

Programación orientada a
objetos con Java

Microlearning

Principios fundamentales de la POO: herencia y encapsulación

¡Has acabado de ver el video con los principios fundamentales de la POO: herencia y encapsulación! Estás avanzando muy bien, continúa así.

Durante el video los conceptos más importantes fueron:

- ▶ Herencia
- ▶ Herencia y constructores
- ▶ Encapsulación
- ▶ Niveles de acceso
- ▶ Getters y Setters

No olvides que al igual que los seres vivos, la herencia en la POO funciona de manera similar. Solo que en este caso las clases hijas heredan los atributos y los métodos de las clases padres.

Como lo vimos en el ejemplo del video, una nueva clase “Deportivo” fue creada en el mismo archivo donde se encontraba nuestra clase Coche. Y para poder heredar de la clase padre solo fue necesario utilizar la palabra “extends”.

```
Coche.java X Main.java
1  class Coche {
2      String motor;
3      int puertas;
4      int ruedas;
5
6      void arrancar() {
7          System.out.println("El coche arranca.");
8      }
9
10     void frenar() {
11         System.out.println("El coche frena.");
12     }
13
14     void acelera() {
15         System.out.println("El coche acelera.");
16     }
17
18 }
19
20 class Deportivo extends Coche {
21
22 }
23
24
25 }
```

A pesar de que la clase Deportivo no contenía ningún código dentro de ella, si se creaba una instancia de esta en el programa principal, los mismos atributos y métodos de la clase Coche (padre) podían utilizarse.

```

3 public class Main {
4
5     public static void main(String[] args) {
6
7         Deportivo coche_deportivo = new Deportivo();
8
9         coche_deportivo.acelerar();
10        coche_deportivo.arrancar();
11        coche_deportivo.frenar();
12    }
13 }

```

Console X
<terminated> Main (4) [Java Application] /Library/Java/JRE/jdk1.8.0_111/bin/java
El coche acelera.
El coche arranca.
El coche frena.

Esto sucedía así, ya que los métodos de la clase Coche eran heredados a la clase Deportivo.

Ahora, ¿qué pasa con los constructores que se heredan? ¿Recuerdas que el constructor de la clase padre que no tiene argumentos siempre se ejecutará? En el video vimos que si ambas clases padre e hija tenían un constructor...

```

Coche.java X Main.java
1  class Coche {
2
3     String motor;
4     int puertas;
5     int ruedas;
6
7     Coche() {
8         System.out.println("Inicializa clase padre coche.");
9     }
10
11    void arrancar() {
12        System.out.println("El coche arranca.");
13    }
14
15    void frenar() {
16        System.out.println("El coche frena.");
17    }
18
19    void acelerar() {
20        System.out.println("El coche acelera.");
21    }
22 }
23
24 class Deportivo extends Coche {
25
26     Deportivo() {
27         System.out.println("Inicializa clase hija deportivo.");
28     }
29 }
30
31
32

```

Al instanciar un objeto, tanto el constructor de la clase hija y de la clase padre (sin argumentos) se ejecutarán.

```

1  public class Main {
2
3      public static void main(String[] args) {
4          // TODO Auto-generated method stub
5
6          Deportivo coche_deportivo = new Deportivo();
7
8      }
9
10 }
11
12 }

Console X
<terminated> Main (4) [Java Application] /Library/Java/JavaVirtualMachines/jdk-11.0.12.jdk/Contents/Home/bin/java (M
Inicializa clase padre coche.
Inicializa clase hija deportivo.
    
```

Si quisieramos ejecutar un constructor de la clase padre que tuviese argumentos, desde la clase hija debemos utilizar la palabra “super” dentro del constructor de la clase hija.

```

1  class Coche {
2
3     String motor;
4     int puertas;
5     int ruedas;
6
7     Coche(String motor, int puertas, int ruedas) {
8         this.motor = motor;
9         this.puertas = puertas;
10        this.ruedas = ruedas;
11        System.out.println("Inicializar clase padre con argumentos: " +
12                           this.motor + ", " +
13                           this.puertas + ", " +
14                           this.ruedas);
15    }
16
17
18    void arrancar() {
19        System.out.println("El coche arranca.");
20    }
21
22    void frenar() {
23        System.out.println("El coche frena.");
24    }
25
26    void acelerar() {
27        System.out.println("El coche acelera.");
28    }
29 }
30
31 class Deportivo extends Coche {
32
33    Deportivo() {
34        super("6 cilindros", 2, 4);
35    }
36
37    System.out.println("Inicializar clase hija.");
38 }
39
40
    
```

