# POO en Java

Joselu Cantero

# Introducción POO en Java

La programación orientada a objetos (POO) es un paradigma de programación que usa objetos para crear aplicaciones.

Está basada en tres pilares fundamentales:

- Herencia
- Polimorfismo
- Encapsulación

Su uso se popularizó a principios de la década de 1990.

# Ventajas de la POO en Java

Lo interesante de la POO es que proporciona conceptos y herramientas con las cuales se modela y representa el mundo real tan fielmente como sea posible.

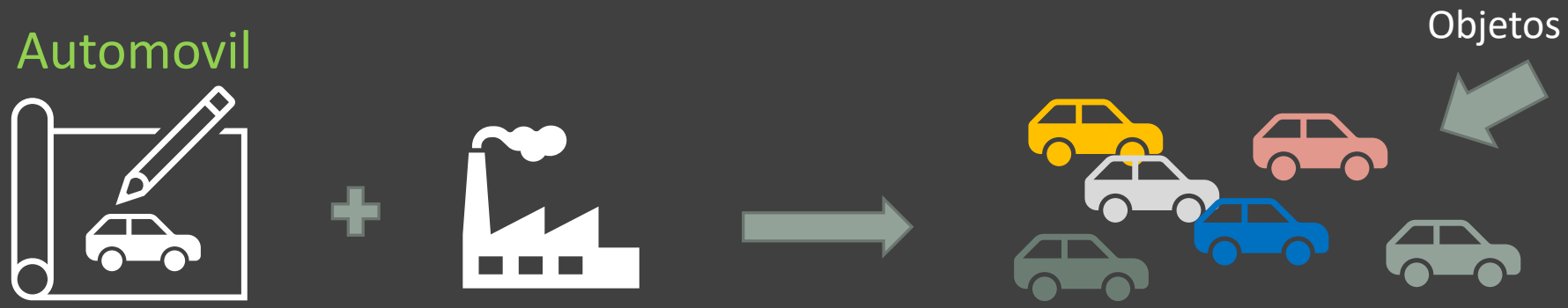
| REUSABILIDAD | MANTENIBILIDAD | MODIFICABILIDAD | FIABILIDAD |
|---|---|---|---|
| Cuando hemos diseñado adecuadamente las clases, se pueden usar en distintas partes del programa y en numerosos proyectos. | Debido a las sencillez para abstraer el problema, los programas son más sencillos de leer y comprender, pues nos permiten ocultar detalles de implementación dejando visibles sólo aquellos detalles más relevantes. | La facilidad de añadir, suprimir o modificar nuevos objetos nos permite hacer cambios de una forma muy sencilla. | Al dividir el problema en partes más pequeñas podemos probarlas de manera independiente y aislar mucho más fácilmente los posibles errores que puedan surgir. |

# Las Clases

En el mundo real, normalmente tenemos muchos objetos del mismo tipo. Por ejemplo, nuestro automóvil es sólo uno de los miles que hay en el mundo.

Si hablamos en términos de la programación orientada a objetos, podemos decir que nuestro automóvil es una instancia de una clase conocida como “Automovil".

Los automóviles tienen características (marca, modelo, color…) y comportamientos (acelerar, frenar…).




La Clase es una plantilla  Constructor de la clase  Instancias de la clase Automovil

# Las Clases

La clase es un modelo o prototipo que define las variables y métodos comunes a todos los objetos instanciados de ella.

También se puede decir que una clase es una plantilla genérica para un conjunto de objetos de similares características.

# Los Objetos

Un objeto del mundo real es cualquier cosa que vemos a nuestro alrededor. Por ejemplo un árbol, un automóvil...

No necesitamos ser expertos en mecánica para saber que un automóvil está compuesto por infinidad de componentes: las ruedas, el motor, los asientos... El trabajo en conjunto de todos estos componentes hace que un automóvil nos pueda trasladar de un lugar a otro.

Cada uno de estos componentes puede ser sumamente complicado y puede ser fabricado por diversas compañías con diversos métodos de diseño.

Cada componente es una unidad autónoma, y todo lo que necesitamos saber es cómo interactúan entre sí los componentes para poder montar el automóvil.

