

Herencia

Sitio: Agencia de Habilidades para el Futuro
Curso: Desarrollo de Sistemas Orientado a Objetos 1º D
Libro: Herencia

Imprimido por: Eduardo Moreno
Día: lunes, 5 de mayo de 2025, 11:09

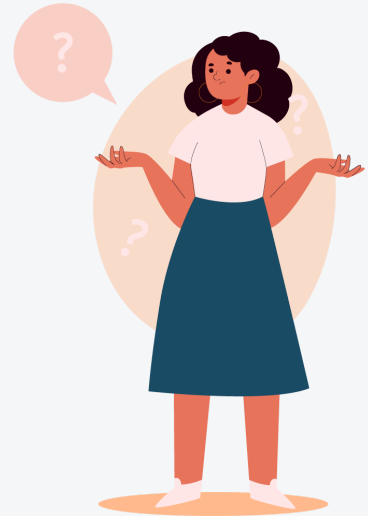
Tabla de contenidos

- 1. Preguntas orientadoras
- 2. Introducción
- 3. Un nuevo pilar: La Herencia
 - 3.1. ¿Cuándo se usa la herencia?
 - 3.2. Más ejemplos de herencia
- 4. Generalización y especialización
- 5. Jerarquía de clases
- 6. Tipos de herencia
- 7. En resumen



Preguntas orientadoras

- Las clases animal, perro, gato, caballo comparten las mismas propiedades... ¿no hay manera de agruparlas o hacer algo para que no se repitan las mismas?
- Y si le agregamos la propiedad color... ¿lo tenemos que hacer en todas las clases?
- ¿Cómo hacemos para que esto no suceda?





Muchas veces nos damos cuenta que escribimos clases que tienen cosas en común, propiedades o comportamiento.

Lo más fácil es copiar y pegar, pero el mantenimiento de esas clases comienza a ser tedioso y no tiene ningún sentido.

En la Programación Orientada a Objetos (POO) existe la [herencia](#).

La herencia en la programación orientada a objetos (POO) sirve para varios propósitos importantes:

1. **Reutilización de código:** La herencia permite aprovechar y reutilizar el código existente. Una clase hija puede heredar propiedades y métodos de una clase padre, lo que evita tener que volver a escribir el mismo código en múltiples lugares. Esto ahorra tiempo de desarrollo y ayuda a mantener un código más limpio y organizado.
2. **Abstracción y generalización:** La herencia permite crear jerarquías de clases que representan relaciones de "es un". Por ejemplo, puedes tener una clase "Animal" como clase padre y clases hijas como "Perro", "Gato" y "Pájaro". Esto permite modelar de manera más precisa y comprensible las relaciones entre los objetos del mundo real.
3. **Polimorfismo:** Otro de los pilares de la POO que veremos más adelante.
4. **Especialización:** La herencia permite crear clases más específicas y especializadas a medida que se descende en la jerarquía de clases. Las clases hijas pueden agregar características adicionales o modificar el comportamiento heredado de la clase padre para adaptarse a necesidades particulares. Esto permite una mayor modularidad y flexibilidad en el diseño del sistema.

La herencia en la POO sirve para reutilizar código, modelar relaciones entre objetos, facilitar el polimorfismo y permitir la especialización de clases. Es una herramienta poderosa que ayuda a crear estructuras de código más eficientes, mantenibles y escalables.

En este libro, vamos a aprender cómo utilizarla y cómo darnos cuenta cuando es necesaria.

[¡Comencemos!](#)



Un nuevo pilar: La Herencia

¡Recuperemos lo visto! Hasta ahora hemos trabajado con 2 pilares de la Programación Orientada a Objetos:

- La abstracción
- El encapsulamiento

¡Llegó el momento del tercer pilar: La herencia!

Hacia una conceptualización

La herencia en POO tiene un significado parecido al que conocemos respecto a heredar rasgos o características de nuestros padres.

¿A qué nos referimos específicamente? Podemos crear nuevas "clases" basadas en otras clases existentes. En la herencia, una clase (llamada clase derivada o subclase) hereda las propiedades y métodos de otra clase (llamada clase base o superclase).

