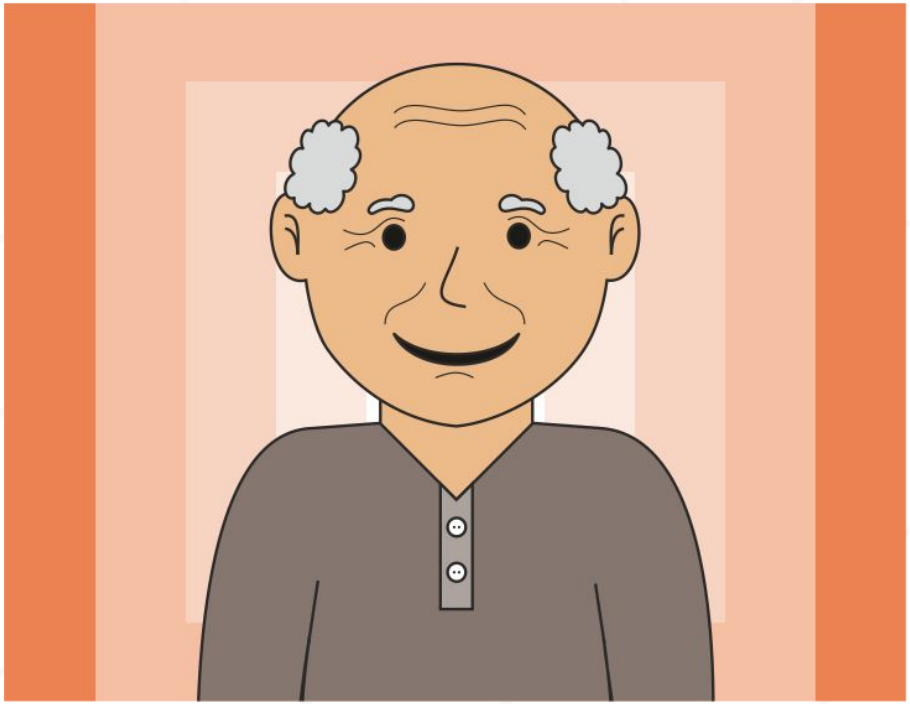
Herencia

DEV.F:
DESARROLLAMOS(PERSONAS);