¿Que tiene que ver esto con la programación? La POO trabaja de esta manera. Todo el programa está construido en base a diferentes componentes (Objetos), cada uno tiene un rol específico en el programa y todos los componentes pueden comunicarse entre ellos de formas predefinidas.

# Los Objetos

La Clase plantea los atributos y los comportamientos que tendrán todas las instancias de esa clase

## Perro

¿?

String raza
String color
Int edad
String nombre

Perro()
correr()
jugar()
saltar()

new

new

new

raza: terrier
color: blanco
edad: 5
nombre: Toby

correr()
jugar()
saltar()

raza: pastor
color: crema
edad: 8
nombre: Luna

correr()
jugar()
saltar()

En la instanciación se definen esos atributos mediante el constructor Perro()

raza: labrador
color: negro
edad: 6
nombre: Lagun

correr()
jugar()
saltar()

# Los Objetos

Un objeto es una instancia de una clase en particular.

Todo objeto tiene unas características (atributos o propiedades) y unos comportamientos (métodos).

Cuando a las características del objeto le ponemos valores decimos que el objeto tiene estados.

Los atributos almacenan los estados de un objeto en un determinado momento.

# El método Constructor

Aunque en un principio pueda parecer lo contrario, un constructor no es en realidad un método estrictamente hablando. Un constructor es un elemento de una clase cuyo identificador coincide con el de la clase correspondiente y que tiene por objetivo obligar a y controlar cómo se inicializa una instancia de una determinada clase, ya que el lenguaje Java no permite que las variables miembro de una nueva instancia queden sin inicializar.

Además, a diferencia de los métodos, los constructores sólo se emplean cuando se quiere crear una nueva instancia.

Por defecto toda clase tiene un constructor sin parámetros cuyo identificador coincide con el de la clase y que, al ejecutarse, inicializa el valor de cada atributo de la nueva instancia: los atributos de tipo primitivo se inicializan a 0 o false, mientras que los atributos de tipo objeto (referencia) se inicializan a null.

Una vez implementado un constructor ya no se puede emplear el constructor por defecto sin parámetros. Si se desea trabajar con él, es necesario declararlo explícitamente.

# Llamada al método Constructor (new)

## Perro

```
String raza
String color
Int edad
String nombre

Perro( raza, color, edad, nombre){
        this.raza = raza;
        this.color = color;
        this.edad = edad;
        this.nombre = nombre;
}
correr()
jugar()
saltar()
```

constructor

```
new Perro( “terrier” , ”blanco” , 5 , ”Toby” );
```

Con new llamamos al constructor de la clase y le pasamos ordenadamente los valores que está esperando…


La instancia se crea en el Heap

# La Herencia

La herencia es uno de los conceptos más cruciales en la POO.