La herencia se basa en el principio de "es un" o "es una". Siempre que hagamos una herencia tiene que poder responder a ese principio.



¡Veamos un ejemplo concreto!

Por ejemplo, si tienes una clase base llamada "**Animal**" y quieres crear una clase derivada llamada "Perro", puedes utilizar la herencia para decir que "**Perro**" es un "**Animal**", con lo cual Perro puede ser una subclase de Animal. En este caso, la clase "**Perro**" heredaría las propiedades y métodos de la clase "**Animal**" (como "nombre", "edad" y "hacerSonido"), pero también podría agregar sus propias propiedades y métodos específicos de los perros como por ejemplo "dar la pata".

Funcionalidad y uso



La herencia permite reutilizar código y promueve la organización y estructura en el diseño de programas. Además, proporciona la capacidad de extender y modificar el comportamiento de las clases base sin necesidad de modificar su implementación original. Esto es posible mediante la adición de nuevos métodos o la modificación de los existentes en la clase derivada.

¿Cuál es la ventaja de su uso? La herencia ofrece una ventaja importante, permite la reutilización del código. Una vez que una clase ha sido depurada y probada, el código fuente de dicha clase no necesita modificarse. Su funcionalidad se puede cambiar derivando una nueva clase que herede la funcionalidad de la clase base y le añada otros comportamientos. Reutilizando el código existente se ahorra tiempo y esfuerzo, ya que solamente tiene que verificar la nueva funcionalidad que proporciona la clase derivada y no todo.



¿Cuándo se usa la herencia?

En general se usa la herencia en POO:

| | |
|---|--|
|  | Cuando se dan cuenta que diversos tipos tienen algo en común, por ejemplo en el juego del ajedrez peones, alfiles, rey, reina, caballos y torres, son piezas del juego. Creamos, por tanto, una clase base (pieza) y derivamos cada pieza individual a partir de dicha clase base. |
|  | Cuando se precisa ampliar la funcionalidad de un programa sin tener que modificar el código existente. |

Esto permite crear una estructura jerárquica de clases cada vez más especializada. La gran ventaja es que ya no tenemos que empezar desde cero cuando se desea especializar una clase ya existente. Como resultado, se pueden usar bibliotecas de clases que aportan una base que puede especializarse según la necesidad.

La clase superior de la jerarquía, en cada relación, se denomina superclase, clase base o clase padre y, la clase que hereda de la superclase, se denomina subclase, clase derivada o clase hija.



Más ejemplos de herencia

Veamos algunos ejemplos de herencia:



Ejemplo 1

Clase **Vehículo** y subclases **Coche** y **Motocicleta**

La clase "Vehículo" puede tener propiedades y métodos generales como "marca", "modelo" y "arrancar".

La subclase "Coche" puede heredar esas propiedades y métodos de "Vehículo" y también puede tener propiedades y métodos específicos de los coches, como "número de puertas" y "abrir baúl".

La subclase "Motocicleta" también hereda de "Vehículo" y puede tener propiedades y métodos específicos de las motocicletas, como "cilindrada" y "hacer un caballito o wheelie".



Ejemplo 2

Clase **Figura** y subclases **Rectángulo** y **Círculo**

La clase "Figura" puede tener métodos generales como "calcularÁrea" y "calcularPerímetro".

La subclase "Rectángulo" hereda de "Figura" y puede tener propiedades y métodos específicos de los rectángulos, como "largo" y "ancho".

La subclase "Círculo" también hereda de "Figura" y puede tener propiedades y métodos específicos de los círculos, como "radio" y "calcularDiámetro".



Ejemplo 3

Clase **Empleado** y subclases **Gerente** y **Desarrollador**

La clase "Empleado" puede tener propiedades y métodos generales como "nombre", "salario" y "trabajar".

La subclase "Gerente" hereda de "Empleado" y puede tener propiedades y métodos específicos de los gerentes, como "departamento" y "realizarEvaluaciones".

La subclase "Desarrollador" también hereda de "Empleado" y puede tener propiedades y métodos específicos de los desarrolladores, como "lenguaje de programación" y "escribirCódigo".

