



Metodologías de Programación I

Introducción a Objetos



Herencia

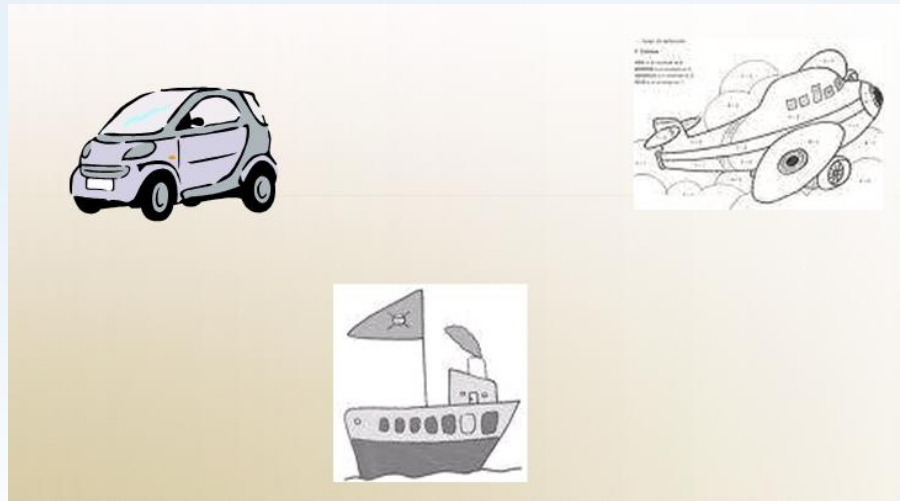
Sobrecarga

Clases Abstractas

Herencia vs Composición

Herencia

Qué características tienen en común?



