

## Unitat 8.

### Programació Orientada a Objectes I

Autors: Carlos Cacho y Raquel Torres

Revisat per: Lionel Tarazon - [lionel.tarazon@ceedcv.es](mailto:lionel.tarazon@ceedcv.es)

Fco. Javier Valero – [franciscojavier.valero@ceedcv.es](mailto:franciscojavier.valero@ceedcv.es)

José Manuel Martí - [josemanuel.marti@ceedcv.es](mailto:josemanuel.marti@ceedcv.es)

Jose Cantó Alonso – [j.cantoalonso@edu.gva.es](mailto:j.cantoalonso@edu.gva.es)

2023/2024

## Llicència



[CC BY-NC-SA 3.0 ES](https://creativecommons.org/licenses/by-nc-sa/3.0/es/) **Reconeixement – No Comercial – Compartir Igual (by- nc-sa)** No es permet un ús comercial de l'obra original ni de les possibles obres derivades, la distribució de les quals s'ha de fer amb una llicència igual a la que regula l'obra original. NOTA: Aquesta és una obra derivada de l'obra original realitzada per Carlos Cacho i Raquel Torres.

## Nomenclatura

Al llarg d'aquest tema s'utilitzaran diferents símbols per a distingir elements importants dins del contingut. Aquests símbols són:



Important



Atenció



Interessant

## Contingut

<b>1. Introducció .....</b>	<b>4</b>
<b>2. Fonaments d'una classe .....</b>	<b>5</b>
<b>3. Objectes .....</b>	<b>7</b>
3.1. Instanciació d'un objecte (l'operador new) .....	7
3.2. Assignació de variables de referència a objectes .....	7
3.3. Recol·lector de fem .....	7
<b>4. Tipus d'accés als membres d'una classe (Visibilitat) .....</b>	<b>8</b>
<b>5. Mètodes .....</b>	<b>9</b>
<b>6. Constructors .....</b>	<b>10</b>
<b>7. Constants de classe i objecte .....</b>	<b>11</b>
<b>8. Array d'objectes .....</b>	<b>12</b>
<b>9. Exemples .....</b>	<b>13</b>
9.1. Exemple 1 .....	13
9.2. Exemple 2 .....	14
9.3. Exemple 3 .....	16
9.4. Exemple 4 .....	19
9.5. Exemple 5 .....	20
9.6. Exemple 6 .....	20
<b>10. Agraïments.....</b>	<b>22</b>

## 1. Introducció

La Programació Orientada a Objectes (POO) fa que els problemes siguin més senzills, en permetre dividir el problema. Aquesta divisió es fa en objectes, de manera que cada objecte funcione de forma totalment independent.



Un **objecte** és un element del programa que posseeix les seues pròpies dades i el seu propi funcionament.



Una **classe** descriu un grup d'objectes que contenen una informació similar (atributs) i un comportament comú (mètodes).



Abans de poder utilitzar un objecte, s'ha de definir la seua classe. La classe és la definició d'una mena d'objecte.

En definir una classe el que es fa és indicar com funciona un determinat tipus d'objectes. Després, a partir de la classe, podem crear objectes d'aqueixa classe.

Les propietats de la POO són la següents:

- **Encapsulament.** Una classe es compon tant de variables (atributs) com de funcions i procediments (mètodes). De fet no es poden definir variables (ni funcions) fora d'una classe (és a dir no hi ha variables globals).
- **Abstracció.** Cal crear versions simplificades dels objectes i classes del món real. També s'estableixen zones ocultes en definir la classe (zona privada) que només són utilitzades per aquesta classe i per alguna classe relacionada. Hi ha zones públiques (anomenada també interfície de la classe) que poden ser utilitzades per qualsevol part del codi.
- **Polimorfisme.** Cada mètode d'una classe pot tindre diverses definicions diferents. En el cas del joc parxís: partida.començar(4) comença una partida per a quatre jugadors, partida.començar(roig, blau) comença una partida de dos jugadors per als colors roig i blau; aquestes són dues formes diferents d'emprar el mètode començar, que és polimòrfic.
- **Herència.** Una classe es pot definir a partir d'una altra ja existent. D'aquesta forma, la nova classe pot heretar propietats (atributs i mètodes) de l'altra.

