# Introducción a la Orientación a Objetos

1º DAM

# ¿Qué es la Programación Orientada a Objetos?

Es un paradigma de programación, es decir, un modelo o un estilo de programación que se basa en el concepto de **clases y objetos**.

Hasta ahora hemos visto cómo **estructurar** el código para fomentar su **claridad y reusabilidad** pero no teníamos una forma de agrupar información de diferentes tipos, ni podíamos relacionar esas propiedades con comportamientos.

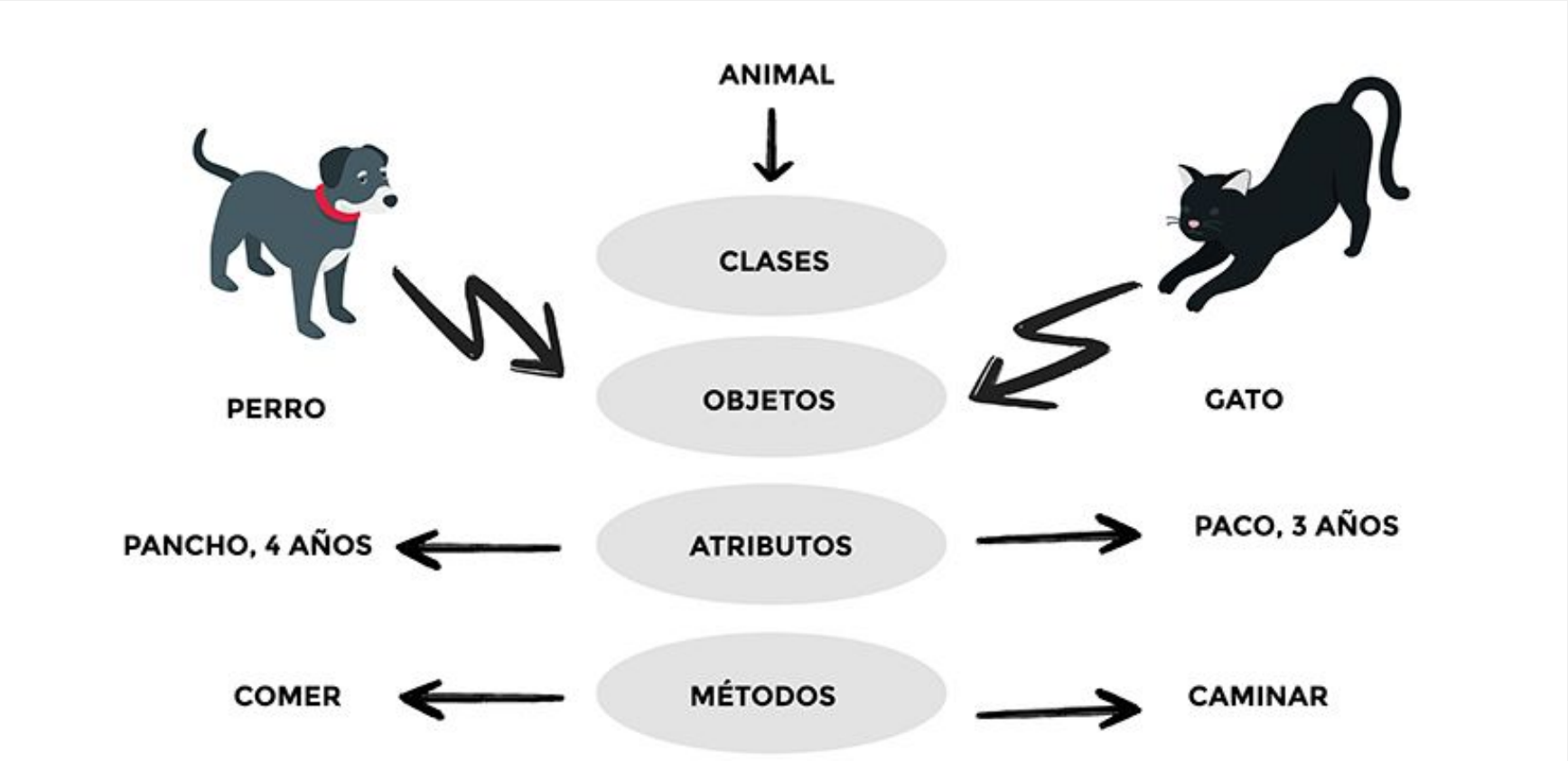
Con el paradigma de Programación Orientado a Objetos lo que buscamos es dejar de centrarnos en la lógica pura de los programas, para empezar a **pensar en objetos,** lo que constituye la base de este paradigma.

