PROGRAMACIÓN ORIENTADA A OBJETOS

PHP 7

Oscar Fernando Aristizábal Cardona

INTRODUCCIÓN:

- Programación Lineal

- Programación Estructurada

- Programación Orientada a Objetos:

  - Es una filosofía de diseño que utiliza objetos y métodos. Un objeto es una construcción autosustentable que permite la reutilización del código. Un método especifica una operación sin proporcionar ningún detalle para describir cómo debe llevarse a cabo la operación.

VENTAJAS:

- Modularidad:
  - La modularidad se refiere al concepto de hacer múltiples módulos primero y luego vincularlos y combinarlos para formar un sistema completo. La modularidad permite la reutilización y minimiza la duplicación.
- Reusabilidad:
  - El código puede ser reutilizado, sin modificaciones, para realizar un servicio específico independientemente de qué aplicación use el código.
- Ocultar Información:
  - Los detalles de la implementación interna de un módulo (clase) permanecen ocultos del mundo exterior.
- Depuración:
  - Más fácil de solucionar problemas porque el módulo (clase) es independiente de otras piezas de código. La modificación de una pieza de código no afecta a otras piezas de código en la aplicación.

FUNDAMENTOS:

- Clase:
  - Una clase es una plantilla para crear objetos.
  - Una clase es una colección de variables(Atributos) y funciones(Métodos) que trabajan con estas variables.
- Atributo/Propiedad:
  - Las variables de la clase se llaman atributos o propiedades.
  - Se definen utilizando una de las palabras clave public, protected o private.
- Método:
  - Un método es un procedimiento asociado con un objeto (similar a las funciones)
  - Es un fragmento de código que puede tomar alguna entrada en forma de parámetro y realiza algún procesamiento y devuelve un valor.
  - Se puede definir el método utilizando una de las palabras clave public, protected o private.

EJEMPLO (ATRIBUTOS/MÉTODOS):

Pokemon	
–	nombre
–	tipo
–	salud
+	atacar()
+	esquivar()
+	evolucionar()

Atributos

Métodos

VISIBILIDAD (NIVEL ACCESO):

- Private:
  - Este tipo permite el acceso solo a miembros de la misma clase.
  - Utilice este tipo si desea que la variable (Atributo) / función (Método) sea visible solo en su propia clase.
- Protected:
  - Este tipo permite el acceso a miembros de la misma clase e instancias de clases que heredan de la clase padre.
  - Utilice este tipo si desea que la variable (Atributo) / función (Método) sea visible en todas las clases que extienden la clase actual, incluida la clase padre.
- Public:
  - Este tipo permite el acceso desde cualquier lugar.
  - Utilice este tipo si desea que la variable (Atributo) / función (Método) esté disponible desde cualquier lugar, otras clases e instancias del objeto.

OBJETO:

- Una instancia de una clase se llama objeto.
- Objeto se refiere a una instancia particular de una clase donde el objeto puede ser una combinación de variables (Atributos), funciones (Métodos) y estructuras de datos.

EJEMPLO (OBJETOS):

Clase

Jugador
– nombre
– nacionalidad
– posicion
+ correr()
+ disparar()
+ pasar()
+ regatear()
+ saltar()

Cr7	Objeto
– Cristiano Ronaldo	
– Portugal	
– Delantero	
+ correr()	
+ disparar()	
+ pasar()	
+ regatear()	
+ saltar()	

Messi	Objeto
– Lionel Messi	
– Argentino	
– Delantero	
+ correr()	
+ disparar()	
+ pasar()	
+ regatear()	
+ saltar()	

ENCAPSULAMIENTO (OCULTAR DE INFORMACIÓN):

- Ocultar las partes internas del objeto protege su integridad al evitar que los usuarios establezcan los datos en un estado no válido o incoherente.

- Puede utilizar la encapsulamiento si las propiedades de un objeto son privadas y la única forma de actualizarlas es a través de métodos públicos.