## 2. Fonaments d'una classe

Una classe descriu un grup d'objectes que contenen una informació similar (atributs) i un comportament comú (mètodes).

Les definicions comuns (nom de la classe, els noms dels atributs, i els mètodes) s'emmagatzemen una única vegada en cada classe, independentment de quants objectes d'aqueixa classe siguin presents en el sistema.

**Una classe és com un motle.** A partir d'ella es poden crear objectes.

És a dir abans de poder utilitzar un objecte s'ha de definir la classe a la qual pertany, aqueixa definició inclou:

- **Atributs.** Les variables membre de la classe. Poden ser *public* (accessibles des d'una altra classe), *private* (només accessibles per codi de la seua pròpia classe) o *protected* (accessibles per les subclasses).
- **Mètodes.** Les funcions membre de la classe. Són les accions o operacions que pot realitzar la classe. Igual que els atributs poden ser *public*, *private* o *protected*.

En notació UML <sup>1</sup>l'estructura d'una classe es defineix així:

Nom de la classe
Atributs
Mètodes

I la sintaxi d'una classe a Java és la següent:

```
[accés] class nomDeClase {
[accés] [static] tipus atribut1;
[accés] [static] tipus atribut2;
//...més atributs...

[accés] [static] tipus mètode1(llistaDeParàmetres) {
//...codi del mètode...
}
[accés] [static] tipus mètode2(llistaDeParàmetres) {
//...codi del mètode...
}
//...més mètodes...
}
```

*Vegem un exemple:*

La classe Persona conté dos atributs (variables) per a emmagatzemar dades sobre una persona (nom i edat) a més de diversos mètodes (funcions) que fan coses amb aquestes dades.

```
public class Persona {
String nom; int edat;

// Estableix el nom de la persona
void setNom(String n) {
nom = n;
}
```

<sup>1</sup> Unified Modelling Language

```
// Estableix l'edat de la persona void setEdat(int e) {
edat = e;
}

// Retorna el nom de la persona
String getNom() {
return nom;
}

// Retorna l'edat de la persona int getEdat() {
return edat;
}

// Mostra el seu nom per pantalla void imprimeNom() {
System.out.println(nom);
}

// Retorna true si és major d'edat, false en cas contrari boolean
esMajorEdat() {
return (edat >= 18)
}
}
```

Cal tindre en compte que **la classe Persona ens servirà per a crear tants objectes Persona com necessitem, cadascun amb el seu nom i edat**. Els mètodes ens permetran manipular les dades de cada objecte. Això s'explica en més detall en el següent apartat.

També és important entendre que cada classe es crea en un arxiu Java diferent (amb el mateix nom de la classe), i s'utilitzen fora de la classe.

Per exemple, podríem tindre un arxiu Persona.java (amb la classe Persona de l'exemple anterior) a més d'un arxiu Programa.java (que només tindrà la funció principal public static void main a la qual estem acostumats). **Des de la funció main del Programa podrem crear objectes de tipus Persona a més de qualsevol altre codi que necessitem**. Això es veurà en exemples posteriors.

L'us atributs i mètodes d'una classe es diuen *membres d'una classe*.

Els atributs (variables) d'una classe es diuen *variables d'instància* perquè cada instància de la classe (és a dir, cada objecte de la classe), conté les seues propies variabes atribut. Per tant, les dades de cada objecte són individuals i independent dels altres objectes.

La paraula opcional *static* serveix per a fer que el mètode o l'atribut a la qual precedeix es pugui utilitzar de manera genèrica (més endavant es parlarà de classes genèriques), els atributs i mètodes així definits es diuen *atributs de classe i mètodes de classe* respectivament.