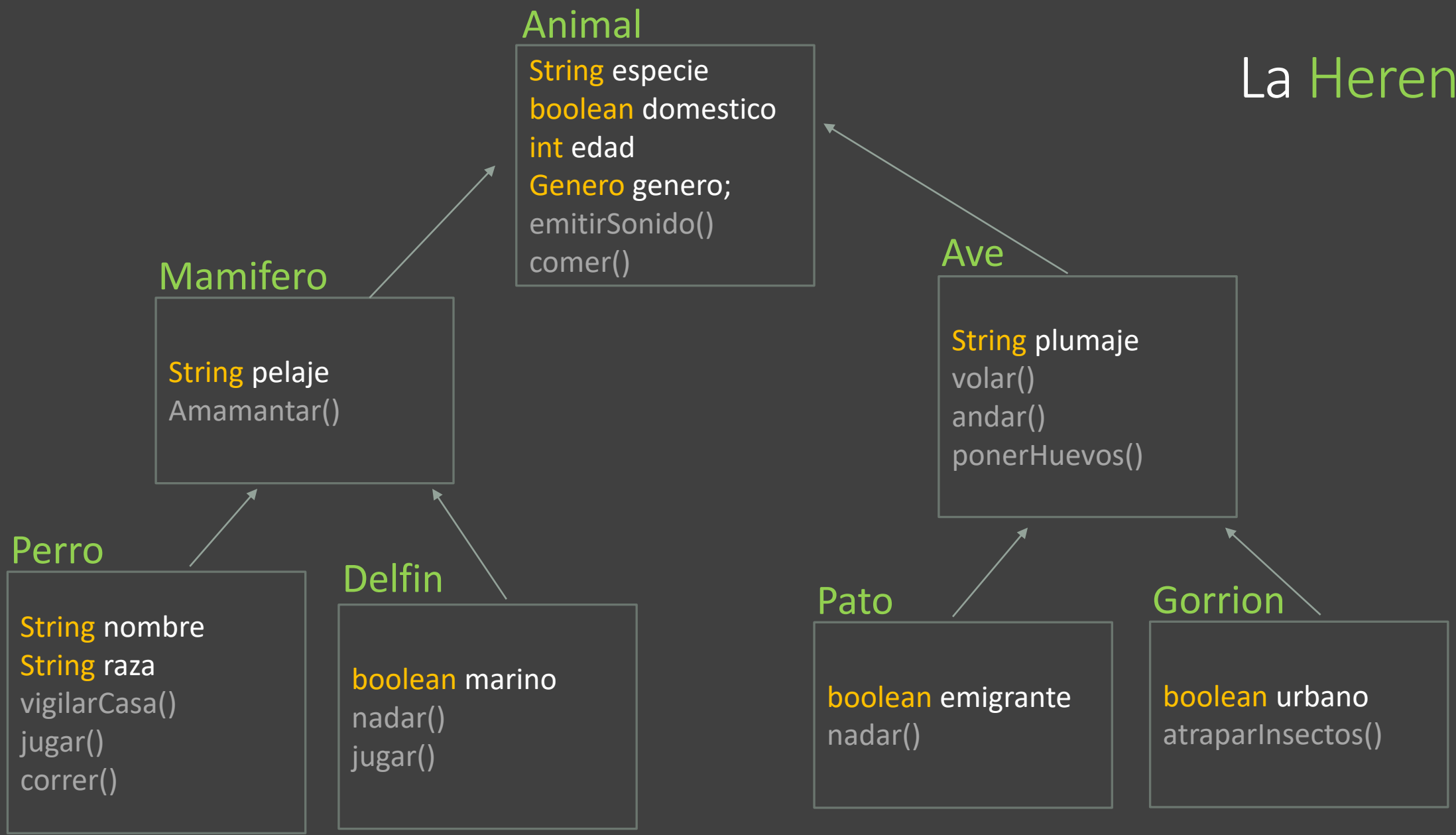
La herencia básicamente consiste en que una clase puede heredar sus variables y métodos a varias subclases (la clase de la que se hereda es llamada superclase o clase padre).

Esto significa que una subclase, aparte de los atributos y métodos propios, tiene incorporados los atributos y métodos heredados de la superclase.

De esta manera se crea una jerarquía de herencia.

| SUPERCLASE | SUBCLASE |
| --- | --- |
| La clase de cuyas características se heredan.<br>La clase “padre” de otra clase | La clase que hereda.<br>La clase “hija” de otra clase. |

# La Herencia




Joselu Cantero

# Tips cuadro Herencia previo 

1 - Animal es la superclase de Mamifero y de Ave

2 - Mamifero es la superclase de Perro y de Delfin

3 - Ave es la superclase de Pato y de Gorrion

4 - Mamifero y Ave son subclases de Animal

5 - Gorrion y Pato son subclases de Ave

6 - Perro y Delfin son subclases de Mamifero

7 - Delfin tiene sus propios métodos y atributos, pero también los de Mamifero y los de Animal

8 - Perro tiene sus propios métodos y atributos, pero también los de Mamifero y los de Animal