Son medios de transporte

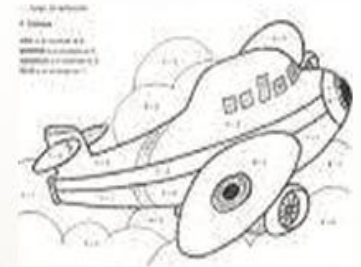
Tienen marca

Color

Modelo

Motor

Herencia

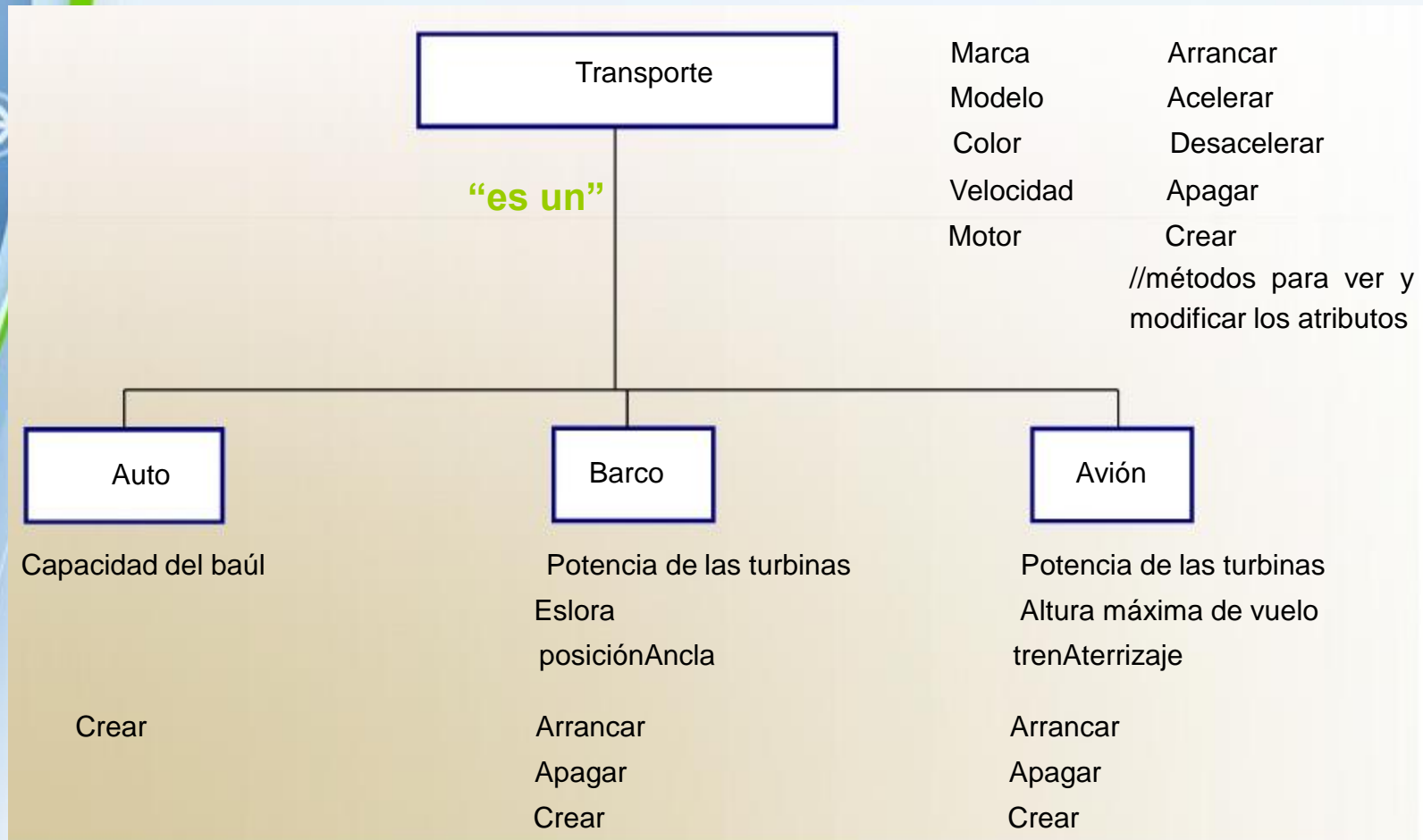


Marca Modelo Color Velocidad Motor	Marca Modelo Color Velocidad Motor	Marca Modelo Color Velocidad Motor
Capacidad del baúl	posiciónAncla Potencia de las turbinas Eslora	trenAterrizaje Potencia de las turbinas Altura máxima de vuelo Tamaño de las alas

Qué se puede notar?


Hay características **comunes** y **propias** de cada uno

Herencia



Cuando ocurren este tipo de cosas aparece el concepto de **HERENCIA**

Herencia



En la POO, la herencia permite que unos objetos puedan basarse en otros existentes.

En términos de clases, la herencia es el mecanismo por el cual una clase X puede heredar propiedades de una clase Y (X hereda de Y) de modo que los objetos de la clase X tengan acceso a los atributos y operaciones de la clase, sin necesidad de redefinirlos.

Herencia

Sin embargo, las propiedades de una clase no son necesariamente la suma de las propiedades de todas sus superclases.

La herencia crea automáticamente una jerarquía de especialización-generalización.

Se suele denominar clase hija/ padre y Subclase / superclase para designar al par de clase que hereda de otra

Herencia

También se suele decir que las subclases son especializaciones de la superclase y que la superclase es generalización de las subclases.

La herencia presenta dos cualidades contradictorias entre si, : “ Una clase hija extiende o amplía el comportamiento de la clase padre, pero también restringe o limita a la clase padre”.

Suele identificarse la herencia mediante la regla “es un” o “es un tipo de ”

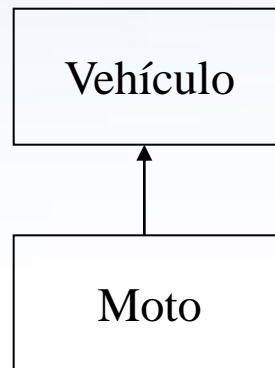
Herencia

- Por ejemplo, la subclase auto, barco y la subclase avión heredan todos los métodos y atributos correspondientes a los transportes, por ser estas subclases de la clase transporte.
- Además, al crear un objeto auto, tendrá no sólo los atributos y comportamiento **propios** de un transporte sino también los **específicos** de un auto, por ejemplo podré conocer la capacidad del baúl.