### 3. Objectes

#### 3.1. Instanciació d'un objecte (l'operador new)

Quan creem una classe definim un nou tipus de dades que pot utilitzar-se per a instanciar (crear) objectes d'eixa classe. La instanciació es fa de la següent manera:

- En primer lloc, cal declarar una variable de la mena de la classe.
- En segon lloc, es necessitarà una **còpia física de l'objecte i assignar-la a aqueixa variable**. Això es fa utilitzant l'operador **new** que assigna dinàmicament (és a dir, durant temps d'execució) memòria a un objecte i retorna una referència. Aquesta referència s'emmagatzema en una variable.

Per exemple, suposant que ja hem creat la classe Poal:

```
Poal p1;      // Crea una variable referència anomenada p1 de tipus Poal
new Poal();   // Crea un objecte Poal i l'assigna a p1
```

També pot fer-se tot en una sola línia de codi:

```
Poal p1 = new Poal();
```

#### 3.2. Assignació de variables de referència a objectes

Les variables de referència a objectes permeten accedir a l'objecte (són una referència a l'objecte, no l'objecte). Per això, actuen de manera diferent al que caldria esperar quan té lloc una assignació. Per exemple, què fa el següent fragment de codi?

```
Poal p1 = new Poal(); Poal p2 = p1;
```

Podríem pensar que a p2 se li assigna una còpia de l'objecte p1, però no és així. El que succeeix és que la referència p1 es copia en p2, per la qual cosa p2 permetrà accedir al mateix objecte referenciat per p1. Per tant qualsevol canvi que es faci a l'objecte referenciat a través de p2 afectarà l'objecte al qual referencia p1.

#### 3.3. Recol·lector de fem

A Java hi ha un recol·lector de fem (*garbage collector*) que s'encarrega de gestionar els objectes que es deixen d'usar i d'eliminar-los de memòria. Aquest procés és automàtic i imprevisible i treballa en un fil (*thread*) de baixa prioritat. En general aqueix procés de recol·lecció de fem, treballa quan detecta que un objecte fa massa temps que no s'utilitza en un programa. Aquesta eliminació depèn de la màquina virtual de Java, en quasi totes la recol·lecció es realitza periòdicament en un determinat lapse de temps.

## 4. Tipus d'accés als membres d'una classe (Visibilitat)

Els membres d'una classe (atributs i mètodes, és a dir, les seues variables i funcions) poden definir-se com públics, privats o protegits. És important entendre la diferència:

- **public**: Es pot utilitzar des de qualsevol classe.
- **private**: Només pot utilitzar-ho la pròpia classe.
- **protected**: Pot utilitzar-ho la pròpia classe i també les subclasses heretades (ara com ara no l'utilitzarem, l'herència de classes es veurà en la següent unitat).

Se'n diu **interfície** als membres d'una classe (atributs i mètodes) que són **public**, perquè són els que permeten interactuar amb la classe des de fora d'ella.

Els principis de la programació orientada a objectes diu que per a mantindre l'encapsulació en els objectes hem d'aplicar l'especificador **public** a les funcions membre que formen la interfície pública i denegar l'accés a les dades membre usats per aqueixes funcions mitjançant els especificats **private**.



En un **paquet**, que és un agrupament lògic de classes en un mateix directori, els atributs i els mètodes d'aquestes classes són **públics** per defecte per a la resta de classes existents en el mateix paquet, i **privats** per a qualsevol classe que es trobe fora (llevat que s'especifique el contrari).

Quan un paquet no està definit, es diu que la classe pertany al paquet per defecte, per tant s'aplicarà el qualificador **public** a la resta de les classes els fitxers de les quals es troben en el mateix directori (veure [exemple 3](#)).



## 5. Mètodes

Es diu mètode a una funció d'una classe. La seua sintaxi general és la següent:

```
tipus nom_del_mètode(llista de paràmetres) {  
  // cos del mètode  
}
```

On:

- el **tipus** especifica el tipus de dades que retorna el mètode. Pot tractar-se de qualsevol tipus vàlid, incloent els tipus de classes que crea el programador. Si el mètode no retorna un valor, el tipus retornat serà *void*.
- el **nom** del mètode l'especifica *nom\_de\_el\_mètode*, que pot ser qualsevol identificador vàlid diferent a aquells ja usats per altres elements del programa.
- la llista **de paràmetres** és una seqüència de parells de tipus i identificador separats per comes. Els paràmetres són variables que reben el valor dels arguments que es passa al mètode quan es diu. Si el mètode no té paràmetres, la llista de paràmetres estarà buida.

