articulo1.getCod() + ", 1 dia:" + articulo1.precio1());
21         System.out.println("Alquiler Art. " + articulo1.getCod() + ", 2 dias:" + articulo1.precio2());
22         System.out.println("Alquiler Art. " + articulo1.getCod() + ", 1 semana:" + articulo1.precio_semana());
23         System.out.println("Alquiler Art. " + articulo2.getCod() + ", 1 dia:" + articulo2.precio1());
24         System.out.println("Alquiler Art. " + articulo2.getCod() + ", 2 dias:" + articulo2.precio2());
25         System.out.println("Alquiler Art. " + articulo2.getCod() + ", 1 semana:" + articulo2.precio_semana());
26     }
27
28 }
```

Y la salida:

```
run:
Alquiler Art. 001, 1 dia:2.5
Alquiler Art. 001, 2 dias:4.5
Alquiler Art. 001, 1 semana:12.5
Alquiler Art. 002, 1 dia:3.0
Alquiler Art. 002, 2 dias:5.3999996
Alquiler Art. 002, 1 semana:15.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

9.5 Ejemplo 5

Para ilustrar el uso de los cualificadores *final* y *static* dentro de nuestra clase *Articulo*, vamos a definir tres atributos principales:

- La constante IVA.
- Un atributo compartido que contabilizará el número de instancias que se hayan definido hasta entonces.
- Un atributo constante String, distinto para cada objeto, que permita identificarlos.

Definiremos IVA como *static* y *final*, podremos acceder a su valor mediante la expresión *Articulo.IVA*, sin necesidad de haber creado ningún objeto de la clase *Articulo*:

```
public static final double IVA = 0.16;
```

El atributo privado con el número de instancia, en realidad, se trata de una variable global accesible para todos los objetos de tipo *Articulo*:

```
private static int numero = 0;
```

Y por último, el identificador constante de cada objeto se indica con un especificador *final* en su descripción.

```
final String identificador;
```

Se deja como ejercicio propuesto hacer estos cambios y probarlo.

9.6 Ejemplo 6

En este ejemplo vamos a crear un array (vector) que contenga diez objetos de tipo *Articulo*. Los instanciaremos todos con nombres genéricos, consecutivos y precios aleatorios.

Primero creamos un vector de Artículos llamado *misArticulos* (es un array de referencias a objetos *Articulo*). Luego instanciaremos los 10 objetos, guardando cada uno de los objetos referenciados.

La clase *Articulo* no se vería modificada.

El código de Ejemplos quedaría así:

```

12 public class Ejemplos {
13
14     public static void main(String[] args) {
15
16         // Definimos el vector de Articulos de tamaño 10
17         Articulo[] misArticulos = new Articulo[10];
18         int cont;
19
20         // Para cada elemento del vector creamos el objeto Articulo
21         for (cont = 0; cont < misArticulos.length; cont++)
22             misArticulos[cont]=new Articulo("00"+cont,"Articulo "+cont,"DVD",(float)Math.random()*10);
23
24
25         // Recorremos el vector
26         for (cont = 0; cont < misArticulos.length; cont++)
27         {
28             System.out.println("Codigo Art. "+misArticulos[cont].getCod());
29             System.out.println("Alquiler 1 dia :"+misArticulos[cont].precio1());
30             System.out.println("Alquiler 2 dias :"+misArticulos[cont].precio2());
31             System.out.println("Alquiler 1 semana :"+misArticulos[cont].precio_semana());
32         }
33     }
34 }
35

```

La salida sería:

```

run:
Codigo Art. 000
Alquiler 1 dia :9.574945
Alquiler 2 dias :17.234901
Alquiler 1 semana :47.874725
Codigo Art. 001
Alquiler 1 dia :2.8240945
Alquiler 2 dias :5.08337
Alquiler 1 semana :14.120473
Codigo Art. 002
Alquiler 1 dia :6.450059
Alquiler 2 dias :11.6101055
Alquiler 1 semana :32.250294
Codigo Art. 003
Alquiler 1 dia :2.0261488
Alquiler 2 dias :3.6470678
Alquiler 1 semana :10.130744
Codigo Art. 004
Alquiler 1 dia :1.0488434
Alquiler 2 dias :1.887918
Alquiler 1 semana :5.244217
Codigo Art. 005
Alquiler 1 dia :9.384731
Alquiler 2 dias :16.892515
Alquiler 1 semana :46.923656
Codigo Art. 006
Alquiler 1 dia :8.714678
Alquiler 2 dias :15.6864195
Alquiler 1 semana :43.573387
Codigo Art. 007
Alquiler 1 dia :4.498719
Alquiler 2 dias :8.097694
Alquiler 1 semana :22.493595
Codigo Art. 008
Alquiler 1 dia :5.8765197
Alquiler 2 dias :10.577735
Alquiler 1 semana :29.382599
Codigo Art. 009
Alquiler 1 dia :1.7480422
Alquiler 2 dias :3.146476
Alquiler 1 semana :8.7402115
BUILD SUCCESSFUL (total time: 0 seconds)

```

10. AGRADECIMIENTOS

Apuntes actualizados y adaptados al CEEDCV a partir de la siguiente documentación:

[1] Apuntes Programación de José Antonio Díaz-Alejo. IES Camp de Morvedre.

Licencia



Reconocimiento - NoComercial - CompartirIgual (by-nc-sa): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.