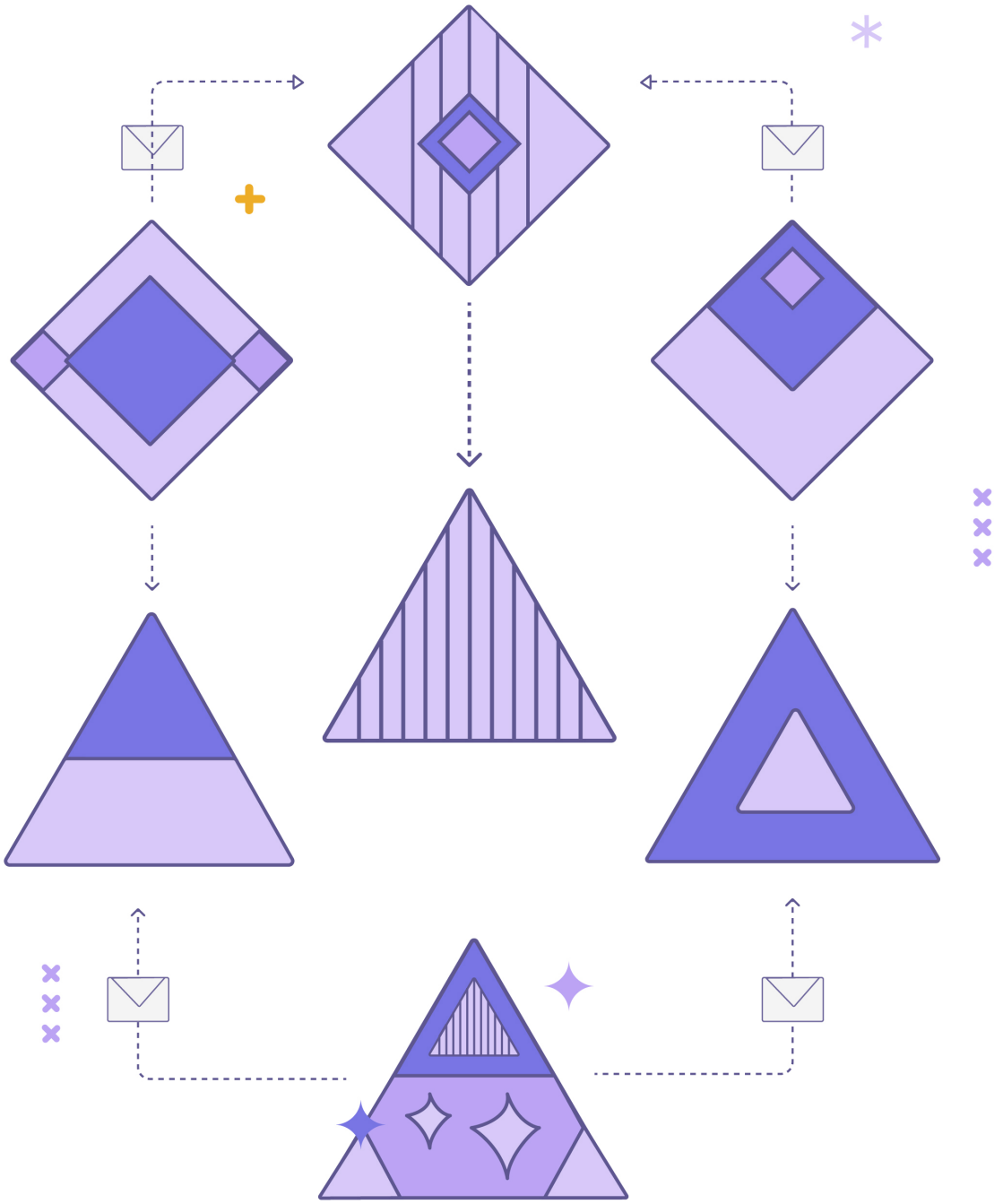
# Capítulo 10

## Herencia y polimorfismo



# Capítulo 10

## Herencia y polimorfismo

## # Objetivo

En este capítulo exploramos en profundidad dos pilares fundamentales de la programación orientada a objetos, la herencia y el polimorfismo. Aprenderemos a construir jerarquías de clases, aprovechar la reutilización de código y aplicar conceptos avanzados como el polimorfismo para crear sistemas flexibles, mantenibles y escalables.

Al finalizar este capítulo, estarás capacitado para utilizar la herencia y el polimorfismo de forma eficaz en tus soluciones. Habrás adquirido una comprensión profunda de lo que implica la herencia, como así también sobre la reutilización de código. Además, estarás en condiciones de diseñar e implementar jerarquías de clases bien definidas, aprovechar el polimorfismo para escribir código más genérico y flexible, y crear sistemas mantenibles y escalables a través del uso de interfaces.

## # Herencia

Es un concepto fundamental de la Programación Orientada a Objetos que permite crear nuevas clases a partir de otras existentes, tomando como base sus características y comportamiento.

La herencia es la capacidad de una clase de heredar propiedades y métodos de una clase padre, lo que permite reutilizar código y hacer que las clases sean más fáciles de entender y mantener.