EJEMPLO (ENCAPSULAMIENTO):

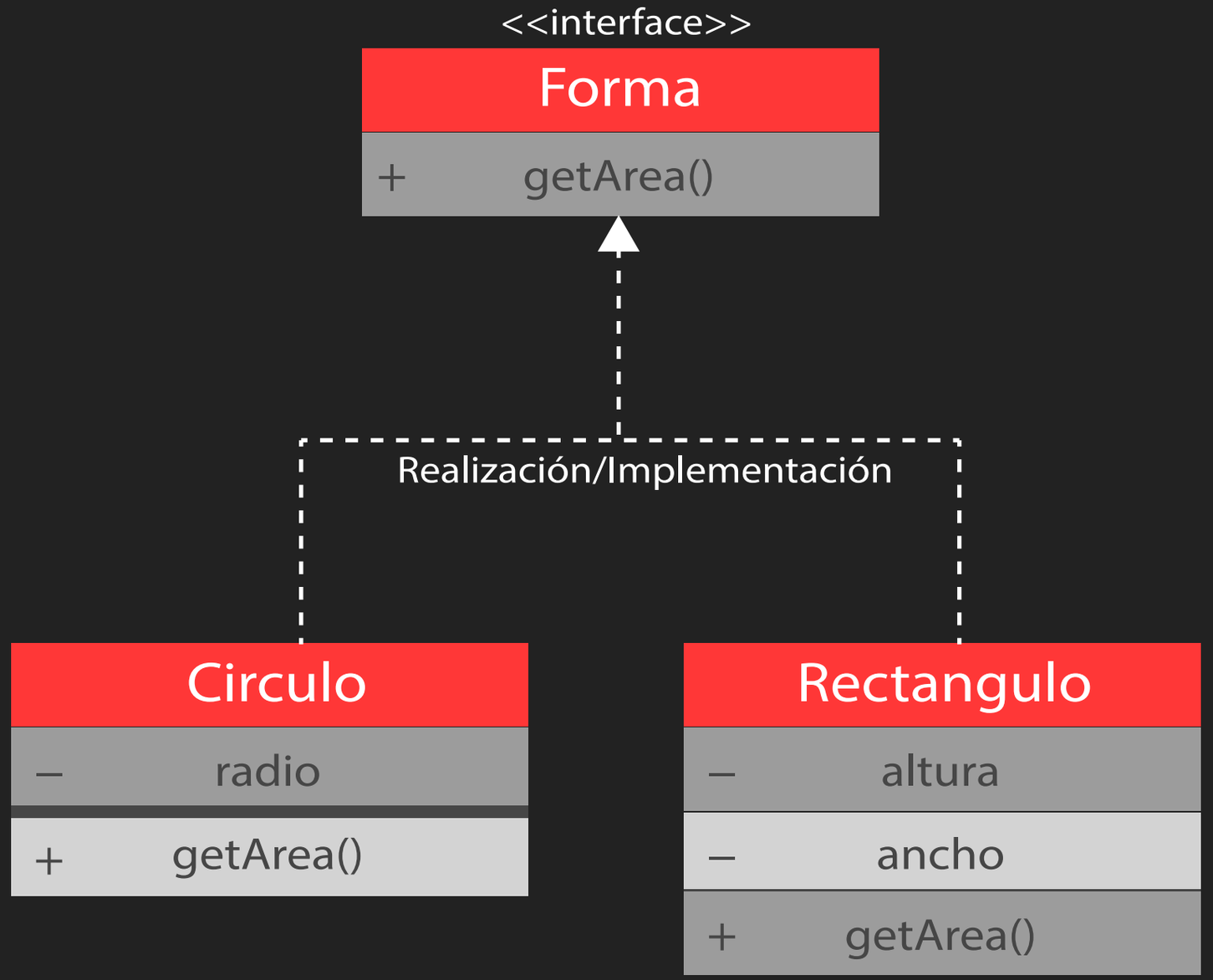
Usuario
– nombre
– genero
+ getNombre()
+ getGenero()
+ setNombre($n)
+ setGenero($g)

POLIMORFISMO:

- Capacidad de procesar objetos de manera diferente. Una parte integral del polimorfismo es la interfaz común.