Herencia

Moto es un Vehículo, luego, la clase

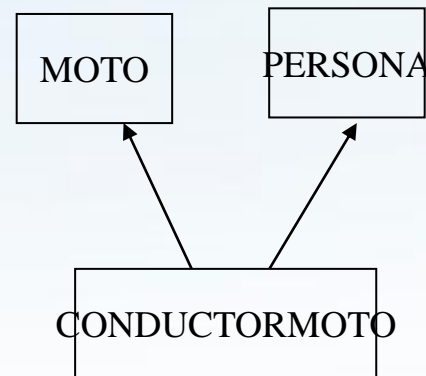
Moto es subclase de la clase Vehículo



Herencia

NO resulta recomendable emplear la herencia si no funciona la regla “X es un Y”, es mas, cuando no pueda justificarse que toda instancia de la clase X es también instancia de Y.

Ej:



Clase ConductorMoto. Que hereda de Moto y Persona, es un mal uso de herencia, pues no toda instancia de ConductorMoto es una instancia de Moto

Herencia

Mas Ejemlos:

Los padres no heredan los rasgos de los hijos ni pueden intercambiarse entre ellos.

Estudiante subclase de Persona.

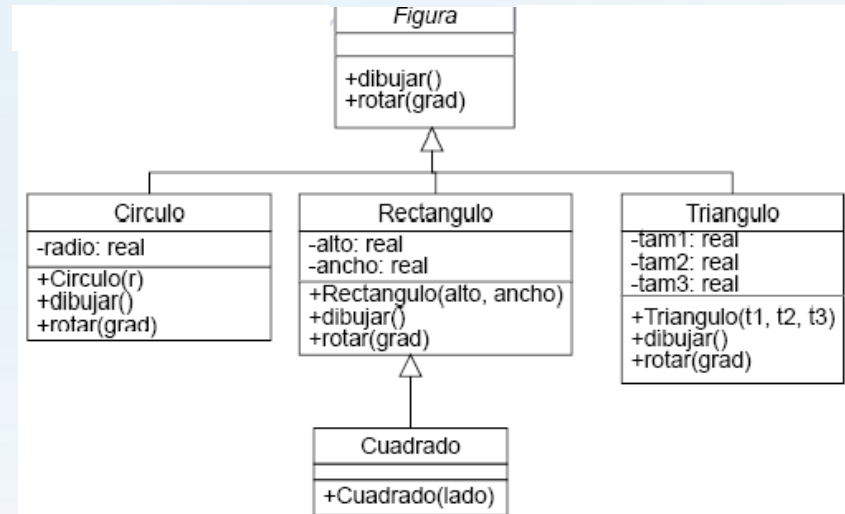
Si en mi implementación utilizo una instancia de Estudiante, luego no puedo reemplazarlo por una instancia de Persona, pues no toda persona es un estudiante.

Consejos Prácticos a la hora de construir Jerarquías

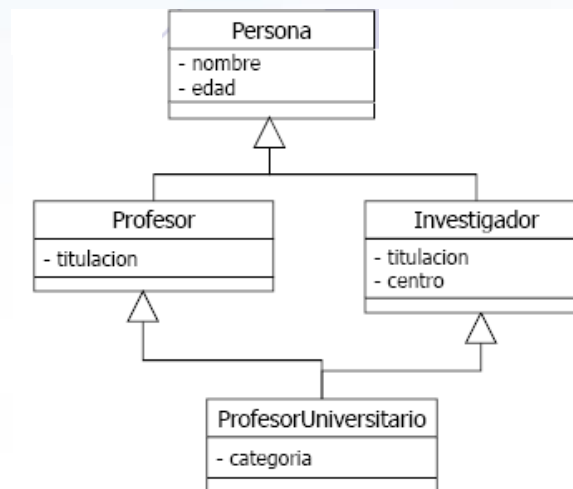
- Construir jerarquías estrechas y profundas. Es decir, con no demasiadas clases en cada nivel de especialización y con muchos niveles.
- Evitar que cualquier clase contenga código para averiguar la clase o el tipo del objeto, la jerarquía bien construida debe ser auténticamente polimorfa

Tipos de Herencia

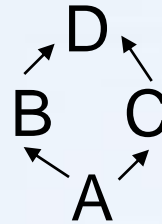
- Simple: Una clase sólo es subclase de una superclase



- Múltiple: una subclase admite más de una superclase.