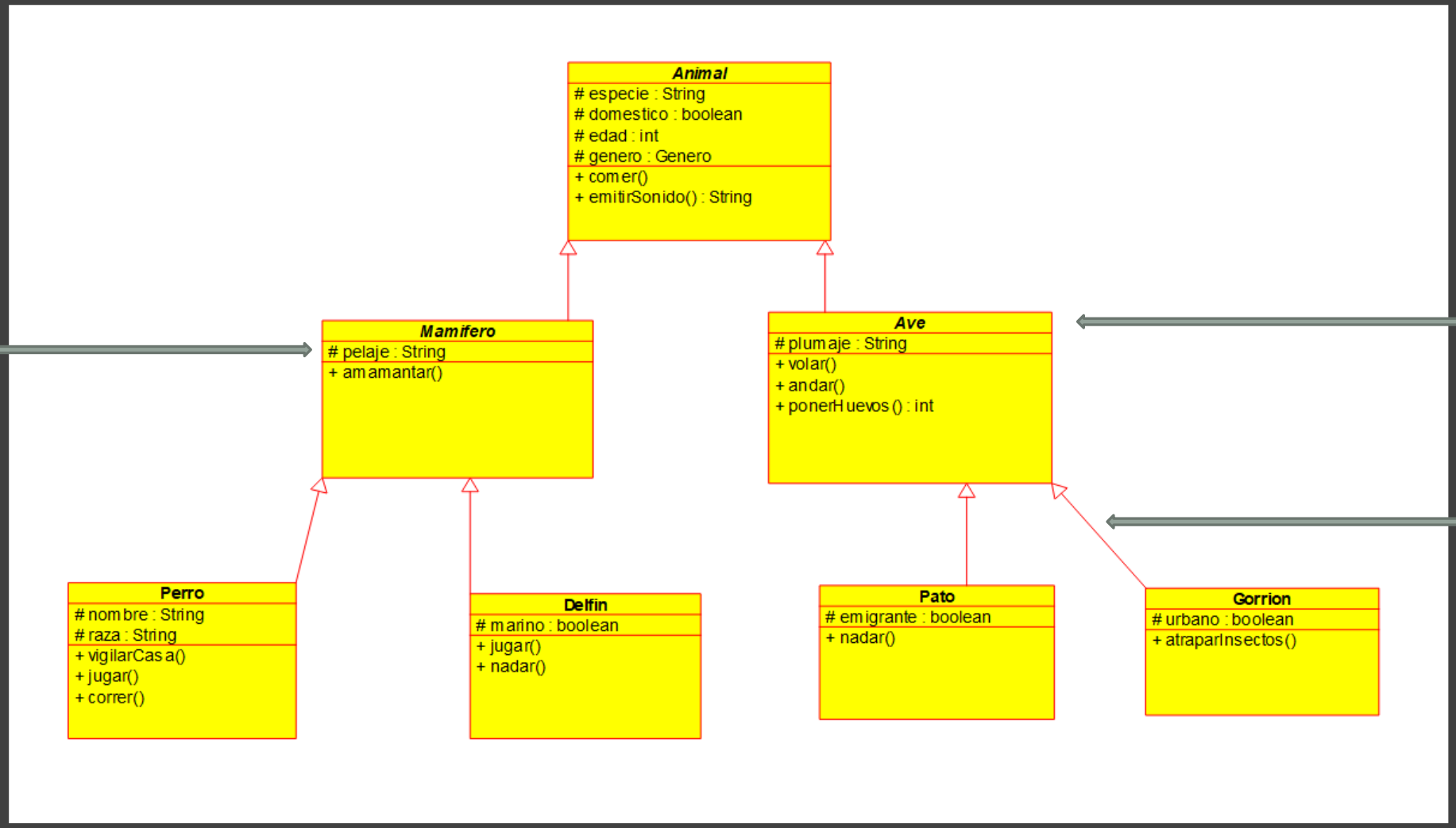
9 - Pato tiene sus propios métodos y atributos, pero también los de Ave y los de Animal

10 - Gorrion tiene sus propios métodos y atributos, pero también los de Ave y los de Animal

11 - Ave tiene sus propios métodos y atributos, pero también los de Animal

12 - Mamifero tiene sus propios métodos y atributos, pero también los de Animal

# Diagrama UML




# protected
+ public
- private

Clase Abstract en cursiva

Gorrion extends Ave

Joselu Cantero

# La Herencia - extends

La herencia en java se indica utilizando en la declaración de la clase la palabra reservada “extends”:

```
public class Ave  extends Animal{ miembros de la clase (atributos y métodos) }
```

La clase “Ave” pasa a ser una subclase de la clase “Animal”.

# La Herencia TIPS I

Existen dos clases, a las que llamaremos padre (superclase o clase base) e hija (subclase o clase derivada).

Al igual que las herencias en la vida real, la clase hija pasa a tener lo que tiene la clase padre: atributos y métodos.

