Herencia

Contenido

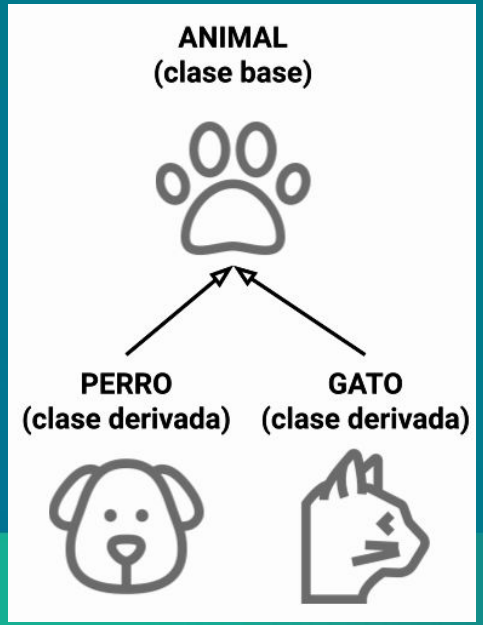
Herencia

- ¿Qué es la herencia?
- Tipos de herencia.
- ¿Qué se hereda?
- Modificadores de acceso.
- Constructores.
- Principio de sustitución.

¿Qué es la herencia?

Se trata de una **relación** entre una o más clases en las que **se comparten atributos y métodos** definidos en otra clase.

La clase *base* es una **generalización** de un grupo de características y comportamiento que tienen en común las clases *derivadas*, mientras que las últimas son una **especialización** de la clase base.

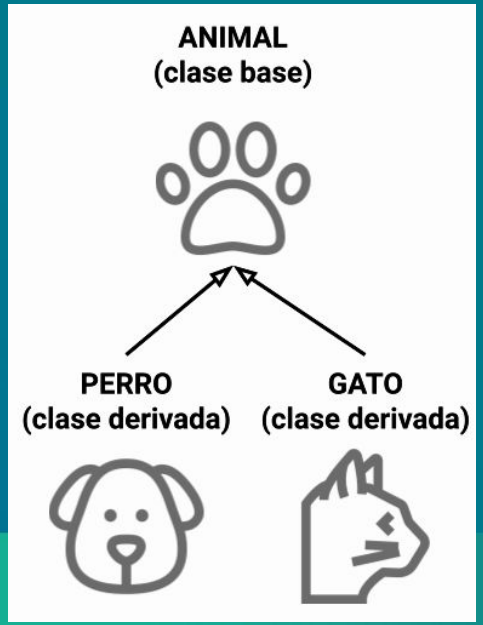


¿Qué es la herencia?

El propósito principal de la herencia es el de organizar mejor las clases que componen una determinada realidad, y poder agruparlas en función de atributos y comportamientos comunes.

La herencia permite crear nuevas clases a partir de otras ya existentes (en lugar de crearlas partiendo de cero).

Se puede decir que una clase derivada es un (is -a) tipo específico de clase base.



Tipos de herencia

Herencia Simple:

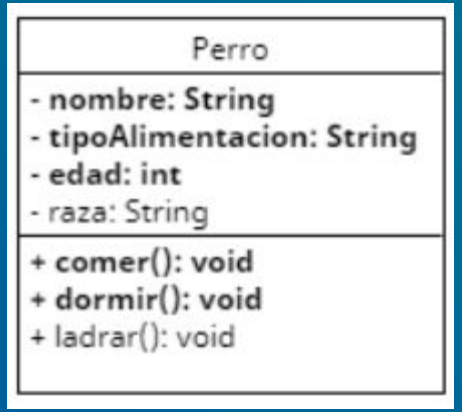
- Una clase derivada puede heredar sólo de una clase base (Java soporta este tipo de herencia).

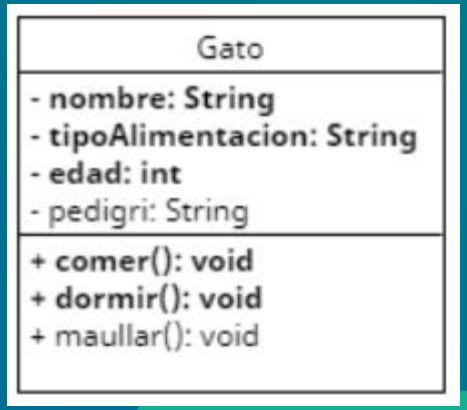
Herencia Múltiple:

- Una clase derivada puede heredar de una o más clases base (C++ es un ejemplo de lenguaje que soporta este tipo de herencia).

Ejemplo de herencia

Estas clases representan a distintas entidades, pero poseen muchos miembros en común.

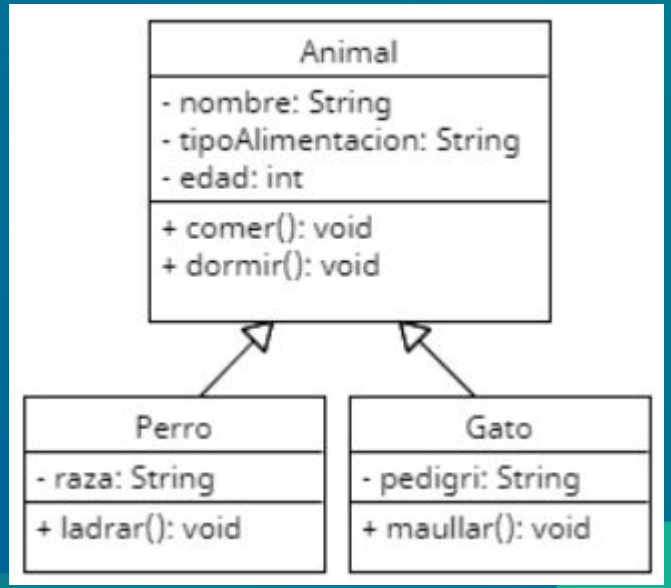




La herencia permite agrupar los miembros en común, evitando la repetición.