Estos ejemplos ilustran cómo la herencia nos permite crear jerarquías de clases, donde las subclases heredan características y comportamientos de las clases base, al tiempo que agregan sus propias características únicas. Esto nos permite organizar y estructurar nuestro código de manera más eficiente y reutilizable.



Generalización y especialización

¿Por qué presentamos este tercer pilar? Con el pilar de la herencia se consigue clasificar los tipos de datos por tipo o por variedad y acerca un poco más el mundo de la programación con el modo de razonar.

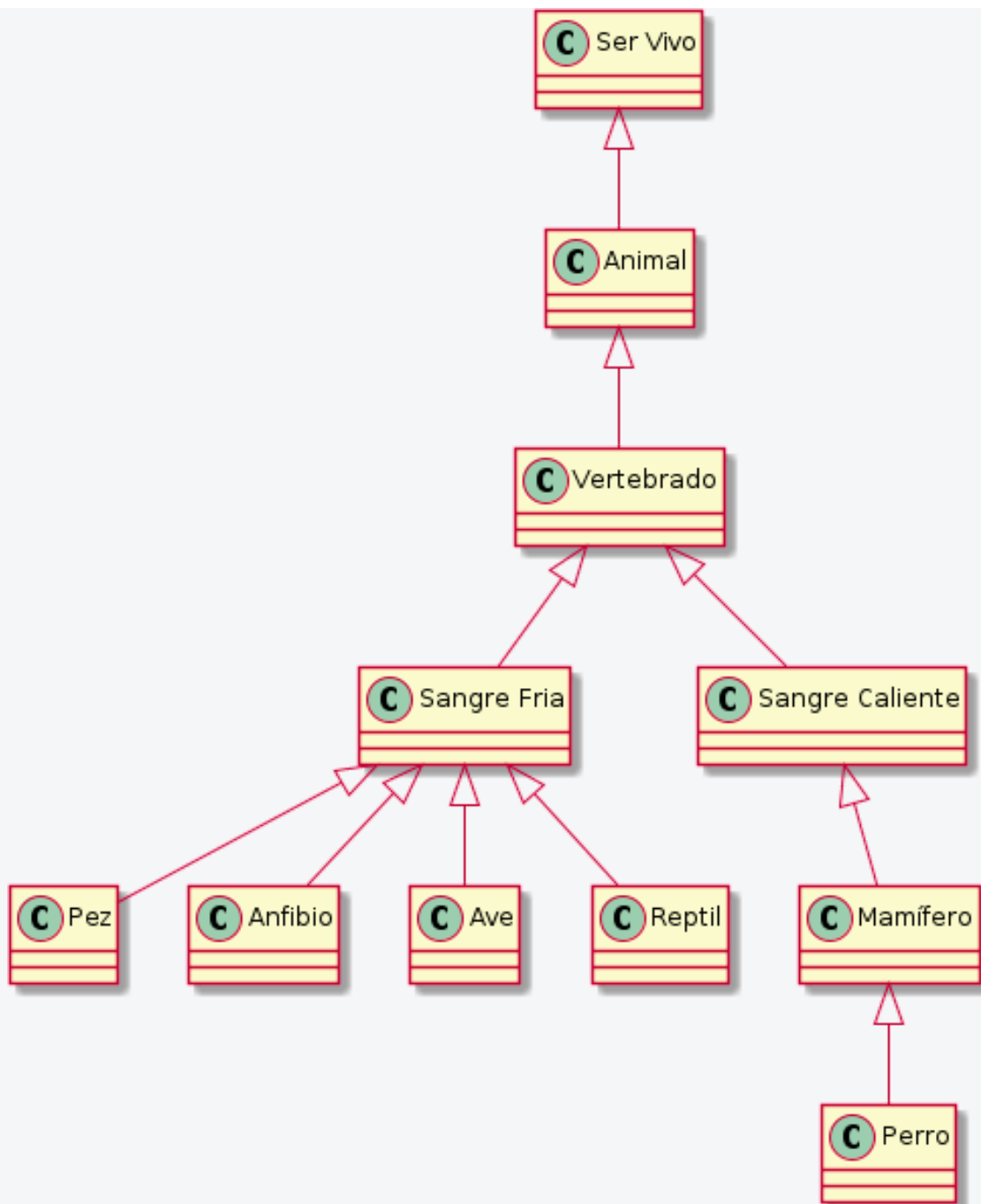
La generalización es una relación entre clases, que determina que la interfaz de la subclase debe incluir todas las propiedades públicas y privadas de la superclase. Gracias a la herencia es posible especializar o extender la funcionalidad de una clase, derivando de ella nuevas clases.