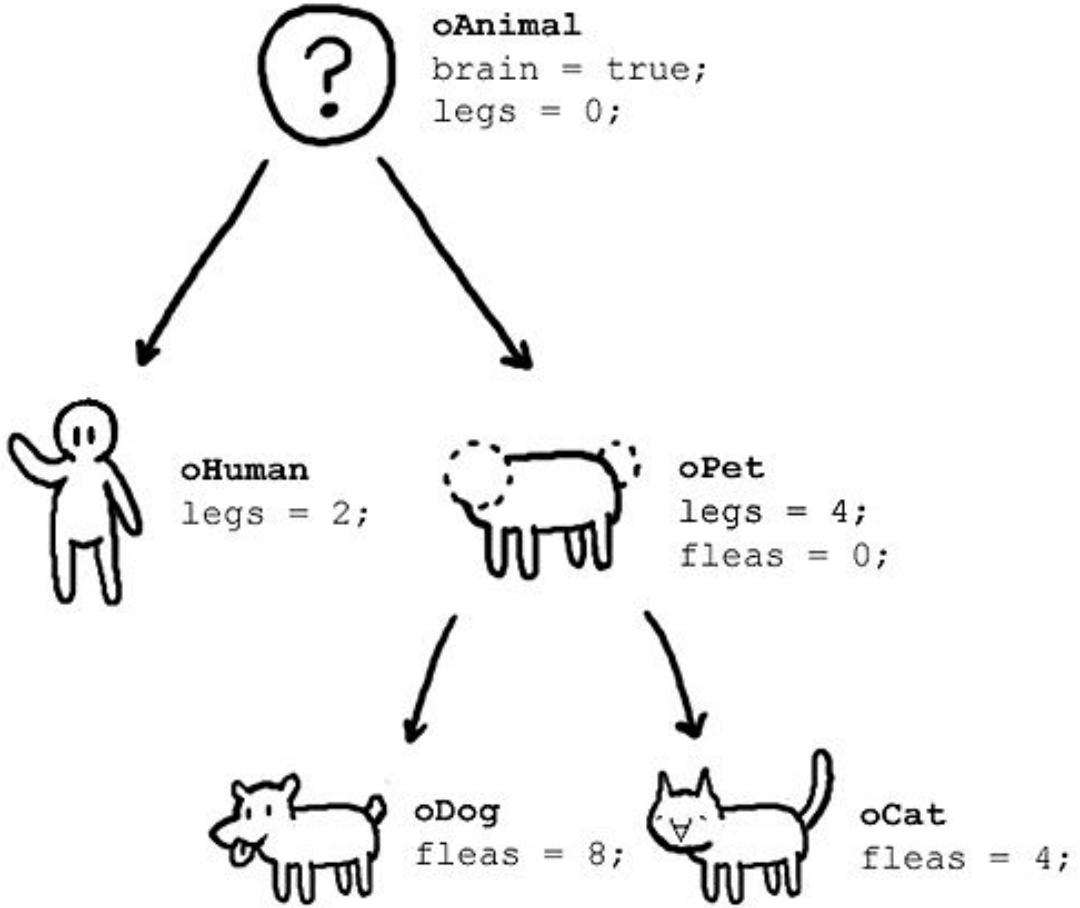
Herencia

Es una clase nueva se crea a partir de una clase existente, esta clase comparte los atributos y comportamientos a la principal.

Otra ventaja de la herencia es la capacidad para definir atributos y métodos nuevos para la subclase.



DEV.F

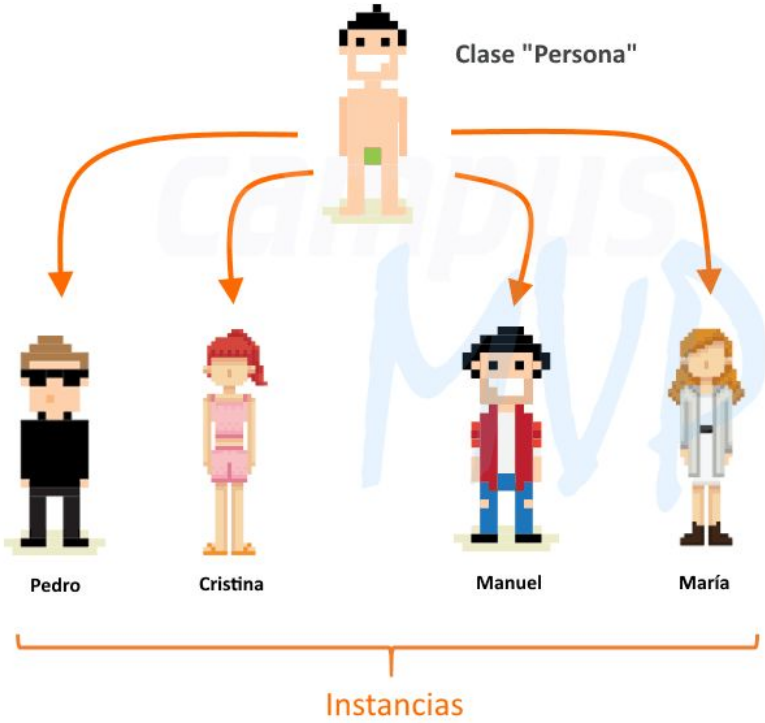

oAnimal
brain = true;
legs = 0;
oHuman
legs = 2;
oPet
legs = 4;
fleas = 0;
oDog
fleas = 8;
oCat
fleas = 4;