Com en qualsevol altra funció, els mètodes retornen un valor amb la sentència **return** llevat que el tipus retornat es definisca com *void*:

```
return valor;
```

Encara que seria perfectament vàlid crear una classe que únicament continga dades, això rarament ocorre. La majoria de les vegades és convenient crear mètodes per a accedir als atributs de la classe.

A més de definir mètodes que proporcionen accés a les dades (s'oculta o abstrau l'estructura interna de les dades) poden definir-se mètodes per a ser utilitzats internament per la classe.

## 6. Constructors

Pot resultar una tasca bastant pesada inicialitzar totes les variables d'una classe cada vegada que es crea una instància. Fins i tot quan s'afigen funcions adequades, és més senzill i requerisc realitzar totes les inicialitzacions quan l'objecte es crea per primera vegada. Com el procés d'inicialització és tan comú, Java permet que els objectes s'inicialitzen quan es creen. Aquesta inicialització automàtica es duu a terme mitjançant un constructor.



Un **constructor** és un mètode especial que no retorna mai un valor, sempre retorna una referència a una instància de la classe i és anomenat automàticament en crear un objecte d'una classe, és a dir en usar la instrucció *new*.

Un *constructor* no és més que un mètode que té el mateix nom que la classe. En general, quan es crea una instància d'una classe, no sempre es desitja passar els paràmetres d'inicialització en construir-la, per això existeix un tipus especial de constructors, que són els anomenats **constructors per defecte**.

Aquests constructors **no porten paràmetres associats**, i **inicialitzen les dades** assignant-los valors per defecte.

Quan no es diu a un constructor de forma explícita, Java crea un constructor per defecte i el crida quan es crea un objecte. Aquest constructor per defecte assigna a les variables d'instància un valor inicial igual a zero a **les variables numèriques** i *null* a totes les referències a objectes.

Una vegada es definisca un constructor per a la classe es deixa d'utilitzar el constructor per defecte.



Existeix la referència *this*, que apunta a la instància que anomena al mètode, i sempre està accessible, és utilitzat per les funcions membre per a accedir a les dades de l'objecte.

(Veure [exemple 4](#)).

## 7. Constants de classe i objecte

Els membres constants es defineixen a Java a través de la paraula reservada *final*, mentre que els membres de classe es defineixen mitjançant la paraula reservada *static*. D'aquesta manera, si considerem que els membres són atributs, tenim quatre possibles combinacions a Java per a indicar si un atribut és constant:

- Atributs ***static***: prenen valors comuns a tots els objectes existents i potencialment variables. Poden utilitzar-se encara que no existisca cap objecte instanciat.
- Atributs ***final***: són valors constants, però potencialment diferents en cadascuna de les instàncies. El seu valor s'inicialitza en la fase de construcció de l'objecte i no poden ser modificats durant el temps de vida d'aquest.
- Atributs ***static final***: combinen les característiques de *static* i *final*.
- Resta d'atributs (sense *static* ni *final*): atributs variables i diferents per a cada objecte

⚡ Els atributs ***static*** de la classe han de ser **limitats**, ja que poden donar lloc a errors de molt difícil depuració, a més d'anar contra el concepte de la programació orientada a objectes.

Les constants d'objecte es defineixen mitjançant la paraula reservada *final*. Es tracta d'atributs que prenen el valor en el constructor de l'objecte, un valor que no pot ser modificat en la resta del programa. D'aquesta manera, a cada objecte correspon un atribut d'aqueix tipus, però invariable per a ell. Aquest tipus d'atributs serveix per a identificar cada objecte de manera única (veure [exemple 5](#)).

## 8. Array d'objectes

A Java hi ha dues sintaxis possibles per a definir un **array d'objectes**:

```
NomDeLaClasse [] Objectes;  
NomDeLaClasse Objectes[];
```

D'aquesta manera es crea la referència l'array d'Objectes.