# Clases y objetos

Una **clase** es una plantilla. Define de manera genérica cómo van a ser los objetos de un determinado tipo.

Por ejemplo, una clase para representar a animales puede llamarse ‘Animal’ y tener una serie de atributos, como ‘nombre’ o ‘edad’ (que normalmente son propiedades), y una serie con los comportamientos que estos pueden tener, como caminar o comer, y que a su vez se implementan como métodos de la clase (funciones).

# Clases y objetos



# Clases y objetos

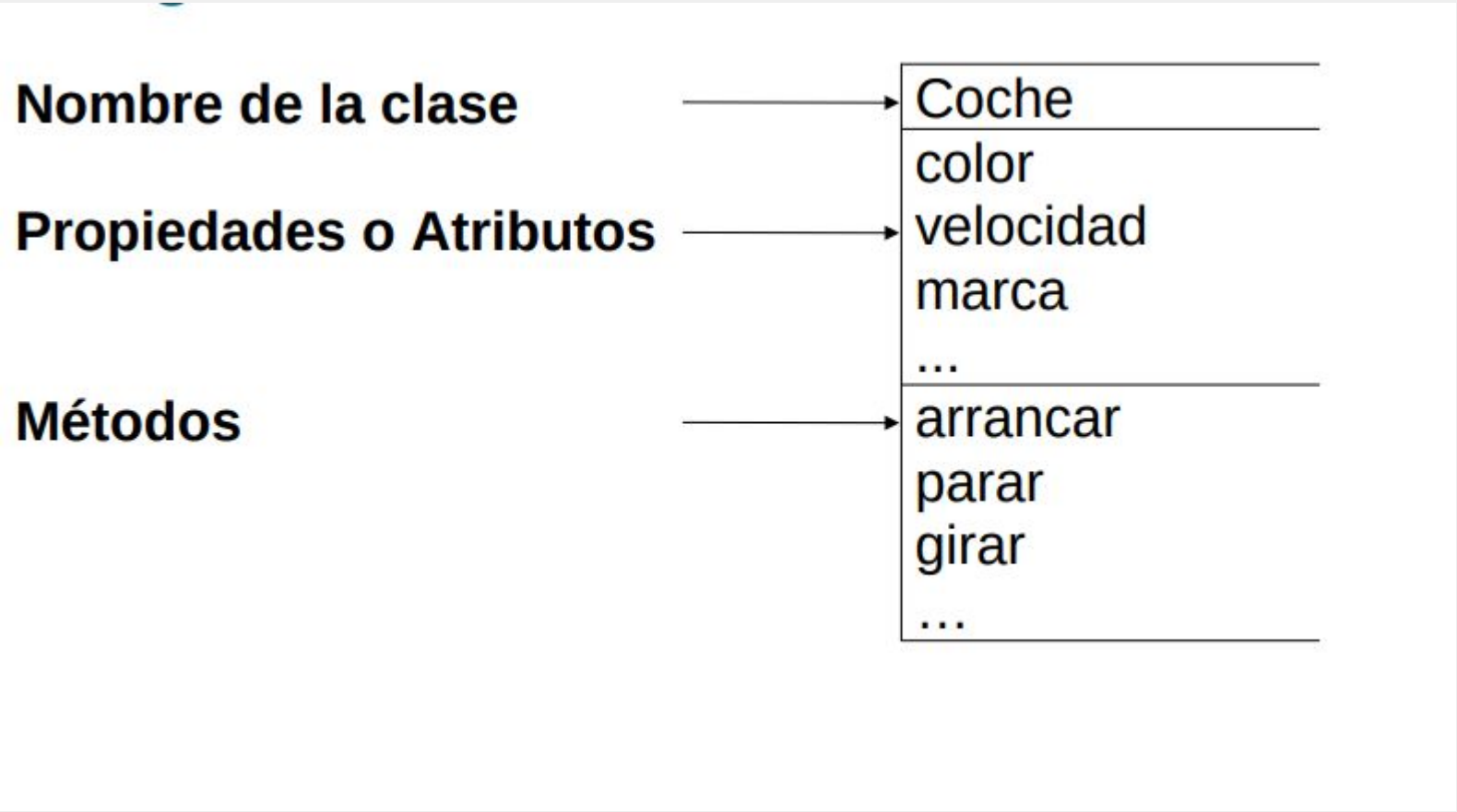
**Componentes de una clase:**

**Nombre de la Clase: Primera letra con mayúsculas**

- **Atributos**: son las propiedades de los objetos de la clase. minúsculas
- **Métodos**: Procedimientos u operaciones que comparten los objetos de la clase. Utilizar verbos
- **Constructores**: Procedimientos que se ejecutan en el momento de crear un objeto de la clase. Tienen el mismo nombre de la clase y pueden existir varios.

# Clases y objetos

## Diagramas de Clases



# Clases y objetos

Un **objeto** es un caso concreto de esa clase, una instancia.

La clase sería el molde con el cual se generan los objetos: define sus propiedades y su comportamiento Ejemplo: En una aplicación bancaria existirá la clase Cuenta, la clase Cliente, la clase Oficina etc.

La cuenta XXXX de Pepe Pérez será un objeto de la clase Cuenta

La oficina 0919 de la calle Sol será un objeto de la clase Oficina

# Declaración de una clase en Java

```
[modifAcceso] class NombreClase
{
    //Atributos o propiedades de la clase
    //(color, velocidad,…etc)
    ...
    //Constructor o constructores
    ...
    //Métodos de la clase
    //(arrancar, parar,…etc)