Pilares de la POO

Programación Orientada a Objetos

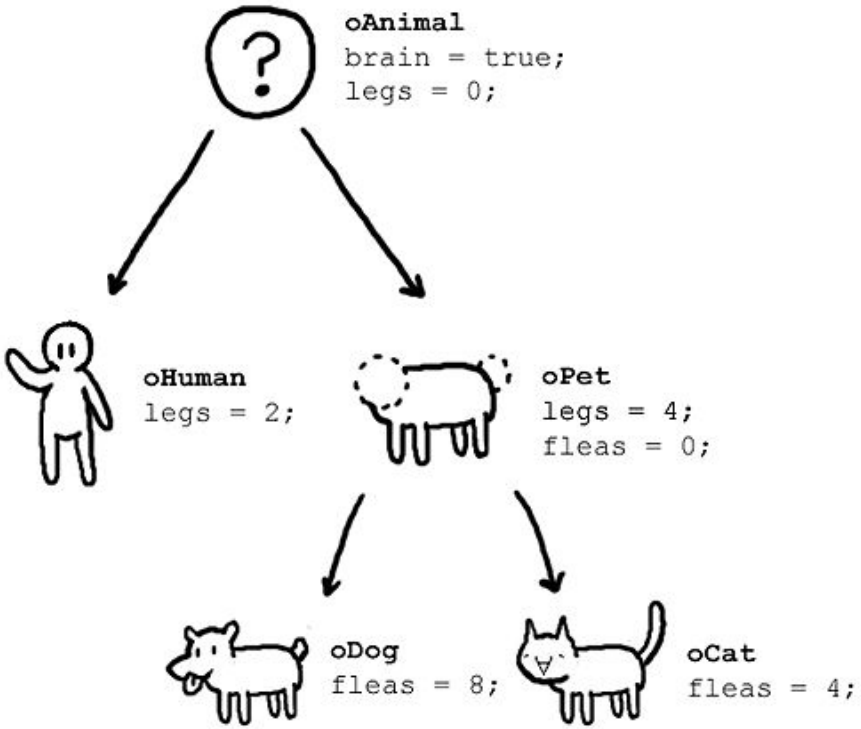
Abstracción

- Debe enfocarse a lo mínimo.
- Se busca definir atributos y métodos más relevantes.
- Eventualmente como programadores desarrollamos la capacidad de abstracción.



Encapsulamiento

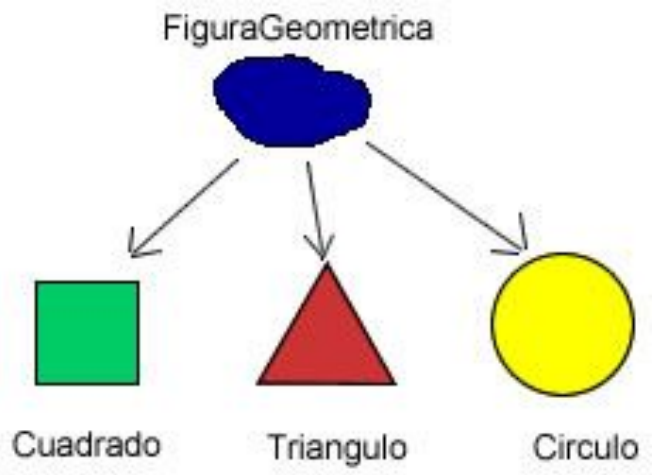
- Hablamos de agrupamiento y protección.
- Colocar atributos y métodos en un mismo lugar (Clase)
- Se busca lograr que un objeto no revele los datos de si mismo a menos que sea necesario.



Herencia

- Crear una clase a partir de una existente.
- Superclase.
- Subclase.
- Se heredan atributos y métodos.

DEV.F::



Polimorfismo

- Se utiliza cuando una clase hereda sus atributos y métodos.