- El polimorfismo es un patrón de diseño en el que las clases tienen una funcionalidad diferente al compartir una interfaz común.

EJEMPLO (POLIMORFISMO):



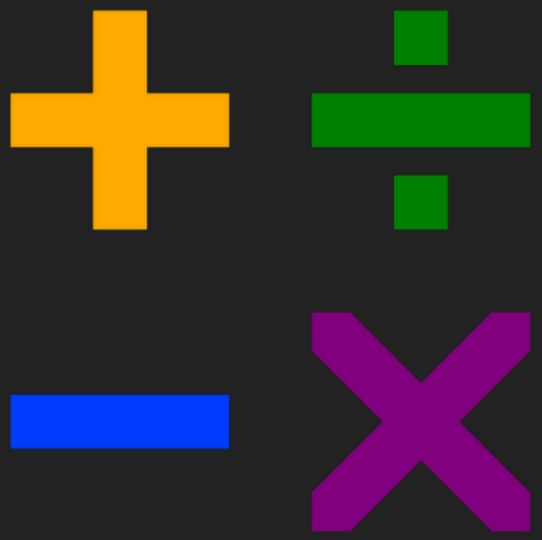
HERENCIA:

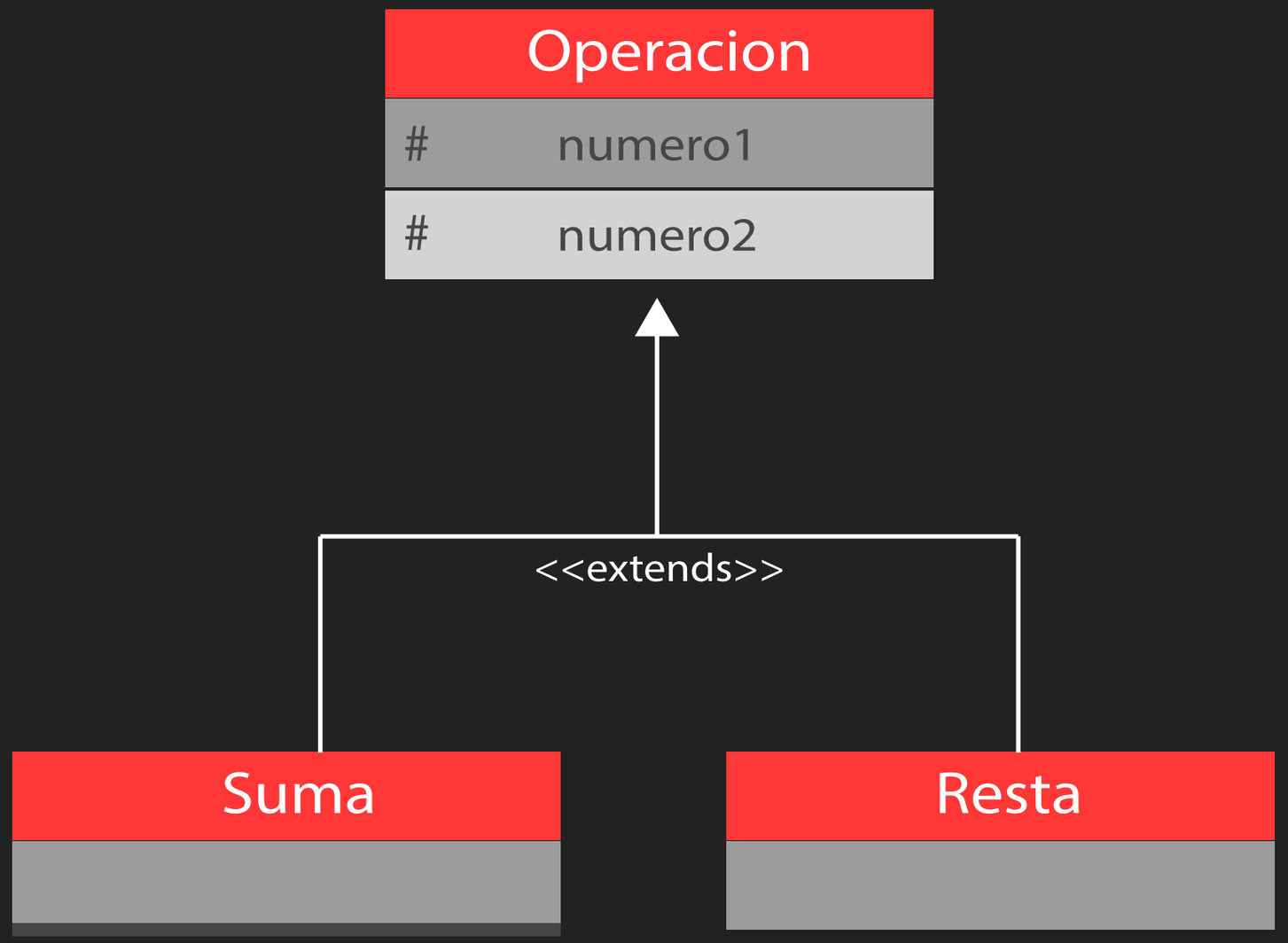
- Tomar propiedades de objetos existentes.