La herencia es siempre transitiva: una clase puede heredar características de superclases que se encuentran muchos niveles más arriba en la jerarquía de herencia.



¡Veamos un ejemplo concreto!

Si la clase **Perro** es una subclase de la clase **Mamífero**, y la clase **Mamífero** es una subclase de **Sangre Caliente**, que a su vez es una subclase de **vertebrado**, entonces el Perro heredará todos los atributos tanto de Mamífero como de "Ser vivo" incluyendo todas los atributos y métodos de las clases intermedias.



Test "Es-un"

Una técnica para determinar si una clase se relaciona con otra mediante herencia es preguntar si la clase **A ES-UN B**. Si la frase tiene sentido, es verdadera y representa la realidad del problema en que se está trabajando, entonces la situación de herencia es la más probable para ese caso.

| Relacionada con herencia | NO usar herencia |
|-------------------------------------|--------------------------------------|
| Un pájaro es un animal. | Un pájaro es un mamífero. |
| Un gato es un mamífero. | Un caracol es un pez. |
| Un pastel de manzana es un pastel | Un pastel de manzana es una manzana. |
| Una matriz de enteros es una matriz | Una matriz de enteros es un entero. |
| Un coche es un vehículo. | Un motor es un vehículo. |

A continuación, trabajemos estas relaciones representándolo en la jerarquía de clases.



Jerarquía de clases

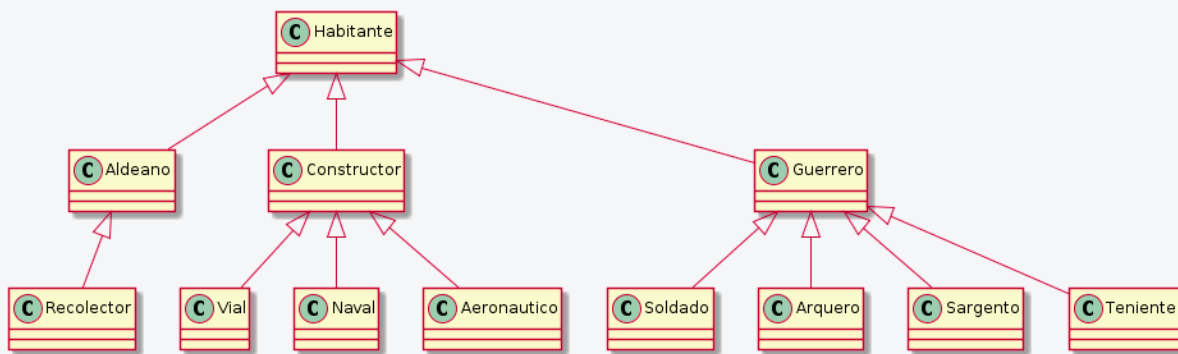
La relación primaria-secundaria entre clases se puede representar desde un punto de vista jerárquico, denominado **vista de clases en árbol**. Esta vista en árbol empieza con una clase general llamada **superclase** (también se le dice clase primaria, clase padre, clase principal o clase madre). Las **clases derivadas** (clases secundarias o subclases) se vuelven cada vez más especializadas a medida que van descendiendo del árbol.