Manda llamar constructor con argumentos de la clase padre.

```

1  public class Main {
2
3      public static void main(String[] args) {
4          // TODO Auto-generated method stub
5
6          Deportivo coche_deportivo = new Deportivo();
7
8      }
9
10 }
11
12 }

Console X
<terminated> Main (4) [Java Application] /Library/Java/JavaVirtualMachines/jdk-11.0.12.jdk/Contents/Home/bin/java (M
Inicializar clase padre con argumentos: 6 cilindros, 2, 4
Inicializar clase hija.
    
```

Para la encapsulación recordemos que este es un principio fundamental de la POO que trata sobre la capacidad de poder esconder algunos detalles del código implementado.

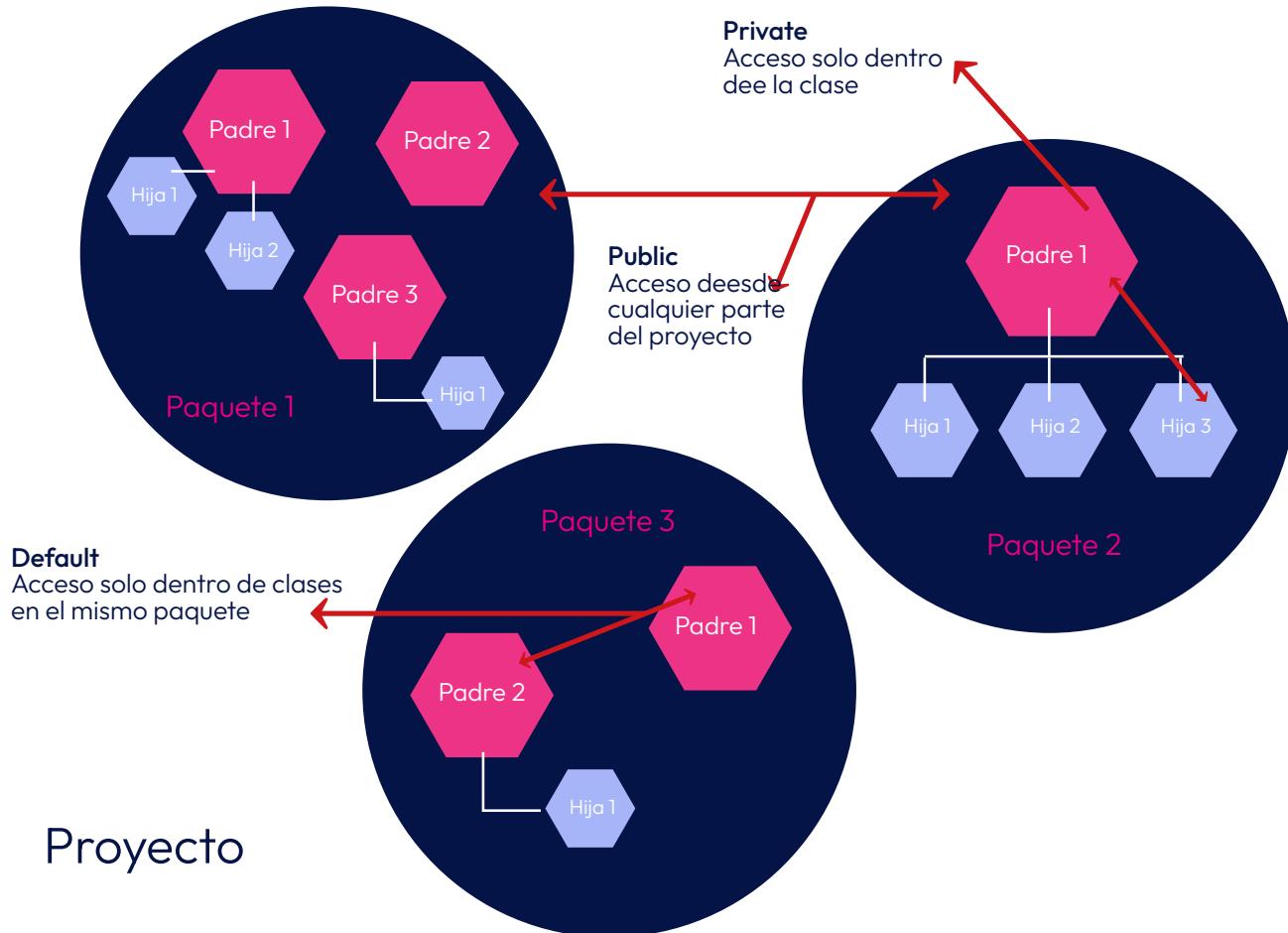
Algunas de las razones por las que quisiéramos usar la encapsulación son:

Utilidad. Un miembro de la clase (métodos y atributos) solo es útil para esa clase y no tiene sentido compartirlo con otra.

Seguridad. Detalles como contraseñas, datos personales o funcionalidades críticas contenidas dentro de la clase no deberían compartirse con otras clases ya que el código se puede volver vulnerable a amenazas o defectos críticos.

Abstracción. Más adelante hablaremos del tema, pero aplicar este principio fomenta que para los usuarios finales de los objetos solo deberían ser expuestos los detalles necesarios.

Para poder realizar el encapsulamiento del código, Java provee cuatro niveles de acceso, sin embargo, en este curso sólo aplicaremos tres. Un nivel de acceso permite acceder a funcionalidades del código solo desde ciertas partes del proyecto.



The screenshot shows the Eclipse IDE interface. On the left, the 'Package Explorer' view displays a project named 'POO_Java' with a 'src' folder containing packages 'paquete_1' and 'paquete_2'. Within 'paquete_1', there are files 'Coche.java', 'Main.java', and 'clase_padre_1.java'. Within 'paquete_2', there are files 'clase_padre_1.java', 'clase_padre_2.java', and 'clase_hija_1.java'. The 'clase_padre_1.java' file contains the following code:

```

1 package paquete_1;
2
3 class clase_padre_1 {
4     // Código clase_padre_1
5 }
6
7 class clase_hija_1 extends clase_padre_1 {
8     // Código clase_hija_1
9 }
10
11 class clase_hija_2 extends clase_padre_1 {
12     // Código clase_hija_2
13 }
14
15 class clase_hija_3 extends clase_padre_1 {
16     // Código clase_hija_3
17 }
18

```

A legend on the right side identifies the color coding: red for 'Proyecto', blue for 'Paquete', green for 'Clase Padre', and purple for 'Clase Hija'.

A continuación recordaremos los niveles de acceso más comunes en java:

- “**Private**”. Es el nivel de acceso con menor accesibilidad, un miembro privado solo puede accederse dentro de la clase donde fue creado. Para usar este nivel la palabra private debe de agregarse al inicio de los atributos o métodos que se desean volver privados.

The screenshot shows the Eclipse IDE interface with a 'Main.java' file open. The code defines a 'Coche' class with three private attributes: 'motor', 'puertas', and 'ruedas'. It also contains three methods: 'arrancar()', 'frenar()', and 'acelerar()'. A red box highlights the private access modifier 'private' used for the attributes.

```

1
2 class Coche {
3
4     private String motor;
5     private int puertas;
6     private int ruedas;
7
8     void arrancar() {
9         System.out.println("El coche arranca.");
10    }
11
12    void frenar() {
13        System.out.println("El coche frena.");
14    }
15
16    void acelerar() {
17        System.out.println("El coche acelera.");
18    }
19
20 }
21
22 class Deportivo extends Coche {
23
24 }
25
26
27

```

- “**Default**”. Un atributo o método con este nivel puede accederse solo dentro del paquete donde se creó la clase. Para asignar el modificador default no se debe agregar ninguna palabra antes del método o atributo. Este atributo está implícito por defecto.