Per a crear l'instància de l'objecte array, hem d'especificar la grandària desitjada amb la sintaxi:

```
Objectes = new NomDeLaClasse[n]; // n és el número de referències a Objectes
```

Podem resumir aquestes dues sentències en una sola:

```
NomDeLaClasse [] Objectes = new NomDeLaClasse[n];  
NomDeLaClasse Objectes[] = new NomDeLaClasse[n];
```

Cal tindre en compte que, quan es crea l'array, el compilador genera una referència per a cadascun dels elements que l'integren, encara que no existisquen encara les instàncies de cap d'ells.

 Per a crear els objectes cal **cridar** explícitament al **constructor** per cada **element creat**:

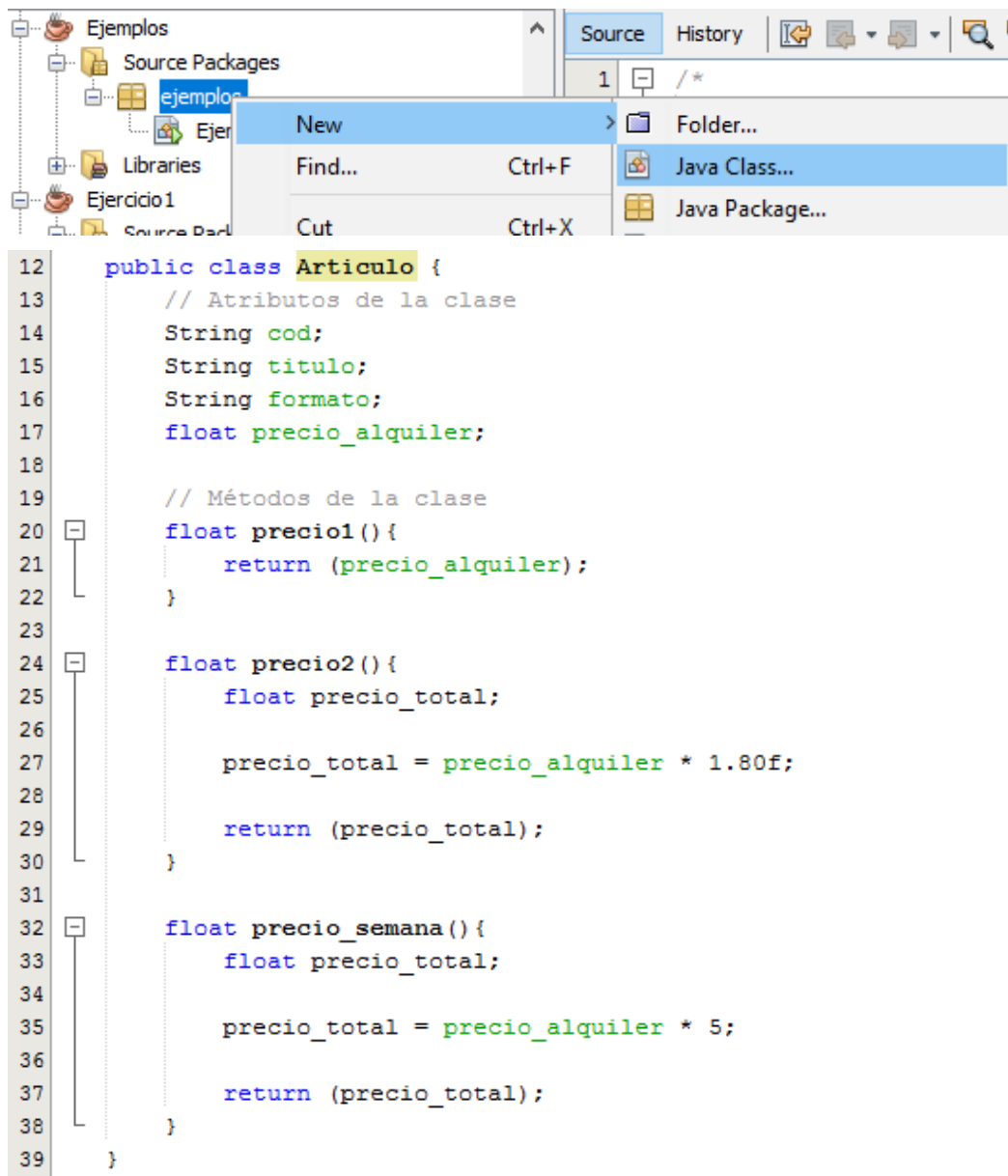
```
Objecte[i] = new NomDeLaClasse();
```

Veure [exemple 6](#).