Veamos un ejemplo

Personajes de un juego

En un juego para computadora podría pensarse en que existen "personajes" que tienen diferentes roles, propiedades, capacidades y que pueden hacer o no hacer determinadas acciones. Un constructor vial puede construir un camino que une ciudades pero no puede recolectar minerales. Un guerrero soldado puede "pelear" con una espada pero no puede "disparar flechas". Aunque cada personaje tiene un lugar en el mundo del juego, una cantidad de "vida" y de "energía" que los diferencian de los demás, todos pueden moverse, recibir "órdenes", alertar si aparece un "enemigo", etc. Así, mientras se juega podría haber cientos de personajes corriendo por todo el mundo del juego pero todos ellos son unos pocos.



Como podemos observar, queda así representada la vista de clases en árbol, comenzando por la clase general o superclase y las clases derivadas que se vuelven cada vez más especializadas a medida que van descendiendo del árbol.

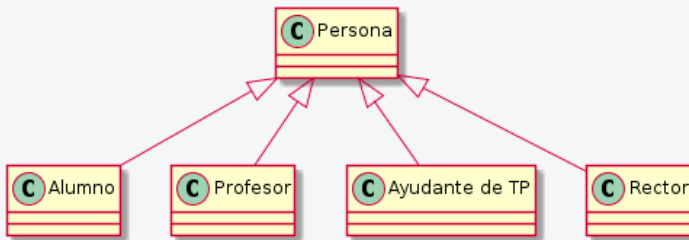
Ahora, necesitamos clasificar los tipos de herencia.



Tipos de herencia

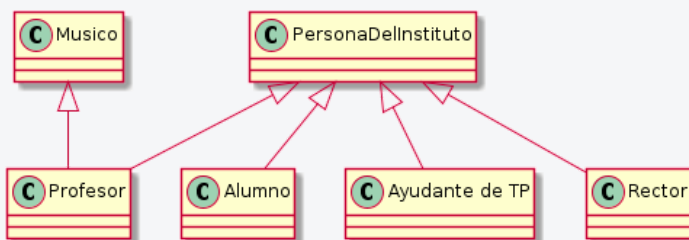
Existen varios tipos de herencia en la POO, cada uno con características y usos específicos de acuerdo a nuestras necesidades. A continuación, se presentan algunos de los tipos más comunes:

- **Herencia simple:** cuando una clase hereda los atributos y métodos de una sola clase.



Alumno, Profesor, Ayudante y Rector heredan las características de Persona.

- **Herencia múltiple:** cuando una clase hereda los atributos y métodos de varias clases inmediatas a la vez. Este tipo de herencia suele introducir complejidad y ambigüedad y si bien algunos lenguajes de programación implementan técnicas distintas para resolverlo, otros como C# y Java lo han descartado y no soportan esta técnica.



Alumno, Profesor, Ayudante y Rector heredan las características de Persona. Pero además, Profesor hereda características de Musico, cosa que ninguna otra clase lo hace. Es un caso hipotético, el Profesor podría ejecutar algún método que toque algún instrumento mientras que las demás clases no podrán.



En resumen

Como vimos, la herencia es un concepto fundamental en la POO que permite crear jerarquías de clases y establecer relaciones de "es un" o "es una".

A través de la herencia, una clase hija puede heredar propiedades y comportamientos de una clase padre, lo que promueve la reutilización de código y facilita la organización y modelado de objetos del mundo real esto nos permite aprovechar y reutilizar el código existente, evitando la duplicación de código y promoviendo la modularidad y el mantenimiento eficiente, pudiéndose crear jerarquías de clases que representan relaciones de "es un/a".

Las clases más generales y abstractas pueden ser la base para clases más específicas y concretas. Esto permite una mejor representación de las relaciones del mundo real y una mayor comprensión del sistema.

Además, las clases hijas pueden agregar nuevas características al heredado de la clase padre permitiendo la especialización de clases y la adaptación a necesidades específicas.

Al comprender y utilizar adecuadamente la herencia, puedes escribir un código más limpio, modular y eficiente, así como crear sistemas más flexibles y escalables.

Recuerda que **la herencia es solo uno de los conceptos fundamentales de la POO**, existen otros conceptos como la **encapsulación**, el **polimorfismo** y la **abstracción** que también son importantes en el desarrollo de aplicaciones orientadas a objetos.