Problemas de Herencia Múltiple



- Herencia Repetida: A hereda de B y C, que a su vez heredan de D, por lo tanto, la clase A hereda 2 veces de D.
- Conflicto de Nombres: Si A hereda simultáneamente de B y C aparece un conflicto si usan el mismo nombre para algún atributo o método


Herencia

La herencia es la causa de muchos fracasos de sistemas que se hacen con objetos.

La herencia es la herramienta peor utilizada entre todas las herramientas que existen en la implementación de este paradigma.

Puede suceder que existan clases mal modeladas, clases esquizofrénicas que hagan diez mil cosas a la vez (imprimirse, guardarse, mostrarse, dormirse, levantarse, maquillarse, etc.) pero no hay peor cosa que tener un modelo donde se utiliza incorrectamente la herencia, porque la herencia es **“una relación estática entre clases”**.

Herencia

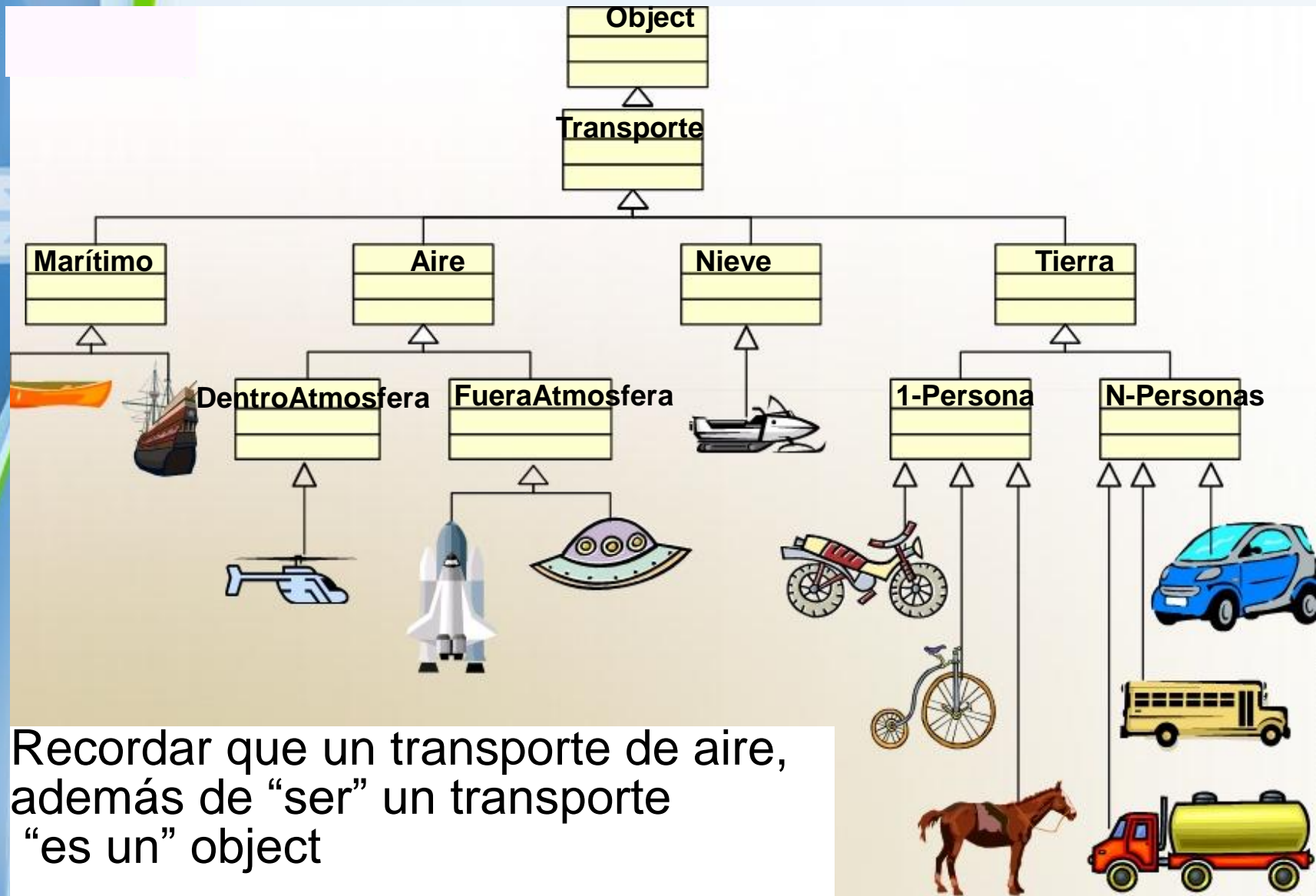


La herencia cumple, al igual que las clases, dos roles. Uno conceptual y otro implementativo. Conceptualmente la herencia modela relaciones “**es un**” entre abstracciones.