    ...
}
```

# Declaración de una clase en Java

**Modificador de acceso:**

- **public.** La clase es visible/ accesible desde cualquier clase
- Si no se indica nada, esa clase será accesible por todas las clases que se encuentren en el mismo paquete.

**Atributos**

```
[modifAcceso] [modifAtributo] tipo nombre [= valorInicial];
```

Un atributo está definido por:

- un tipo. Primitivo o de una Clase
- un nombre

Además puede tener unos modificadores.

# Declaración de una clase en Java

## Atributos

**Modificadores de visibilidad:**

| | |
|---|---|
| **public** | El método o atributo es siempre visible |
| **private** | El método o atributo es privado. Sólo es visible desde dentro de la propia clase |
| **protected** | El método o atributo es visible sólo por la propia clase y las que hereden de ella |
| | El método o atributo es visible pero sólo desde clases que se encuentren en el mismo paquete (friendly) |

**Modificadores de atributos:** static o final (ya lo veremos)

# Declaración de una clase en Java

## Métodos

Se corresponde con el comportamiento o funcionalidad que un objeto tiene por pertenecer a esa clase.

```
[modifAcceso] [modifMétodo] tipoRetorno  nombreMétodo (listaParámetros)
{
    //variables locales al método
    //código del método
    return expresión;   //cuando el tipoRetorno es void no es necesario
}
```

**nombre del método.** Debe ser algo descriptivo de la funcionalidad / comportamiento que contiene. Usar verbos

**tipo de retorno.** Puede ser un tipo primitivo, un objeto de una clase o no devolver nada (void). Esto se hace dentro del cuerpo del método con la palabra return.