Un objeto de la clase hija es también un objeto de la clase padre (un gorrión es un ave…pero un ave no es un gorrión).

# La Herencia TIPS II

En la clase hija se definen las diferencias respecto de la clase padre.

Se utiliza, entre otras cosas, para extender la funcionalidad de la clase padre y para especializar el comportamiento de la clase padre.

En Java, una clase solo puede tener una superclase.

En Java, una clase puede tener muchas subclases.

TODAS LAS CLASES HEREDAN IMPLICITAMENTE DE LA CLASE Object.

# El Encapsulamiento

El encapsulamiento consiste en aglutinar en la Clase las características y comportamientos (las variables y los métodos): tener todo en una sola entidad.

El encapsulamiento se logra gracias a la abstracción y el ocultamiento.

La utilidad del encapsulamiento reside en la facilidad que nos proporciona para manejar la complejidad, ya que tendremos a las Clases como cajas negras donde sólo se conoce el comportamiento pero no los detalles internos:  nos interesará conocer qué hace la Clase pero no será necesario saber cómo lo hace.

# La Abstracción

La abstracción consiste en captar las características esenciales de un objeto, así como su comportamiento. Por ejemplo, volvamos al ejemplo de los animales, ¿qué características podemos abstraer de los animales? O lo que es lo mismo: ¿qué características semejantes tienen todos los animales?.

Todos tendrán un genero, una especie, una edad, etc. Y en cuanto a su comportamiento todos los animales podrán comer, emitir sonidos etc.

En los lenguajes de programación orientada a objetos, el concepto de Clase es la representación y el mecanismo por el cual se gestionan las abstracciones.

# El Ocultamiento

Muchas veces confundido con el concepto de encapsulamiento.

El ocultamiento es la capacidad de ocultar los detalles internos del comportamiento de una Clase y exponer sólo los detalles que sean necesarios para el resto del sistema.

El ocultamiento permite 2 cosas: restringir y controlar el uso de la Clase:

1 - Restringir porque habrá cierto comportamiento privado de la Clase que no podrá ser accedido por otras Clases.

2 - Controlar porque daremos ciertos mecanismos para modificar el estado de nuestra Clase y es en estos mecanismos dónde se validarán que algunas condiciones se cumplan.

En Java el ocultamiento se logra usando las palabras reservadas: public, private y protected delante de las variables y métodos.

# El Polimorfismo en Java

El polimorfismo en la POO es un concepto íntimamente relacionado con la herencia.
No todos los lenguajes tienen en mismo tipo de polimorfismo que Java.

El polimorfismo es la habilidad de un método, variable u objeto de poseer varias formas distintas.
Podríamos decir que un mismo identificador comparte varios significados diferentes.

Java tiene 4 grandes formas de polimorfismo (aunque conceptualmente, muchas más):

| Polimorfismo de ASIGNACIÓN | Polimorfismo PURO | SOBRECARGA | Polimorfismo de INCLUSIÓN |
|---|---|---|---|

# Polimorfismo de Asignación

En Java, una misma variable de referencia (clases, interfaces…) puede hacer referencia a más de un tipo de Clase. El conjunto de las que pueden ser referenciadas está restringido por la herencia o la implementación (interface).

Esto significa, que una variable A declarada como un tipo, puede hacer referencia a otros tipos de variables siempre y cuando haya una relación de herencia o implementación (interface) entre A y el nuevo tipo. Podemos decir que un tipo A y un tipo B son compatibles si el tipo B es una subclase o implementación (interface) del tipo A.

```
Mamifero miPerro = new Perro(); //La clase Perro hereda de la clase Mamifero
```

```
IDomesticable miPerro = new Perro();//La clase Perro implementa la interface IDomesticable
```

Una variable de tipo Mamifero o de tipo IDomesticable puede apuntar sin problema a un objeto de tipo Perro.