- Un objetivo fundamental de POO es la reutilización y la fiabilidad. No es necesario codificar todo desde cero, puede basar una nueva clase en una existente. Las clases de POO son extensibles a través de la herencia.

- Una clase o subclase secundaria en POO puede heredar todas las características de su clase principal o superclase, adaptar algunas de ellas, anular los métodos y propiedades existentes y agregar nuevas propias.

EJEMPLO (HERENCIA):





CLASE ABSTRACTA:

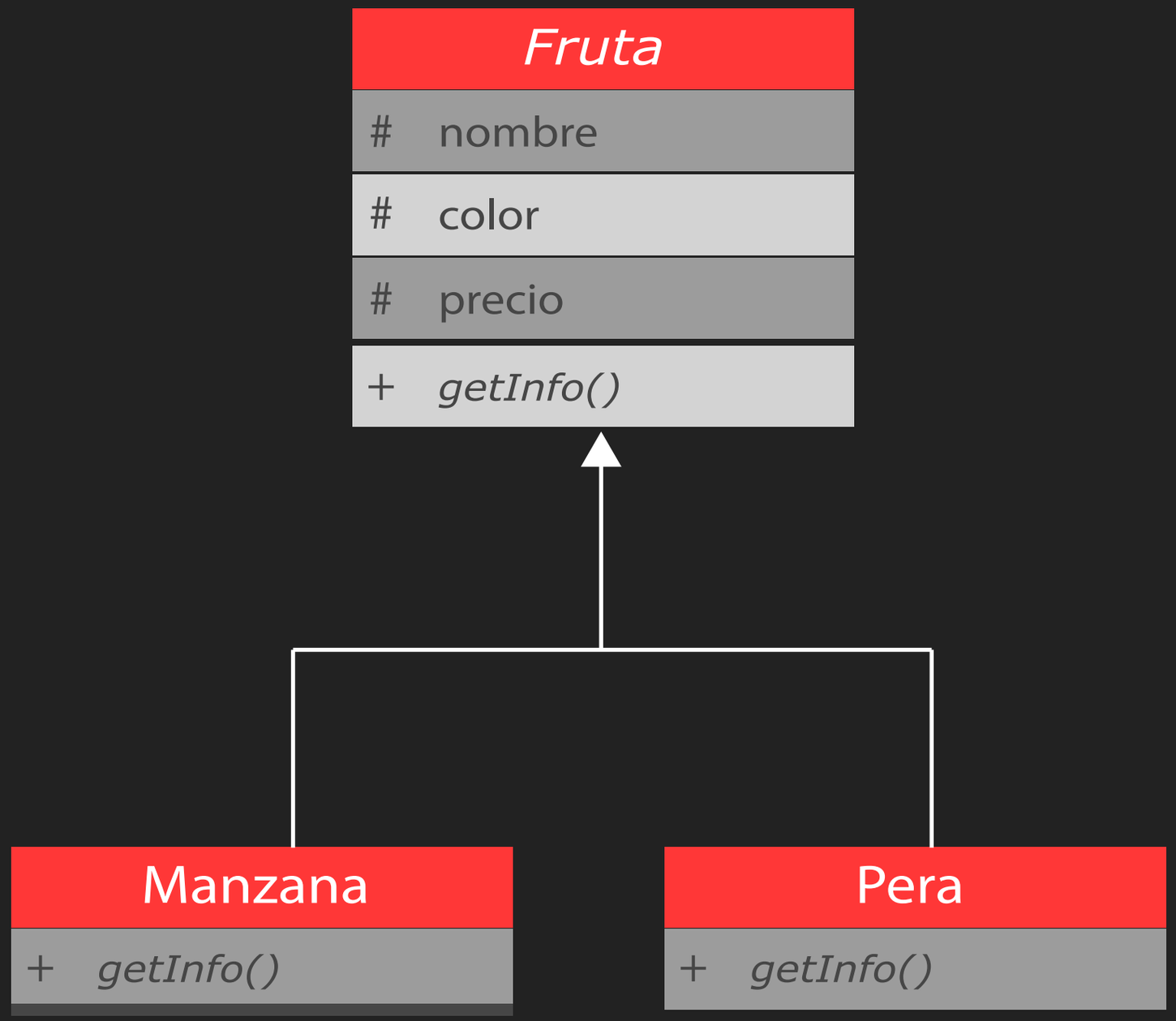
- Las clases abstractas son clases que pueden contener métodos abstractos.

- Un método abstracto es un método que se declara, pero no contiene ninguna implementación.

- Las clases abstractas no se pueden instanciar y requieren subclases para proporcionar implementaciones para los métodos abstractos.

- Los métodos abstractos declarados en la clase padre deben ser definidos por la clase hija.