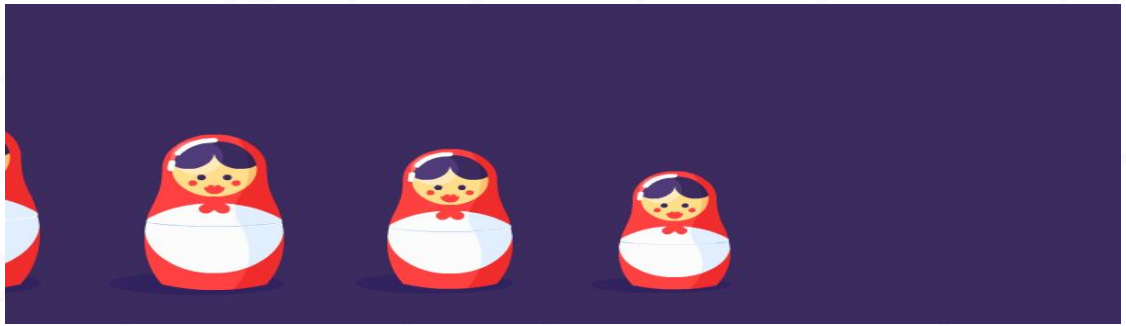
- Sobreescritura de métodos.

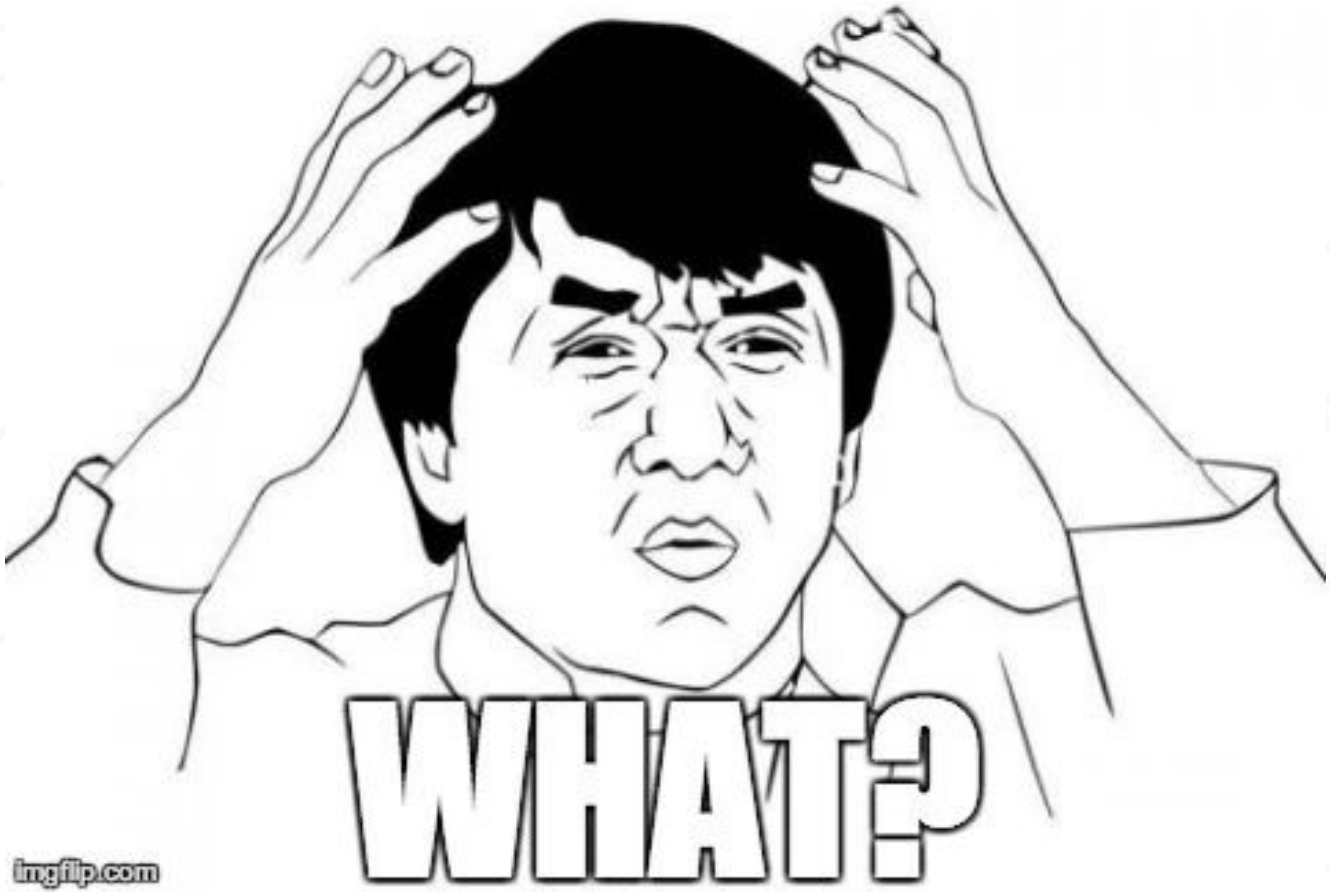
DEV.F::