# Polimorfismo Puro

En el polimorfismo puro un método puede recibir varios tipos de argumentos en tiempo de ejecución. Esto no lo debemos confundir con la sobrecarga, que es otro tipo de polimorfismo en tiempo de compilación.

```
public void funcionPolimorfica(IDomesticable obj){
    // La función acepta cualquier IDomesticable es decir, cualquier objeto que implemente esa interface
    // El tipo de objeto se determina en tiempo de ejecución.
}
```

```
public void funcionPolimorfica2(Animal obj){
    // La función acepta cualquier Animal: Perro, Delfin, Pato, Gorrion
    // El tipo de objeto se determina en tiempo de ejecución.
}
```

# Polimorfismo de Sobrecarga

En el polimorfismo de sobrecarga, dos o más métodos comparten el mismo identificador (nombre del método), pero distinta lista de argumentos. Al contrario que el polimorfismo puro, el tipado de los argumentos se especifica en tiempo de compilación:

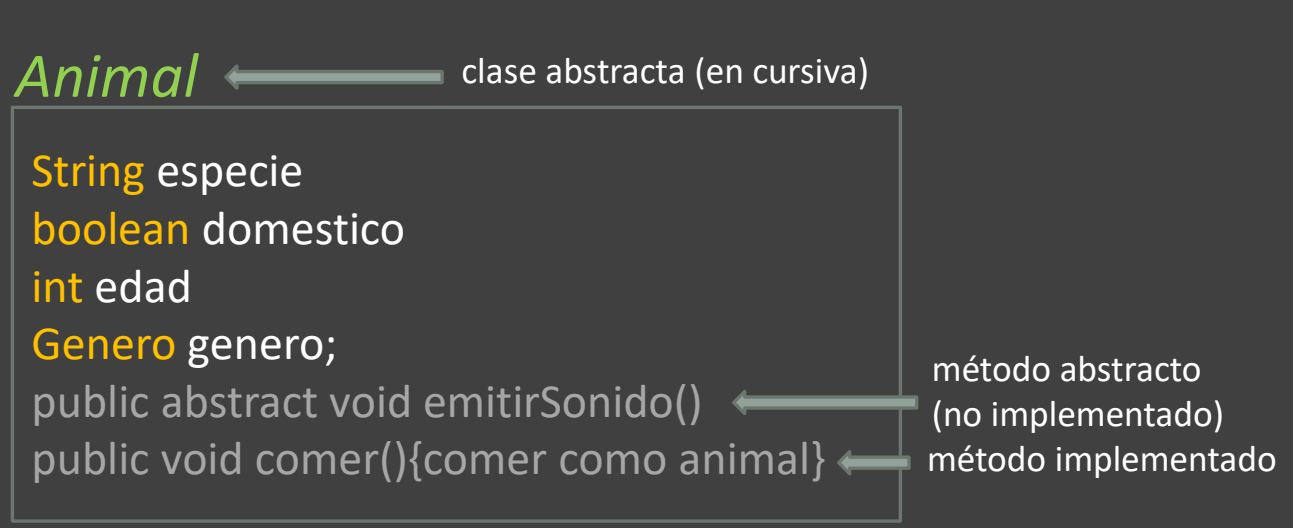
```
public String ladrar(int veces){ …hacer cosas… }

public String ladrar(int veces, int fuerza){ …hacer cosas… }

public String ladrar(int veces, int fuerza, boolean agresividad){ …hacer cosas… }
```

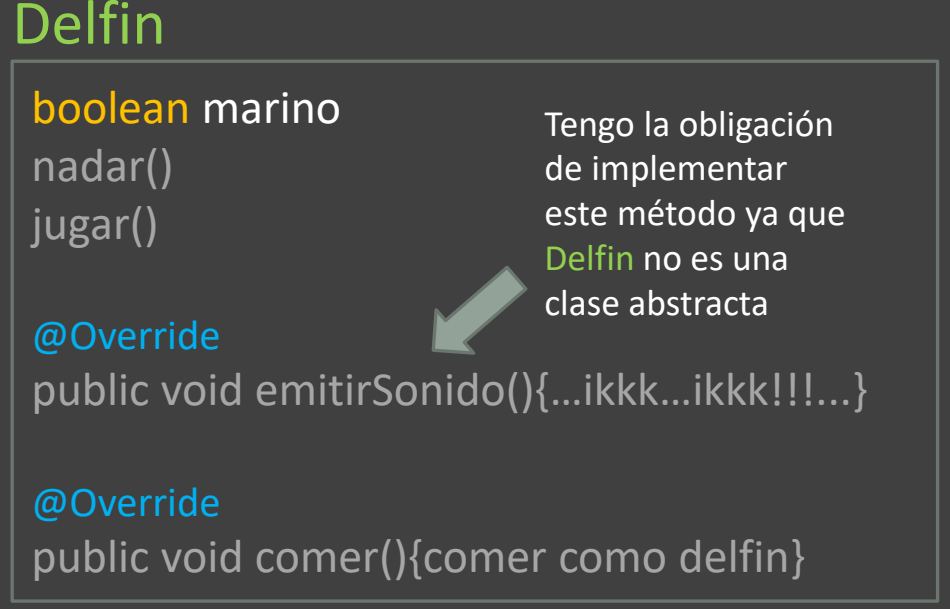
Vemos un mismo método (ladrar) con diferentes listas de argumentos. En función de los argumentos especificados en la llamada a su método (ladrar), la clase Perro utilizará uno u otro para adecuarse al contexto.  El polimorfismo de sobrecarga es de definición similar al polimorfismo puro, pero de implementación muy distinta.