## 9. Exemples

### 9.1. Exemple 1

En aquest exemple implementarem la classe *Articulos*. Aquesta classe representa cada objecte amb els següents atributs: *codigo\_articulo*, *titulo*, *formato* i *precio\_alquiler*. També defineix tres mètodes que permeten calcular, respectivament, el preu de lloguer d'un dia, de dos dies i una setmana. El primer que necessitem, i això ho farem per cada classe que necessitem en tots els exemples, serà crear-nos una nova classe a Java. Per a això punxarem amb el botó dret sobre el paquet on anem a tindre les classes i després en *NEW > Java Class*. Crearem una classe anomenada *Articulo*.



El mètode *precio1* retorna el valor del preu de lloguer de l'article. El mètode *precio2* calcula el preu de lloguer de dos dies fent un descompte del 20% (per això es multiplica per 1,8). Per últim el mètode *precio\_semana* calcula el preu d'una setmana multiplicant per 5 el preu de lloguer.

Crearem una classe nova anomenada *Ejemplos*, amb la funció `public static void main`, on anem a instanciar (crear) objectes de *Articulo* i utilitzar-los.

Amb l'operador *new* crearem una instància de la classe *Articulo*. Com de moment la classe no té constructors s'invocarà al constructor per defecte de la classe.

**RECORDA:** Quan instanciem una classe estem creant un objecte d'aquesta classe.

```

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

```

L'eixida és:

```

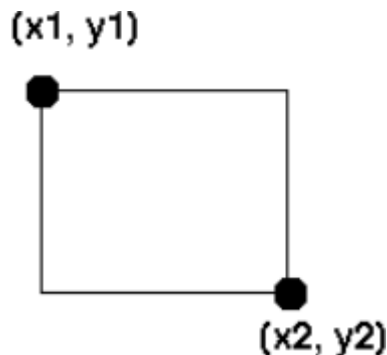
run:
Alquiler Art. 001, 1 dia:2.5
Alquiler Art. 001, 2 dias:4.5
Alquiler Art. 001, 1 semana:12.5
Alquiler Art. 002, 1 dia:3.0
Alquiler Art. 002, 2 dias:5.3999996
Alquiler Art. 002, 1 semana:15.0
BUILD SUCCESSFUL (total time: 0 seconds)

```

## 9.2. Exemple 2

En aquest exemple anem a implementar la classe *Cuadrado*, que representa quadrats mitjançant dues coordenades 2D, i defineix tres mètodes que permeten calcular, respectivament, la diagonal, el perímetre i l'àrea.

El criteri de representació pren les coordenades horitzontals (x) creixents d'esquerra a dreta, i les verticals (y) creixents de dalt a baix.



El codi de *Cuadrado.java* seria:

```

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 codi de *Ejemplos.java* seria:

```

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 }

```

I l'eixida:

```

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. Exemple 3

En aquest exemple aplicarem el principi de encapsulament fent *private* els atributs de la classe i *public* els mètodes. Per a això modificarem la classe *Articulo* de l'Exemple 1.

Però ara no podrem llegir ni modificar els atributs de la classe des de fora d'ella (perquè són *private*). Així que definirem mètodes que ens permeten fer-ho:

- Crearem mètodes *public* (un per atribut) que ens retorne el valor de cada atribut. A això se'n diu mètodes "**get**" o "**getters**" (de l'anglès agafar).
- De la mateixa manera, mètodes que ens permeten modificar el valor dels atributs. A això se'n diu mètodes "**set**" o "**setters**" (de l'anglès establir). En l'exemple es diu *modificaValores* i permet canviar tots els valors en una sola crida.