Liskov substitution principle (Principio de POO)

En lenguaje más formal:

Si **S** es un subtipo de **T**, entonces los objetos de tipo **T** en un programa de computadora pueden ser sustituidos por objetos de tipo **S** (es decir, los objetos de tipo **S** pueden sustituir objetos de tipo **T**), sin alterar ninguna de las propiedades deseables de ese programa (la corrección, la tarea que realiza, etc).




WHAT?
imgflip.com

Ejemplo

En un cine se reproducen largometrajes. Puedes, no obstante, tener varios tipos de largometrajes, como películas, documentales, etc.

Quizá las películas y documentales tienen diferentes características, distintos horarios de audiencia, distintos precios para los espectadores y por ello has decidido que tu clase "Largometraje" tenga clases hijas o derivadas como "Película" y "Documental".

Imagina que en tu clase "Cine" creas un método que se llama "reproducir()".

Este método podrá recibir como parámetro aquello que quieres emitir en una sala de cine y podrán llegarte a veces objetos de la clase "Película" y otras veces objetos de la clase "Documental".

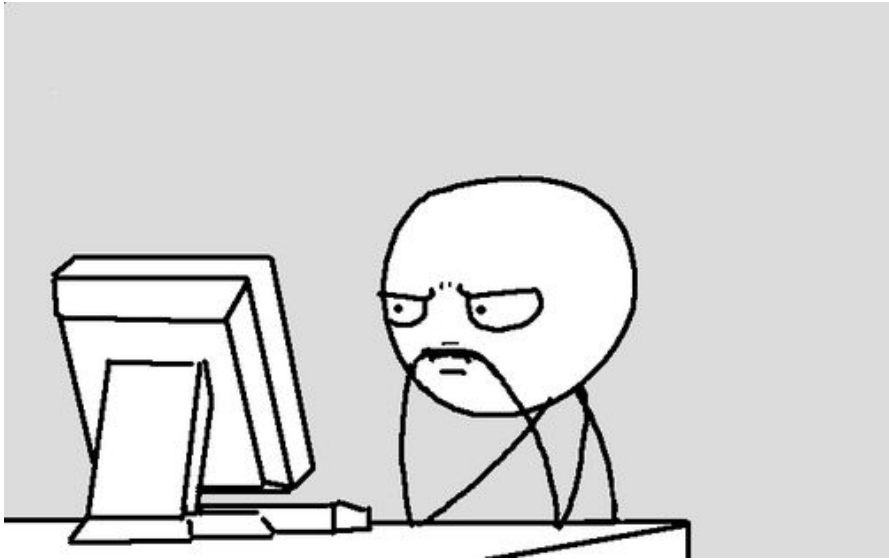
Si quisiera reproducir una película tendría los siguiente:

reproducirPelicula(peliculaParaReproducir){... }

Pero si luego tienes que reproducir documentales, tendrás que declarar:

reproducirDocumental(documentaParaReproducir){... }

¿Realmente es necesario hacer dos métodos?



