Introducción a la P.O.O.

Antonio Espín Herranz

La programación orientada a objetos

- Se rompe la dependencia con la máquina y se piensa mas como en el mundo real.
- Sigue siendo imperativa.
- Es más fácil realizar modificaciones, así como la integración de las distintas partes.
- Se crean aplicaciones de una forma mas modular.

Conceptos y Características

CONCEPTOS

- Clase
- Objeto
- Instancia
- Método
- Propiedad / Atributo
- Colección

CARACTERÍSTICAS

- Herencia
- Polimorfismo
- Abstracción
- Encapsulación
- Reutilización
- Modularización

Diseño O.O.

- Cambia la forma de pensar:
- En comparación a la programación estructurada lo podríamos comparar como una estructura que tiene asociada una funcionalidad (TADs) pero con posibilidades mas amplias.
- Tipos primitivos → TADs → Objetos
- Perdemos un parámetro:
  - Estructurada: **complejo=sumar(complejo1, complejo2);**
  - POO: **complejo = complejo1.sumar(complejo2);**
  - Diremos que un objeto recibe el mensaje de súmate con …

Ejemplo de la P.O.O.

- Supongamos una fábrica de coches. En esta se define todas las características y funcionamiento de los coches que se va a construir → **Clase**.

- Cada Coche (**Objeto**) se creará con la definición de la clase → se instanciará con una serie de valores. Tendrá una serie de propiedades.

- Propiedades del Coche → **Atributos**.
  - Color, número de puertas, potencia.
  - Las ruedas: Son todas iguales, que tendrán una presión, modelo, marca. Se pueden agrupar en una **Colección**. Colección de Objetos Rueda.

- Y una serie de servicios / comportamiento → **Métodos**
  - frenar(), arrancar(), parar(), acelerar()

Definiciones

- **Clase**: Plantilla que define las características de un objeto.
  - De que se componen los objetos de una determinada clase y que hacen.
- **Objeto**: Es un elemento que pertenece a una clase que tiene una serie de propiedades y un comportamiento propio.
  - Las propiedades ya tienen valores asignados.
- **Instancia**: Representa la creación de un objeto a partir de una clase. Objeto = Instancia.
- **Propiedad**: Es una característica de un objeto.
  - ¿De que color es el coche?
  - ¿Cual es el suelo del empleado?
- **Método**: Es un servicio o comportamiento que nos ofrece un objeto.
  - Algo que podemos hacer con el objeto.
  - Es una acción. coche.arrancar();
- **Colección**: Un conjunto de objetos que en nuestro caso pertenecerán a la misma clase.
  - Clases contenedoras, quiero mantener todos los empleados de mi empresa

Relaciones entre objetos

- Cada objeto asume una responsabilidad.
- Se deben de tratar como entes independientes.
- Se crean distintas relaciones entre los objetos:
  - Un objeto puede estar formado por otros mas sencillos. **Relación de Composición.**
  - **Relación de Asociación**. Dos objetos interactúan para llevar a cabo una determinada función.
  - **Relación de Herencia**: A partir de una clase podemos especializarla. Partimos de algo genérico lo vamos completando.

Relación por Composición

- En el caso del Coche.
  - Podemos decir que un coche se compone de:
    - Radio.
    - Motor.
    - Etc.

    - Cada parte integrante se puede ver como un objeto en sí.
    - Que tendrá un determinado estado representado por propiedades / atributos.
    - Y una determinada funcionalidad.

    - miCoche.radio.on();
    - miCoche.motor.potencia = 2000;

Relación por Asociación

- En el caso del Coche, si necesitamos repostar podemos ver el Surtidor como otro objeto.
- En un instante en el tiempo se asocian.
  - miCoche.repostar(unSurtidor)
- Clase Surtidor:
  - Propiedades:
    - Tipo_de_combustible,
    - Capacidad
    - Capacidad_max
  - Métodos:
    - Surtir();
    - hayCombustible()

Tenemos una clase Persona y queremos grabar sus datos en un fichero.

La clase Persona puede interactuar con un stream para grabar.