Desde el punto de vista implementativo, la herencia permite “**reutilizar**” código.


Es cuando se la utiliza únicamente desde el punto de vista implementativo que se cometen las peores aberraciones que se puedan ver.

Jerarquía



Recordar que un transporte de aire, además de “ser” un transporte “es un” object

Ejemplo



Cómo aplicamos la herencia en nuestro ejemplo de transportes?

Cuántas clases definimos?

Dónde definimos cada atributo?

Dónde definimos cada método?

Dónde implementamos cada método?

Ejemplo

Cuántas clases definimos?

La clase transporte, auto, barco, avión

Dónde definimos cada atributo?

Los atributos comunes a todas las clases en la clase “padre” (Transporte), y los particulares de cada clase en cada una de ellas (Auto, Barco, Avión).


Dónde definimos cada método?

Los métodos comunes en la clase “padre” y los correspondientes a los atributos propios de cada clase en cada una de ellas. Además un método puede definirse en la clase hijo y padre a la vez.

Dónde implementamos cada método?

Depende de que queramos implementar, ya lo veremos

Herencia



Cada vez que estén tentados a heredar de una clase, piensen primero si no se están olvidando de algún otro objeto que debería asumir la responsabilidad que le quieren dar a esta nueva subclase.

HERENCIA - Consideraciones

Cuando un programador define una jerarquía de clases, es porque identifica características comunes en los objetos, y algunas características que los diferencian.

Los atributos comunes se definen en la “super” clase (en nuestro ejemplo la clase transporte).

Los atributos diferentes se definen en cada clase (en nuestro ejemplo la clase auto, barco, avión).

Los métodos que se implementan de igual manera para todas las clases, deben implementarse en la “super” clase.

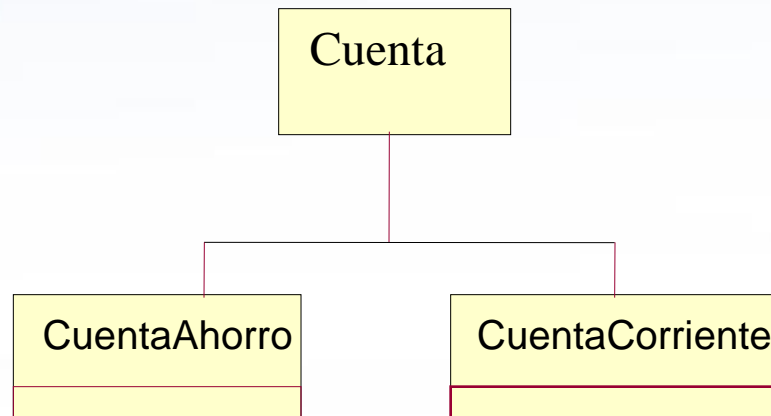
Los métodos que se implementan de manera diferente en cada subclase, deben implementarse en cada una.

Para hacer referencia a un método de una super clase desde una subclase debe ponerse: `super.nombremetodo`.

Clase Abstracta

La herencia permite que existan clases que nunca sean instanciadas directamente.


La ausencia de instancias específicas es su única particularidad, para todo lo demás es como cualquier otra clase



Redefinición

- Si en una clase en particular se invoca a un método, y el método no está definido en la misma, es buscado automáticamente en las clases superiores. Sin embargo, si existieran dos métodos con el mismo nombre y distinto código, uno en la clase y otro en una superclase, se ejecutaría el de la clase, no el de la superclase.
- Por lo general, siempre se puede acceder explícitamente al método de la clase superior mediante una sintaxis diferente, la cual dependerá del [lenguaje de programación](#) empleado.