The screenshot shows the Eclipse IDE interface with a 'Main.java' file open. The code defines a 'Coche' class with three attributes: 'motor', 'puertas', and 'ruedas'. It contains three methods: 'arrancar()', 'frenar()', and 'acelerar()'. A red box highlights the default access modifier (lack of explicit modifier) used for the attributes. In addition, a method 'acceder_atributos()' is shown, which changes the value of the 'motor' attribute.

```

1
2 class Coche {
3
4     String motor;
5     int puertas;
6     int ruedas;
7
8     void arrancar() {
9         System.out.println("El coche arranca.");
10    }
11
12    void frenar() {
13        System.out.println("El coche frena.");
14    }
15
16    void acelerar() {
17        System.out.println("El coche acelera.");
18    }
19
20 }
21
22 class Deportivo extends Coche {
23
24     void acceder_atributos() {
25         motor = "5 cilindros";
26     }
27
28 }
29

```

- “Public”, el último nivel de acceso permite que un miembro de la clase pueda accederse desde cualquier lugar del proyecto. Para especificar este nivel de acceso la palabra public debe de ser especificada en los miembros de la clase.

```

1 package Paquete1;
2
3 public class Coche {
4
5     public String motor;
6     public int puertas;
7     public int ruedas;
8
9
10    public void arrancar() {
11        System.out.println("El coche arranca.");
12    }
13
14    public void frenar() {
15        System.out.println("El coche frena.");
16    }
17
18    public void acelerar() {
19        System.out.println("El coche acelera.");
20    }
21 }
22
23 class Deportivo extends Coche {
24
25    void acceder_atributos() {
26        motor = "5 cilindros";
27    }
28 }
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45

```

Por último recordemos que los setters y los getters son un tema relacionado con la encapsulación y son llamados en conjunto como mutadores.

En general todos los atributos de una clase deberían ser declarados como privados para evitar ser modificados desde otra parte del código, ya sea por error o intencionalmente.

Si los atributos de una clase son privados, la forma en cómo se accede o se modifica su valor es usando setters para modificar el valor del atributo y getters para obtener el valor del atributo.

```

1 package Paquete1;
2
3 public class Coche {
4
5     private String motor;
6     private int puertas;
7     private int ruedas;
8
9     public String getMotor() {
10         return motor;
11     }
12
13     public void setMotor(String motor) {
14         this.motor = motor;
15     }
16
17     public int getPuertas() {
18         return puertas;
19     }
20
21     public void setPuertas(int puertas) {
22         this.puertas = puertas;
23     }
24
25     public int getRuedas() {
26         return ruedas;
27     }
28
29     public void setRuedas(int ruedas) {
30         this.ruedas = ruedas;
31     }
32
33     public void arrancar() {
34         System.out.println("El coche arranca.");
35     }
36
37     public void frenar() {
38         System.out.println("El coche frena.");
39     }
40
41     public void acelerar() {
42         System.out.println("El coche acelera.");
43     }
44 }
45

```

Los getters y setters permiten acceder a los atributos privados de una clase desde otra parte del código.

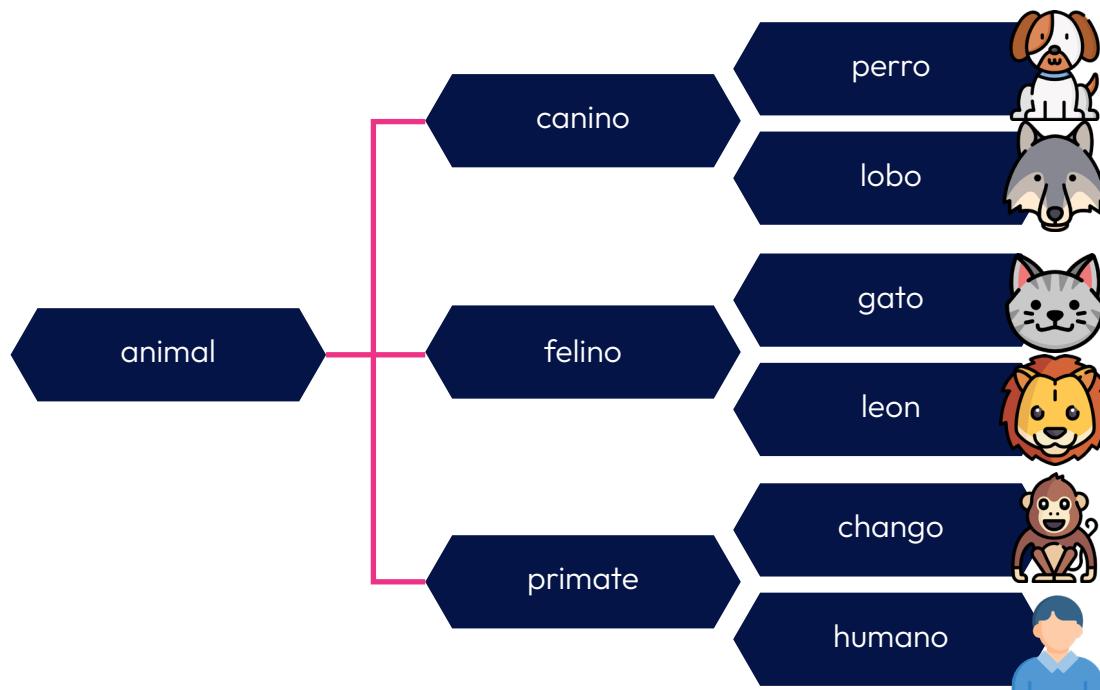
Ahora pongamos lo aprendido en práctica:

El primer ejercicio que haremos es: El Reino Animal

Te tomará 10 minutos.

Y para hacerlo necesitas:

- Diseñar en un archivo Animal.java la siguiente jerarquía de clases:



- En total debes de diseñar 10 clases con las siguientes características:
 - a)La clase padre debe de contener solo 3 atributos: nombre, sonido y extremidades, y 2 métodos: caminar y comunicar.
 - b)El segundo nivel de clases (canino, felino y primate) solo debe definir un nuevo atributo llamado especie.
- El programa debe imprimir una descripción de cada animal. ¿Dentro de qué clase debe de ir este método?

- Al crear una instancia de las clases hijas (perro, lobo, gato, león, chango y humano) se debe poder llamar los métodos caminar, comunicar y describir con los datos correctos para cada animal.
- Para este ejercicio debes de usar la herencia, todos los miembros de la clase deben tener un nivel de acceso default y se pueden usar constructores si es necesario.

¿Listo?

Aquí te dejamos un ejemplo de lo que debería imprimirse en consola y cómo construir el programa principal:

```

1  Main.java 2  Animal.java
2
3  public class Main {
4
5      public static void main(String[] args) {
6
7          Perro perro = new Perro();
8          Lobo lobo = new Lobo();
9          Gato gato = new Gato();
10         Leon leon = new Leon();
11         Chango chango = new Chango();
12         Humano humano = new Humano();
13
14         perro.caminar();
15         perro.comunicar();
16         perro.describir();
17         System.out.println();
18
19         lobo.caminar();
20         lobo.comunicar();
21         lobo.describir();
22         System.out.println();
23
24         gato.caminar();
25         gato.comunicar();
26         gato.describir();
27         System.out.println();
28
29         leon.caminar();
30         leon.comunicar();
31         leon.describir();
32         System.out.println();
33
34         chango.caminar();
35         chango.comunicar();
36         chango.describir();
37         System.out.println();
38
39         humano.caminar();
40         humano.comunicar();
41         humano.describir();
42         System.out.println();
43
44     }
45 }
46 
```