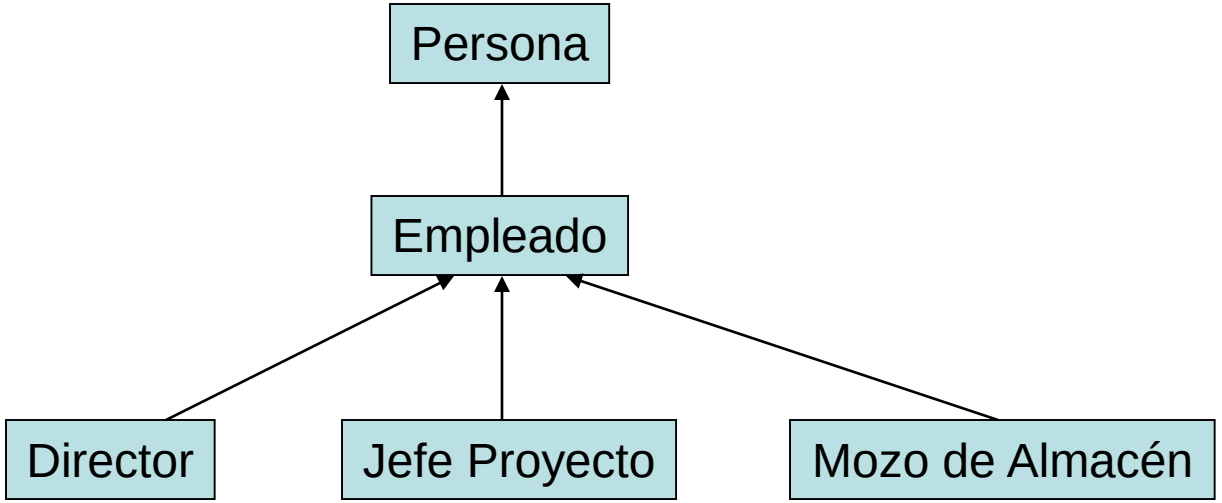
Relación por Herencia

- En los lenguajes de POO tenemos dos tipos de Herencia, Simple y Múltiple.
- No todos los lenguajes de POO soportan Herencia Múltiple.
- En el caso de C++, si está soportada.

Herencia Simple

- Nos permite crear jerarquías:
- En donde la clase mas alta en la jerarquía es la mas genérica (la llamaremos superclase) y las que cuelgan de esta serán subclases.
- Según bajamos por el árbol de herencia nos vamos especializando.
- Cuando una clase hereda de otra, hereda todo atributos y métodos (que serán o no accesibles)

Herencia Simple



La clase Empleado hereda de Persona,

Persona es la superclase de Empleado.

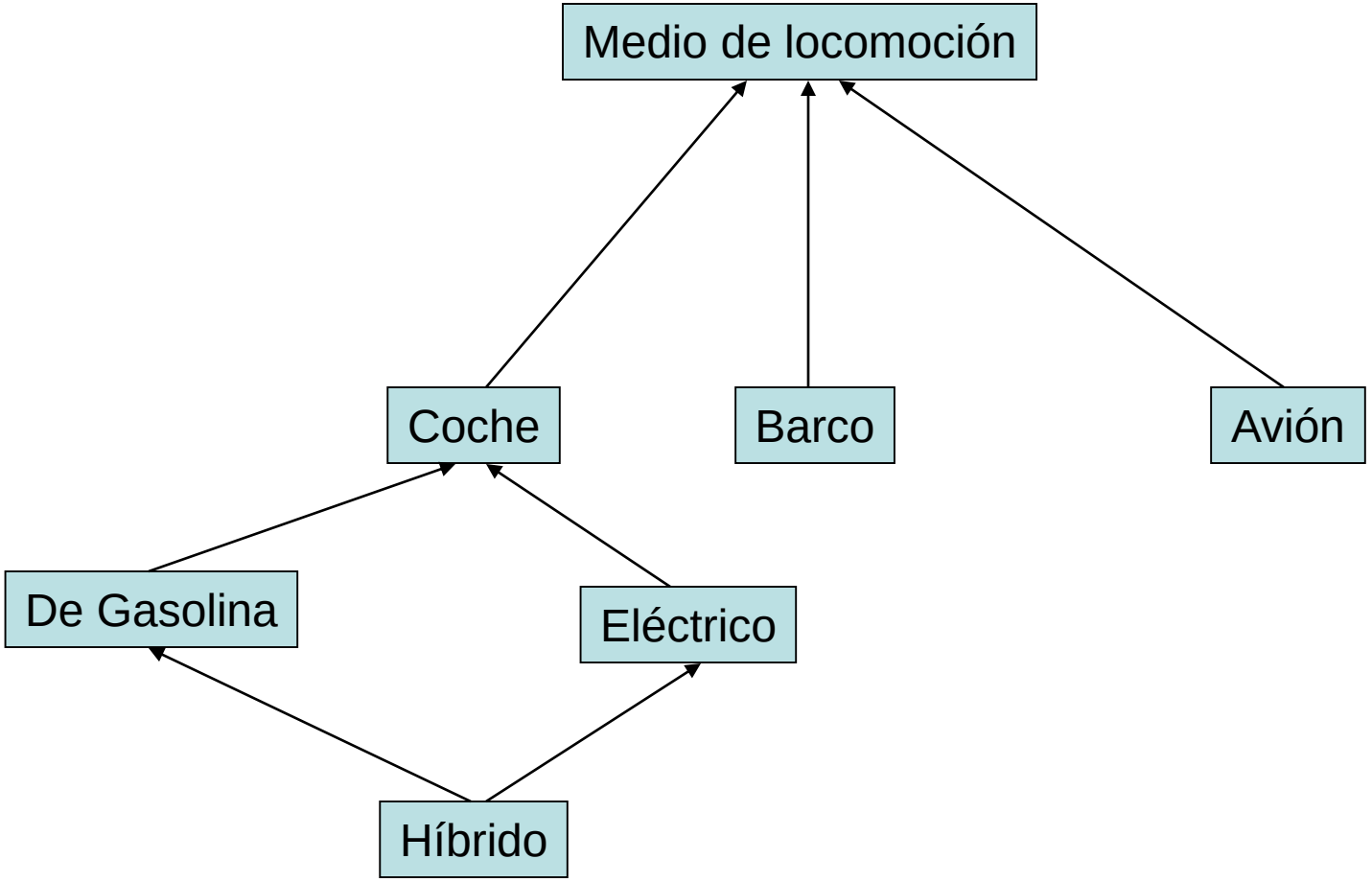
Empleado es subclase de persona y superclase de Director, Jefe de proyecto, Mozo de Almacén