Ventajas

- 
- Ayuda a los programadores ahorrar código y tiempo.
 - Los objetos pueden ser contruidos a partir de otros similares. Para ello es necesario que exista una clase base y una jerarquía de clases.
 - La clase derivada puede heredar código y datos de la clase base, añadiendo código o modificando lo heredado.
 - Las clases que heredan propiedades de otra clase pueden servir como clase base de otras.

Polimorfismo



Por polimorfismo entendemos aquella cualidad que poseen los objetos para responder de distinto modo ante el mismo mensaje.

Ejemplo: las clases hombre, vaca y perro, si a todos les damos la orden -enviamos el mensaje- Come, cada uno de ellos sabe cómo hacerlo y realizará este comportamiento a su modo.

Polimorfismo



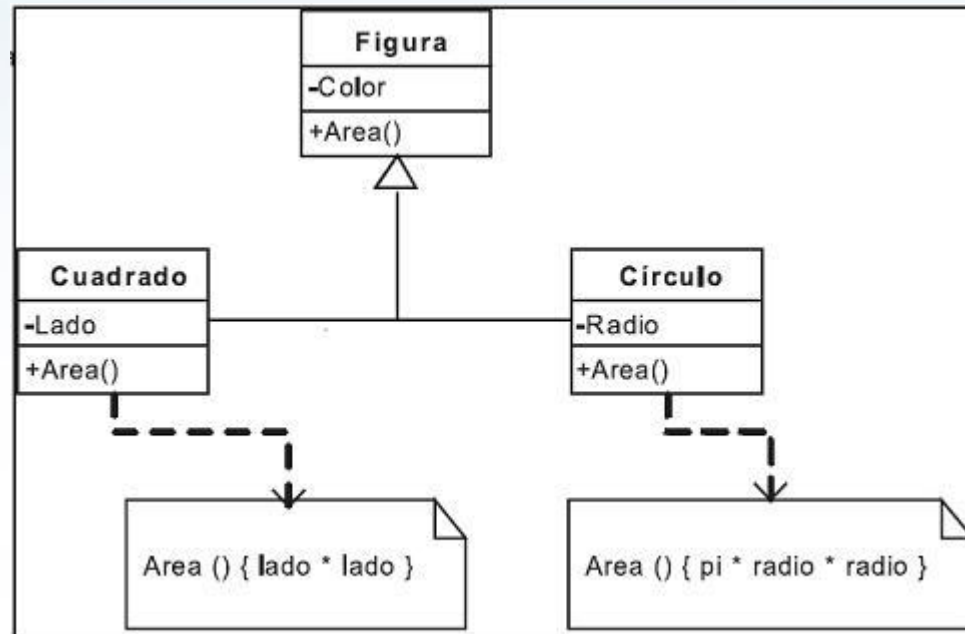
- Los objetos actúan en respuesta a los mensajes que reciben.
- El mismo mensaje puede originar acciones completamente diferentes al ser recibido por diferentes objetos. Este fenómeno se conoce como polimorfismo.
- El polimorfismo, entonces, se refiere a que una misma operación puede tener diferente comportamiento en diferentes objetos.

Polimorfismo


Veamos otro ejemplo algo más ilustrativo.

Si tenemos un Cuadrado y un Circulo podemos decirle a cualquiera de los dos que calcule su área.

Ambas clases sabrán cómo hacerlo porque hemos redefinido para cada uno de ellos el método Area() que no podría estar implementado en la clase padre.



Sobrecarga



La sobrecarga puede ser considerada como un tipo especial de polimorfismo que casi todos los lenguajes de OOP incluyen.

Varios métodos (incluidos los "constructores", de los que se hablará más adelante) pueden tener el mismo nombre siempre y cuando el tipo de parámetros que recibe o el número de ellos sea diferente

Ejemplo

Clase Transporte;

marca: string;
modelo: string
color: string;
velocidad: integer;
miMotor: motor;

metodo crear (unaMarca,unModelo,unColor:string; marcaMotor:string;
valvulasMotor:integer)

metodo crear (unaMarca,unModelo,unColor:string)

metodo crear (unaMarca,unModelo, marcaMotor:string)



Proxima Clase

Metodos concretos, abstractos.

Metodo Template

Instancias. Constructores. Pseudovariables