El codi de *Articulo.java* seria:



```

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 codi de Ejemplos.java amb el main seria:

```

12 public class Ejemplos {
13
14     public static void main(String[] args) {
15         // Creamos dos articulos
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 dia:" + articulo1.precio1());
25         System.out.println("Alquiler Art. " + articulo1.getCod() + ", 2 dias:" + articulo1.precio2());
26         System.out.println("Alquiler Art. " + articulo1.getCod() + ", 1 semana:" + articulo1.precio_semana());
27         System.out.println("Alquiler Art. " + articulo2.getCod() + ", 1 dia:" + articulo2.precio1());
28         System.out.println("Alquiler Art. " + articulo2.getCod() + ", 2 dias:" + articulo2.precio2());
29         System.out.println("Alquiler Art. " + articulo2.getCod() + ", 1 semana:" + articulo2.precio_semana());
30     }
31 }
32

```

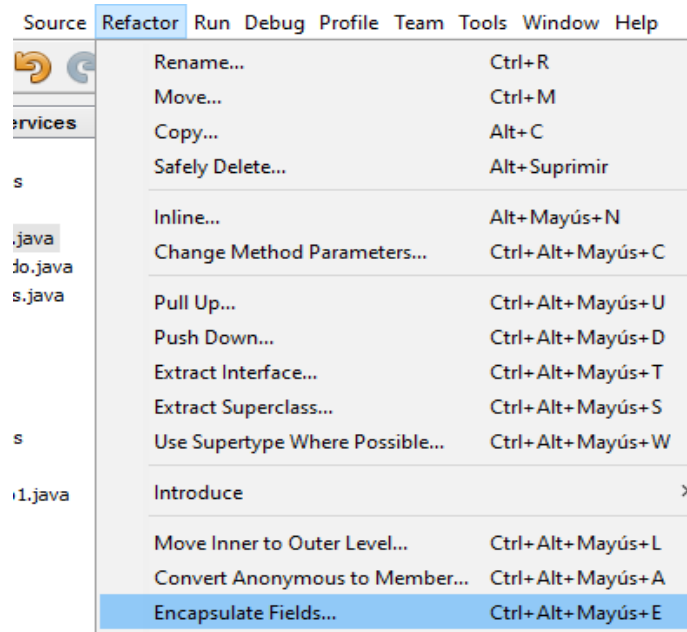
I l'eixida:

```

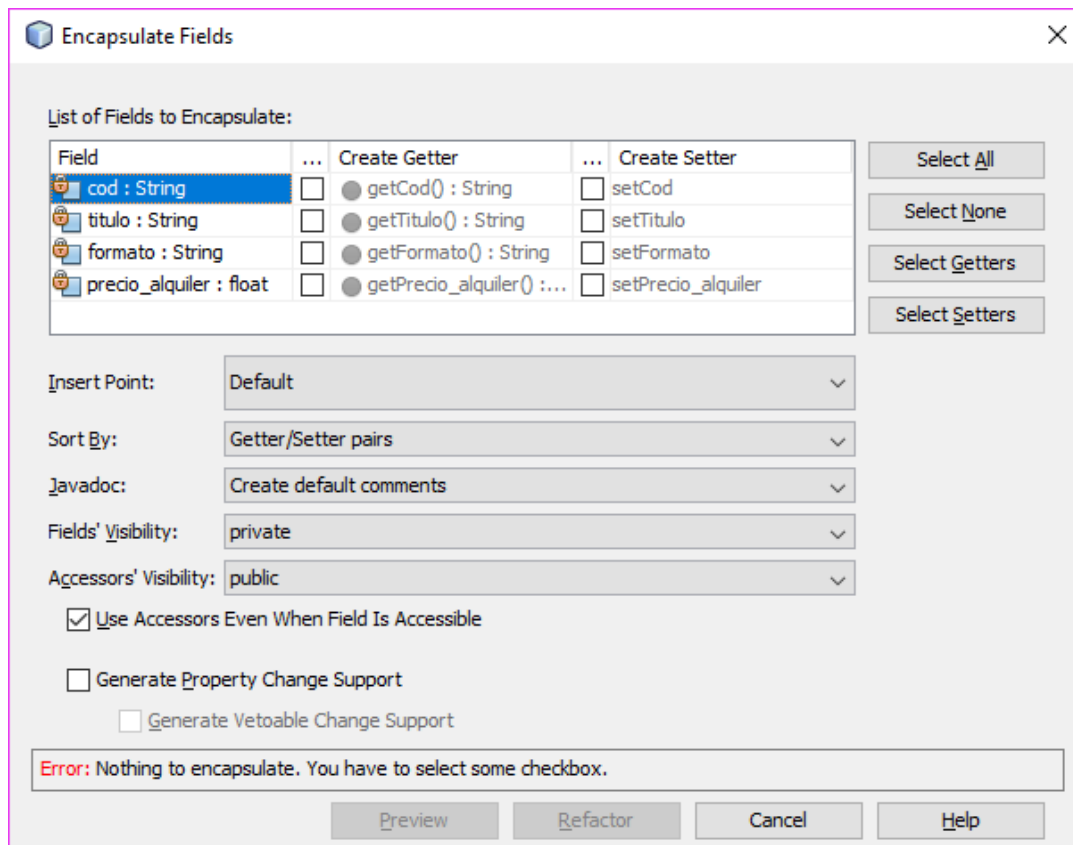
run:
Alquiler Art. 001, 1 dia:2.5
Alquiler Art. 001, 2 dias:4.5
Alquiler Art. 001, 1 semana:12.5
Alquiler Art. 002, 1 dia:3.0
Alquiler Art. 002, 2 dias:5.3999996
Alquiler Art. 002, 1 semana:15.0
BUILD SUCCESSFUL (total time: 0 seconds)

```

Una altra manera de crear els mètodes *getters* i *setters* és utilitzant el *refactor* de Netbeans. Per a això punxem en el menú *refactor* i després en *Encapsulate Fields*.



Seleccionem els getters i setters que vulguem crear i punxem en *refactor*.



Automàticament apareixeran els mètodes en el codi.

## 9.4. Example 4

En aquest exemple veurem com crear un constructor de classe. Per a això ens basarem en la classe

*Articulo* creada anteriorment i li afegirem el constructor. El codi 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     }

```

Ara en *Ejemplos.java* utilitzem el constructor (al instanciar l'objecte amb `new`). Això permet instanciar l'objecte i definir atributs en una sola instrucció.

```

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

```

I l'eixida:

```

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)

```

## 9.5. Exemple 5

Per a il·lustrar l'ús dels qualificadors *final* i *static* dins de la nostra classe *Articulo*, definirem tres atributs principals:

- La constant IVA.
- Un atribut compartit que comptabilitzarà el nombre d'instàncies que s'hagen definit fins llavors.
- Un atribut constant String, diferent per a cada objecte, que permeta identificar-los.

Definirem IVA com *static* i *final*, podem accedir al seu valor mitjançant l'expressió *Articulo.IVA*, sense necessitat d'haver creat cap objecte de la classe *Articulo* :

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

L'atribut privat amb el número d'instància, en realitat, es tracta d'una variable global accessible per a tots els objectes de tipus *Articulo*:

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

I finalment, l'identificador constant de cada objecte s'indica amb un especificador *final* en la seua descripció.

```
final String identificador;
```

Es deixa com a exercici proposat fer aquests canvis i provar-ho.

## 9.6. Exemple 6

En aquest exemple crearem un array (vector) que continga deu objectes de tipus *Articulo*. Els instanciem tots amb noms genèrics, consecutius i preus aleatoris.

Primer creem un vector de *Articulos* anomenat *misArticulos* (és un array de referències a objectes *Articulo*). Després instanciem els 10 objectes, guardant cadascun dels objectes referenciats.

La classe *Articulo* no es veuria modificada. El codi d'Ejemplos quedaria així:

```

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 }

```

L'eixida seria:

```

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. Agraïments

Anotacions actualitzades i adaptats al CEEDCV a partir de la següent documentació:

- [1] Anotacions Programació de José Antonio Díaz-Alejo. IES Camp de Morvedre.