Herencia Múltiple

- Vamos a poder heredar de varias clases.

- Pero con el mismo criterio de antes, siempre las clases mas altas en la jerarquía son las mas genéricas y según vamos bajando por el árbol son más especializadas.

Herencia Múltiple



Encapsulación

- El diseño de una clase solo la conoce el diseñador.
- Al usuario / programador de la clase no le deberíamos permitir acceder directamente a las propiedades de la clase.
- Le ofrecemos un interfaz público para poder trabajar con esa clase.
  - En el caso del coche: el método arrancar() poner en marcha el motor pero a mi se me ocultan todos los detalles.
- La encapsulación favorece un bajo acoplamiento y repercute directamente en un fácil mantenimiento.
- Permite alterar el diseño interno de la clase y si no modificamos el interfaz público de la clase. El usuario / programador ni se entera.

Modularización

- Cambia la forma de pensar, vemos las aplicaciones como un conjunto de objetos que interactúan entre sí.

- Repercute en la **reutilización** del código. Podemos utilizar clases que ya están implementadas y depuradas.

- La **colaboración**: cada equipo / programador puede desarrollar unas determinadas clases.

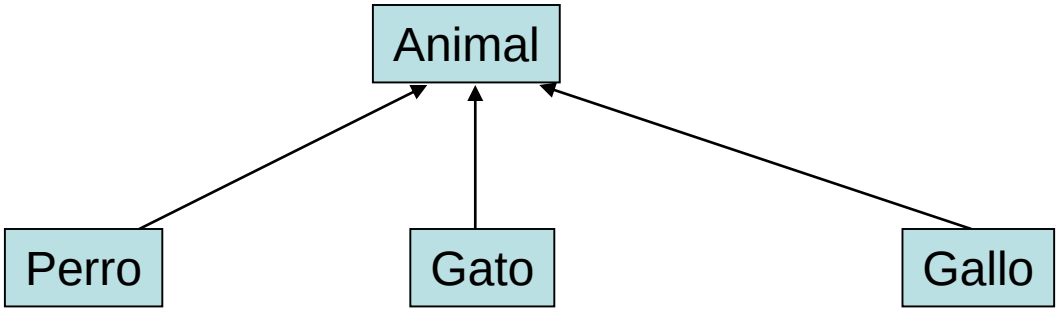
Polimorfismo

- Es la capacidad que tienen los objetos de responder al mismo método pero con un comportamiento distinto especificado en cada clase.

- El mismo método (con el mismo prototipo) tiene un comportamiento distinto según a la clase que pertenece.

Ejemplo

- Supongamos la siguiente relación de herencia:

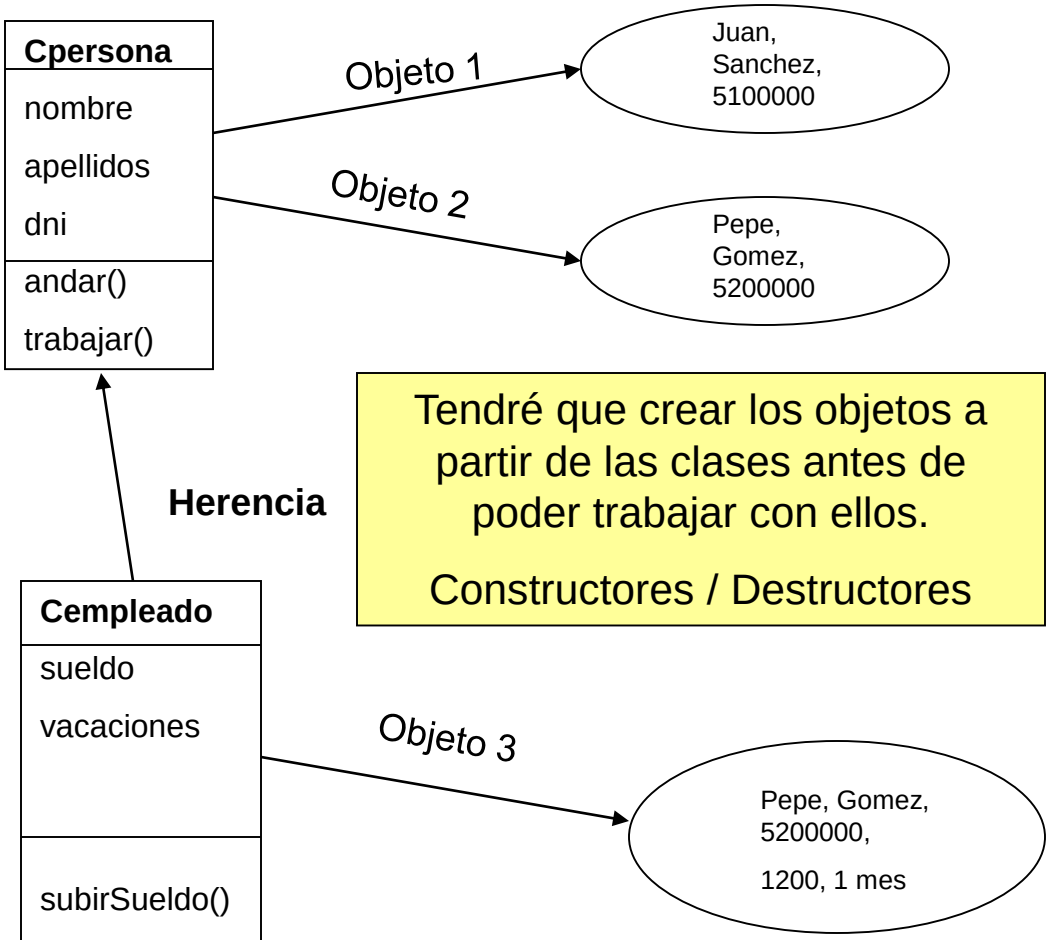


- Todos los objetos podrían tener el método **emitirSonido**(), a nivel de Animal yo no sé que sonido emitirá cada uno, pero en el caso de un animal concreto si puedo saber el sonido que emiten.
- El perro, el gato, el gallo son animales.
- Todos los animales emiten un sonido.
- El sonido emitido es distinto dependiendo del animal.
- El polimorfismo siempre está ligado a una relación de herencia.

Abstracción

- Abstracción consiste en aislar un elemento de su contexto o del resto de los elementos que lo acompañan.
- Tiene que ver con el diseño OO ver el sistema como un conjunto de entidades (objetos) que interactúan entre sí.
- Nos centramos en el objeto coche, viendo sus propiedades y funcionalidades.

Al trabajar con Objetos



Tendré que crear los objetos a partir de las clases antes de poder trabajar con ellos.

Constructores / Destructores

A partir de la clase o plantilla podemos crear objetos.

Cada objeto es independiente , almacena un estado y sólo se puede manipular a través de su funcionalidad.

Podemos enviar un mensaje al Objeto 1.

Objeto1.trabajar()

Ventajas de la POO

- Fomenta la reutilización y extensión del código.
- Permite crear sistemas más complejos.
- Relacionar el sistema al mundo real.
- Agiliza el desarrollo de software.
- Facilita el trabajo en equipo.
- Facilita el mantenimiento del software.

Cuestiones de nomenclatura

- Es una buena costumbre adoptar unas bases a la hora de codificar con objetos.

- Por ejemplo:
  - Los nombres de las clases pueden empezar con mayúsculas.
    - Coche, Empleado, Persona.
  - Las propiedades y los métodos:
    - 1ª letra con minúscula,
      - Propiedades:
        - » color, sueldo, etc.
      - Métodos:
        - » arrancar(), subirSueldo();

Cuestiones de diseño

- A la hora de diseñar la aplicación podemos realizar un diseño buscando las entidades del sistema: Empleado, Persona.

- O se pueden asociar clases a distintos dispositivos o soportes: Base de datos, video, etc.

- Cuando los diseños se complican es interesante aplicar **Patrones de Diseño**.