# Declaración de una clase en Java

**Métodos: lista de parámetros.**

Se definirá la lista de argumentos como:

(tipo1 nombreArg1, tipo2 nombreArg2, …)

Los argumentos que recibe irán separados por comas.

Esta lista puede ser vacía. ()

Los argumentos serán variables de tipo primitivo u objetos de esta u otras clases.

# Declaración de una clase en Java

## Paso de parámetros

- Los argumentos de los **tipos básicos o primitivos** se pasan siempre **por valor.** El método recibe una copia del argumento actual; si se modifica esta copia, el argumento original que se incluyó en la llamada **no queda modificado**.
- Sin embargo, los **objetos** se pasan por **referencia**, es decir, se pasa el puntero a la zona de memoria donde está almacenado el objeto, es por ello que pueden **modificar los objetos referenciados**.

# Declaración de una clase en Java

**Creación de objetos y Constructores:**

Un constructor sirve para inicializar el objeto y establecer sus propiedades y valores predeterminados.

Un constructor en Java tiene e**l mismo nombre que la clase** a la que pertenece, y **no posee un tipo de retorno explícito,** lo que significa que **no** se utiliza la palabra clave **void ni ningún otro tipo de datos** para especificar el valor de retorno del constructor.

Un constructor puede aceptar **argumentos**, lo que permite configurar el objeto con valores específicos al momento de su creación. Estos parámetros se especifican en la definición del constructor, entre los paréntesis.

# Declaración de una clase en Java

## Creación de objetos y Constructores:

Por ejemplo, supongamos que tenemos una clase Persona con los campos nombre y edad. Podemos definir un constructor para la clase Persona:

```
public class Persona {
    String nombre;
    String apellidos;
    int edad;

    public Persona(
        String nombre, String apellidos, int edad)
    {
        super();
        nombre = nombre;
        apellidos = apellidos;
        edad = edad;
}}
```

# Declaración de una clase en Java

**Creación de objetos y Constructores:**

Los constructores se invocan mediante la palabra **new.**

```
Persona p = new Persona();
Persona p1 = new Persona("María", "Mora", 22);
```

Prueba ahora a :

1. Usar este constructor e imprime luego sus atributos
2. Crear un objeto mediante new Persona(); ¿Qué ocurre?

**Tipos de constructores:**

- **Por defecto.**
- **Con parámetros.**

# Declaración de una clase en Java

**Constructor por defecto:**

El constructor predeterminado lo proporciona automáticamente Java **sólo en caso de que no se defina ningún constructor explícitamente** en la clase.

Este constructor no toma ningún parámetro y su cuerpo está vacío. Simplemente inicializa los miembros de datos de la clase con sus valores predeterminados e invoca al constructor de su tipo padre (**super();**)

```
public Persona() {
        super();
    }
```

# Declaración de una clase en Java

## Constructor con parámetros

El constructor parametrizado toma uno o más parámetros y se utiliza para inicializar los miembros de datos de la clase con valores específicos proporcionados por el usuario.

Un constructor parametrizado debe definirse explícitamente en la clase y se utiliza para crear objetos con diferentes estados y comportamientos, en función de los valores proporcionados en los parámetros.

# Declaración de una clase en Java

**Uso del this y del super:**

La palabra reservada this nos sirve para referirnos al propio objeto instanciado de una clase, de forma que sus atributos y métodos pueden ser accesibles desde dentro de la propia clase con this.nombreatributo o this.nombreMetodo()

Por ejemplo, el constructor con parámetros definido antes podría ser reescrito de esta forma, y sería equivalente:

```
public Persona(
String nombre, String apellidos, int edad) {
        super();
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.edad = edad;
    }
```

# Declaración de una clase en Java

**Uso del this y del super:**

Es tu turno:
Construye un método dentro de la clase Persona que me cree una persona con los mismos valores para cada atributo pero que no sea una referencia a la misma zona de memoria.

```
Persona creaCopia()
{

    // Completa al código
}
```


¿Dudas?