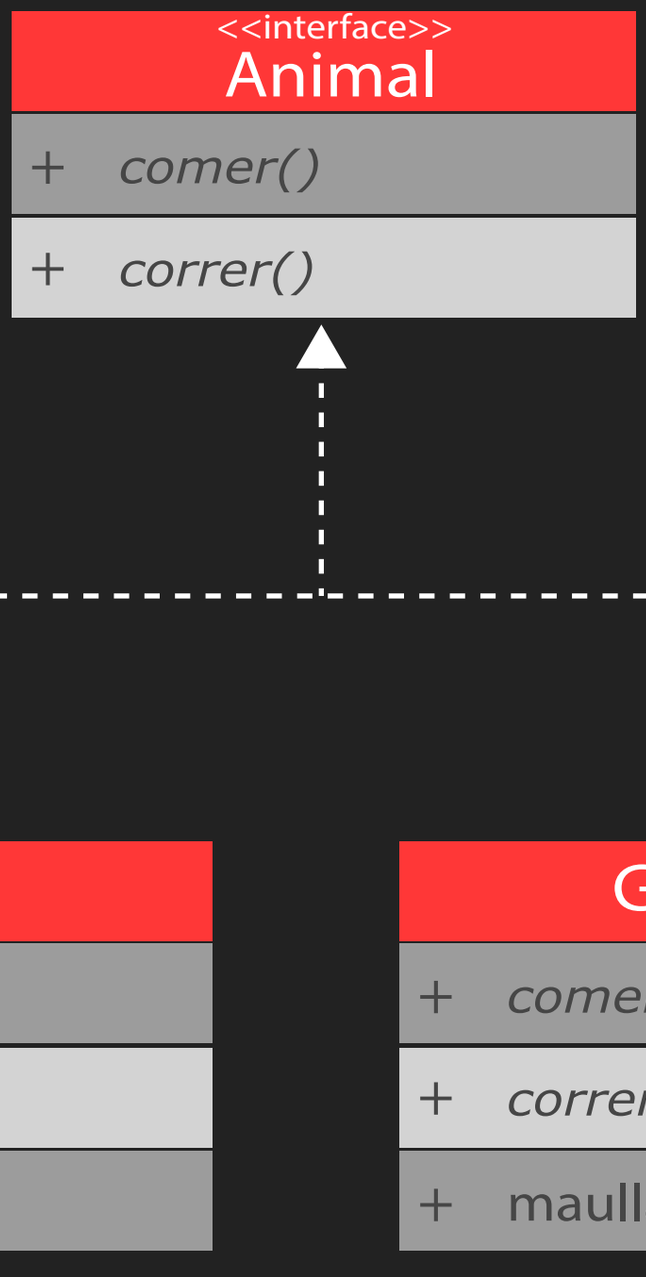
EJEMPLO (CLASE ABSTRACTA / MÉTODO ABSTRACTO):



INTERFACES:

- Es un concepto de abstracción y encapsulamiento.
- Una interfaz es muy similar a una clase abstracta, pero no tiene propiedades y no puede definir cómo se implementarán los métodos.
- Es simplemente una lista de métodos que deben implementarse.

EJEMPLO (INTERFACE):