Console output:

```

terminated> Main [5] [Java Application] /Library/Java/JavaVirtualMachines/jdk-11.0.12.jdk/Contents/Home/bin/
El perro canina con 4 patas.
El perro se comunica con ladridos.
El perro es un animal doméstico que pertenece a la especie canina.

El lobo camina con 4 patas.
El lobo se comunica con aullidos.
El lobo es un animal salvaje que pertenece a la especie canina.

El gato camina con 4 patas.
El gato se comunica con maullidos.
El gato es un animal doméstico que pertenece a la especie felino.

El león camina con 4 patas.
El león se comunica con gruñidos.
El león es un animal salvaje que pertenece a la especie felino.

El chango camina con 4 patas.
El chango se comunica con gritos.
El chango es un animal salvaje que pertenece a la especie primate.

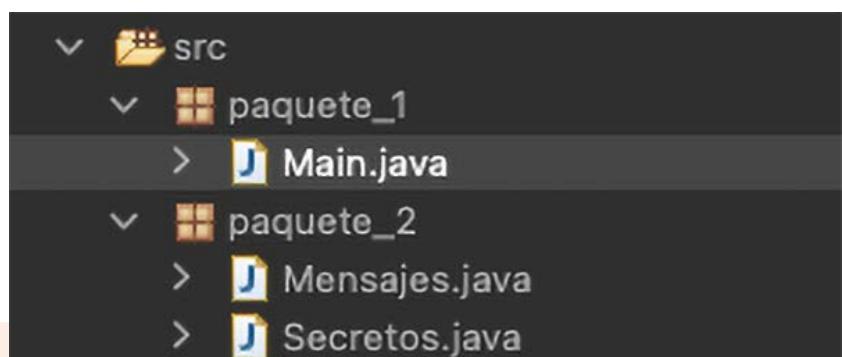
El humano camina con 2 pies.
El humano se comunica con palabras.
El humano es un animal que pertenece a la especie primate.

```

El segundo ejercicio que haremos es: Mensajes y secretos

Te tomará 10 minutos.

- Crear 2 paquetes en el proyecto.
- El primer paquete debe contener un archivo Main.java y el otro paquete dos archivos: Mensajes.java y Secretos.java con una clase dentro de ellos con el mismo nombre.



- Tu tarea es diseñar las clases Mensajes y Secretos añadiendo los niveles de acceso correcto.

- La clase Mensajes debe contener 3 atributos: mensaje_1, mensaje_2 y mensaje_3.
- La clase Secretos debe contener 6 atributos: secreto_1, secreto_2 y secreto_3, mensaje_secreto_1, mensaje_secreto_2 y mensaje_secreto_3.
- Los mensajes deben de poder ser impresos al llamar los atributos de la clase mensajes.
- Los mensajes_secretos solo deben de poder ser impresos desde la clase Mensajes.
- Los secretos solo deben de poder ser impresos desde la clase Secretos.

- Como tip adicional, recuerda que no solo se pueden crear instancias en el programa Main.java, una instancia se puede crear en cualquier lugar del proyecto (dependiendo de su nivel de acceso).

¿Listo?

Aquí te dejamos un ejemplo de cómo debería mostrarse el resultado en consola y el programa principal:



No olvides importar las clases del paquete 2 al paquete 1.

```

1 package paquete_1;
2
3 import paquete_2.Mensajes;
4 import paquete_2.Secretos;
5
6 public class Main {
7
8     public static void main(String[] args) {
9         // TODO Auto-generated method stub
10
11         Mensajes mensajes = new Mensajes();
12         Secretos secretos = new Secretos();
13
14         System.out.println(mensajes.mensaje_1);
15         System.out.println(mensajes.mensaje_2);
16
17         System.out.println();
18
19         mensajes.imprimir_mensaje_secreto(1);
20         mensajes.imprimir_mensaje_secreto(2);
21
22         System.out.println();
23
24         secretos.imprimir_secreto(1);
25         secretos.imprimir_secreto(2);
26
27     }
28
29 }

```

Console

```

<terminated> Main [6] [Java Application] /Library/Java/JavaVirtualMachines/jdk-11.0.12.jdk/Contents/Home/bin/java (Mar 20, 2022, 7:41:06 PM -)
Mensaje 1 es público.
Mensaje 2 es público.

Mensaje secreto 1 es default.
Mensaje secreto 2 es default.

Secreto 1 es privado.
Secreto 2 es privado.

```

Has terminado.
¡Felicidades!

El paso siguiente es realizar un quiz interactivo para poner a prueba lo que has aprendido hasta ahora.

¡Estamos emocionados por seguir
aquí contigo!



IMPORTANTE

Toda la información contenida en el presente documento es propiedad única, exclusiva y reservada de Exponential Education, S. de R.L. de C.V. (en lo sucesivo “Bedu”) y es considerada como Información Confidencial en términos de la legislación de Propiedad Industrial aplicable. Derivado de lo anterior, cualquier copia, retransmisión u otros usos distintos sin autorización expresa de Bedu, se encuentra estrictamente prohibida.