Interpretemos este concepto a través de la analogía que puede darse en la vida entre un padre y un hijo.

Imaginemos a Eugenio, un hombre de 35 años con pelo castaño y ojos marrones, y a su hijo Julián, un niño de 10 años con características físicas muy similares, pelo castaño y ojos marrones.

Julián ha heredado de Eugenio muchas características, tanto físicas como de personalidad y habilidades. Físicamente, ambos comparten el color de pelo y el color de sus ojos, además de tener una altura similar. En cuanto a la personalidad, Julián ha heredado la inteligencia, el sentido del humor y la amabilidad de su padre. Incluso, ha desarrollado

algunas de las habilidades de Eugenio, como la habilidad para tocar la guitarra y el gusto por la lectura.

Sin embargo, Julián también tiene características únicas que lo diferencian de su padre. Su color de piel, por ejemplo, es más claro que el de su padre. Además, tiene intereses distintos, como el gusto por jugar al fútbol, mientras que Eugenio prefiere jugar al squash. Finalmente, Julián ha desarrollado una habilidad que Eugenio no posee: la habilidad para patinar.

Esta analogía, que ilustra cómo funciona la herencia en la vida real, también aplica al mecanismo que el paradigma orientado a objetos nos provee. Una clase hija, como Julián, hereda las características de una clase padre, como Eugenio. De esta manera, la clase hija puede reutilizar el código de la clase padre, a la vez que tiene sus propias características únicas.

La herencia puede ser simple o múltiple, esto significa poder heredar de una clase o más de una. Java sólo permite a las clases la herencia simple, o sea, heredar de una sóla clase. La herencia múltiple en java queda sólo para las interfaces, un concepto que luego desarrollaremos.

## [-] subclases y superclases

Las clases se organizan en una jerarquía, donde las del nivel superior se denominan "clases base" o "superclases", y las del nivel inferior se denominan "clases derivadas", "subclases" o incluso “clases hijas”.

La herencia permite a una subclase heredar los atributos y métodos de su superclase, permitiendo además añadir nuevos atributos y métodos que por supuesto serán propios.

El siguiente gráfico muestra la clase base -o superclase- Animal y sus clases derivadas -o subclases- Perro y Gato.

Como puede apreciarse en la figura, la clase Animal tiene atributos propios como el “nombre” y la “edad”. Las clases derivadas -clases hijas- tendrán estos mismos atributos aunque no se hayan especificado.

Lo mismo ocurre con los métodos, tanto la clase Perro como la clase Gato tendrán de forma implícita los métodos “comer()” y “dormir()” definidos en la superclase Animal.

Que las clases tengan los métodos implícitos significa que los objetos de estas clases derivadas podrán enviar estos mensajes. Del mismo modo, los métodos propios de las clases derivadas podrían utilizar los atributos de la clase padre.

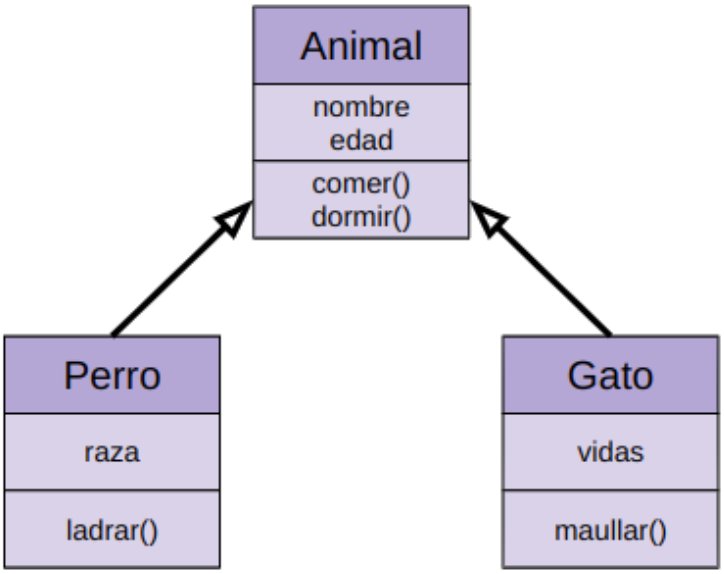



*Fig.42 - jerarquía de clases.*

Se puede llamar superclase a una clase siempre y cuando exista al menos una clase que herede de ella. Por lo tanto, la clase Animal es la superclase de Perro o de Gato, y estas no son superclase de nadie.

> La relación "es un" se utiliza para indicar una relación de tipo entre una clase hija y una clase padre.

Veamos como se especifican estas características en el código.

La clase Animal tendrá la siguiente implementación:

```
public class Animal {

   //
   private String nombre;
```

```java
//
 private int edad;

 //
 public Animal(String nombre, int edad) {
     this.nombre = nombre;
     this.edad = edad;
 }

 //
 public int getEdad() {
     return edad;
 }

 //
 public void setEdad(int edad) {
     this.edad = edad;
 }

 //
 public String getNombre() {
     return nombre;
 }

 //
 public void setNombre(String nombre) {
     this.nombre = nombre;
 }

 //
 public void comer(){
     System.out.println("comiendo...");
 }
```