COMPOSICIÓN Y AGREGACIÓN:

- Relaciones entre objetos.

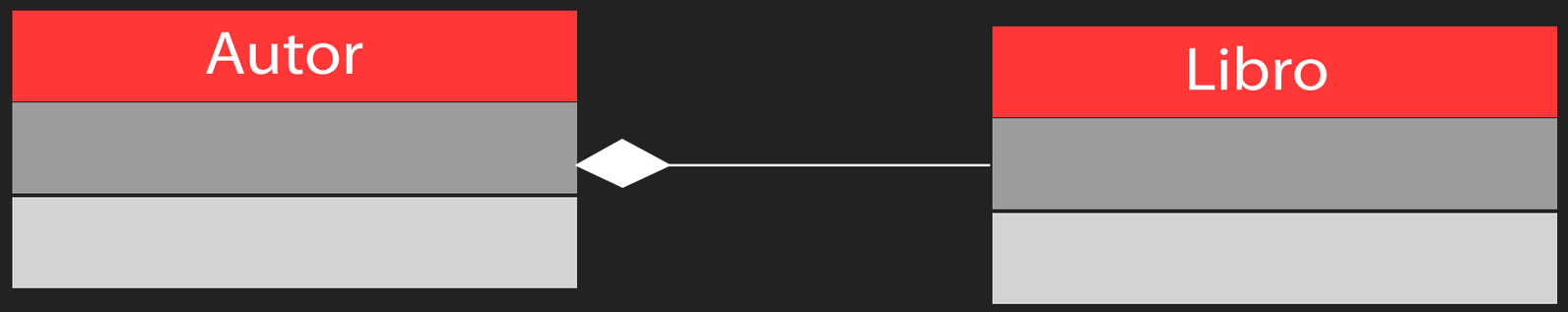
- La composición ocurre cuando un objeto crea otro objeto.

- La agregación ocurre cuando un objeto se compone de múltiples objetos.

- Las relaciones en una composición son requeridas, en la agregación son opcionales.

EJEMPLO (COMPOSICIÓN Y AGREGACIÓN):

Composición



Agregación