# Polimorfismo de Inclusión

La habilidad para redefinir por completo un método de una superclase dentro de una subclase es lo que se conoce como polimorfismo de inclusión (o sobreescritura o redefinición).

En él, una subclase implementa un método que existe en una superclase respetando su lista de argumentos (si se define otra lista de argumentos, estaríamos haciendo sobrecarga y no redefinición).



# Diferencia entre Overriding y Overloading

## Overriding o Sobreescritura

La sobreescritura consiste en redefinir un método de una clase padre (super) dentro de una clase hija (sub) sin tocar los parámetros del método definido en la clase padre (super).

Es decir usando la misma firma.

Algunos textos se refieren a la Sobreescritura como polimorfismo en tiempo de ejecución.

## Overloading o Sobrecarga

La sobrecarga consiste en que dentro de una clase existen dos o más declaraciones de un método con el mismo nombre pero con parámetros distintos (distinta firma).

Podríamos afirmar que sobrecargar un método consiste en hacer diferentes versiones de él.

El compilador, sabe a qué método llamar en función de los parámetros que le estamos pasando.

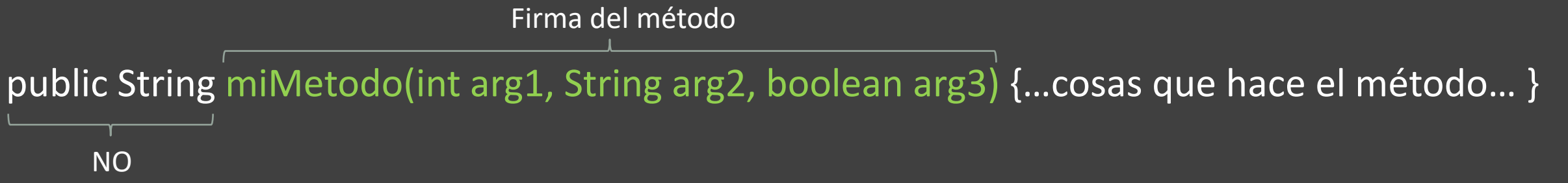
Algunos textos se refieren a la Sobrecarga como polimorfismo en tiempo de compilación.

# ¿Qué es la firma de un método?

Dos o más métodos dentro de la misma clase pueden compartir el mismo nombre, siempre que sus declaraciones de parámetros sean diferentes.

Cuando esto sucede, se dice que estamos usando sobrecarga de métodos (method overloading). En general sobrecargar un método consiste en declarar versiones diferentes de él. Y aquí es donde el compilador se ocupa del resto y donde el término firma cobra importancia.

Una firma es el nombre de un método más su lista de parámetros. Dentro de la firma no contamos con el tipo de retorno ni con el modificador de acceso.



# Otras preguntas frecuentes sobre Polimorfismo en Java

¿Se pueden sobrecargar métodos estáticos?: SI

¿Es posible sobrecargar la clase main() en Java?: SI

Siempre que definamos correctamente los parámetros de entrada como en el siguiente ejemplo:

```
public static void main(String[] args) {}

@Override
public static void main(String arg1) {}

@Override
public static void main(String arg1, String arg2) {}
```