SOLID

DEV.F::
DESARROLLAMOS(PERSONAS);

Single Responsibility (Responsabilidad única)

Open Closed

Liskov Substitution

Interface Segregation

Dependency Inversion

- ***Principio de responsabilidad única***

Una clase debe preocuparse por una sola responsabilidad.

Mantén clases y métodos pequeños

KISS:

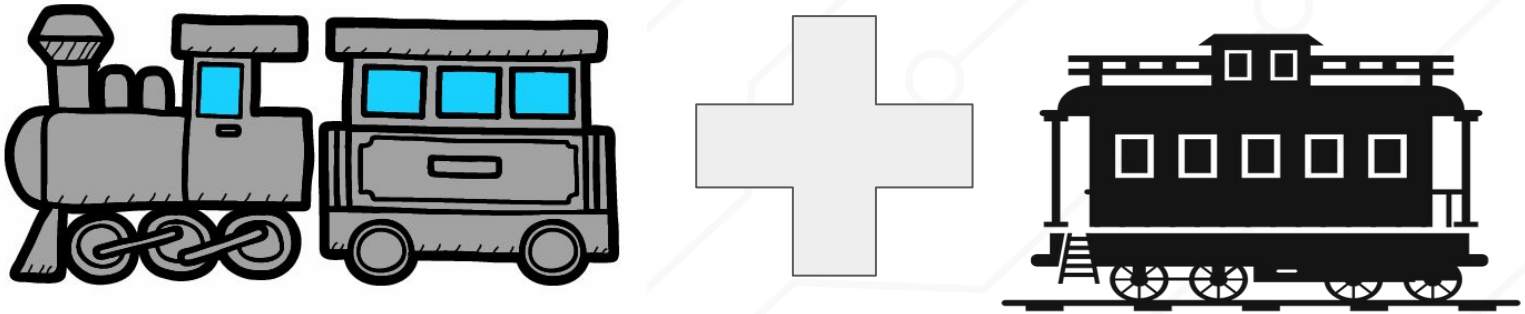
Keep It Simple, Stupid

- ***Open Closed***

Las clases deben estar abiertas para extensión pero cerradas para modificación.

Escribir código que no se deba de cambiar si cambian los requerimientos.

  - Herencia
  - Polimorfísmo



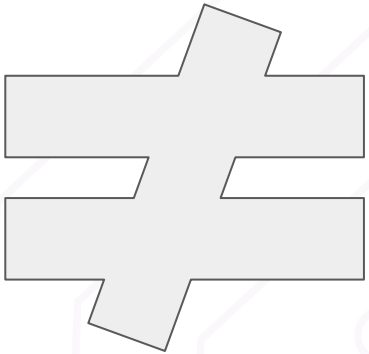
- ***Liskov Substitution***

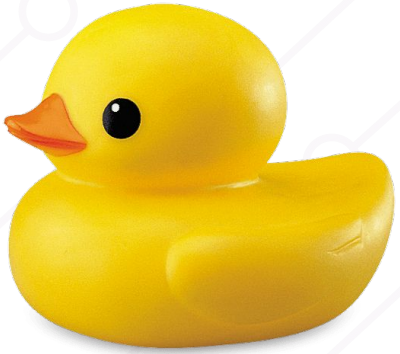
Un objeto debería ser reemplazado con instancias de sus subtipos sin alterar el funcionamiento.

IVolar()

INadar()

ICuack()





IVolar()

INadar()

- ***Interface Segregation***

Debemos usar muchas interfaces pequeñas en vez de una muy grande de propósito general.

No debemos de forzar a una clase a tener métodos que no se necesitan.

Es mejor tener pequeñas clases y especializadas.

Interface Segregation Principle

When more means less

- ***Dependency Inversion***

Los módulos de alto nivel no deben depender de los módulos de bajo nivel.

Ambos deben depender de **abstracciones**.