Ejemplo de herencia

Superclase
Clase base
Clase padre



Subclases
Clases derivadas
Clases hijas

Ejemplo de herencia

```java
public class Animal {

    private String nombre;
    private String tipoAlimentacion;
    private int edad;

    public void comer() {
        // Método a resolver...
    }

    public void dormir() {
        // Método a resolver...
    }
}
```

```java
public class Perro extends Animal {

    private String raza;

    public void ladrar() {
        // Método a resolver...
    }

}
```

```java
public class Gato extends Animal {

    private String pedigri;

    public void maullar() {
        // Método a resolver...
    }

}
```

¿Qué se hereda?

Las clases derivadas heredan todos los miembros **no privados** de la clase base (excepto los constructores).

Modificadores de acceso

Modificador en Java	Modificador en UML	Misma clase	Subclase en mismo paquete	Clase en mismo paquete	Subclase en otro paquete	Clase en otro paquete
`public`	+	✓	✓	✓	✓	✓
`protected`	#	✓	✓	✓	✓	✗
`(sin modificador)`		✓	✓	✓	✗	✗
`private`	-	✓	✗	✗	✗	✗

Modificador protected

A diferencia de otros lenguajes, colocar los atributos como protegidos en Java, puede derivar en problemas de encapsulamiento:

- Habría que asegurarse que las clases que componen a la jerarquía residan en un paquete de forma exclusiva, ya que cualquier otra clase en el mismo paquete podría acceder directamente a sus atributos.

  (ver tabla)

*Lo recomendable es definir atributos privados y métodos **getter/setter** cuando sean necesarios.*

Constructores

Los constructores **no** se heredan.

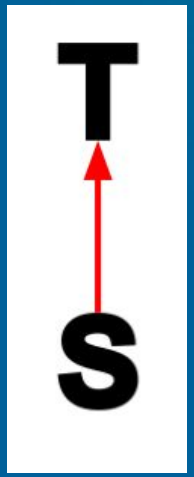
Cada clase dentro de la jerarquía debe tener su constructor, que debe invocar primero al constructor de la clase base.

```java
public Animal(String nombre, String alimento, int edad){

    this.nombre = nombre;
    this.tipoAlimentacion = alimento;
    this.edad = edad;
}
```

```java
public Perro(String nombre, String alimento, int edad, String raza){

    super(nombre, alimento, edad);

    this.raza = raza;
}
```

```java
public Gato(String nombre, String alimento, int edad, String pedigri){

    super(nombre, alimento, edad);

    this.pedigri = pedigri;
}
```

Principio de sustitución



Cada clase que hereda de otra puede usarse como su padre sin necesidad de conocer las diferencias entre ellas.

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.

Barbara Liskov

Herencia

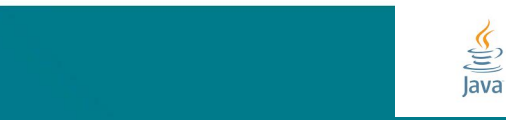

LISKOV SUBSTITUTION PRINCIPLE
If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

Principio de sustitución

```java
public static void main(...) {

    Animal a = new Animal(...); // OK

    Perro p = new Perro(...); // OK

    Animal a2 = new Perro(...); // OK, TODO PERRO ES UN ANIMAL

    Perro p1 = new Animal(...); // ERROR!, NO TODO ANIMAL ES UN PERRO
}
```

```java
public static void main(...) {

    Perro p = new Perro(...);

    Animal a = p; // OK

    Animal a2 = (Animal)p; // OK, ÍDEM ANTERIOR

    Perro p1 = a; // ERROR!

    Perro p2 = (Perro)a; //OK

    Gato g = (Gato)a;// ERROR!
}
```

Principio de sustitución




```java
public static void main(...) {

    Perro p = new Perro(...);

    p.comer(); // OK

    p.ladrar(); // OK

    Animal a = p;

    a.comer(); // OK

    a.ladrar(); // ERROR!

    Object x = p;

    x.comer(); // ERROR!

    x.ladrar(); // ERROR!

}
```

```java
public static void main(...) {

    Animal a = p;

    a.comer(); // OK

    ((Perro)a).ladrar(); // OK

    Object x = p;

    ((Perro)x).comer(); // OK

    ((Perro)x).ladrar(); // OK

}
```

Operador instanceof

Operador lógico que permite conocer si una variable contiene un objeto de un determinado tipo o no.

```
public static void main(...) {

    Animal p = new Perro(...);

    System.out.println( p instanceof Perro); // TRUE

    System.out.println( p instanceof Animal); // TRUE

    System.out.println( p instanceof Object); // TRUE

    System.out.println( p instanceof Gato); // FALSE

    System.out.println( p instanceof String); // ERROR! DEBE PERTENECER A LA JERARQUÍA

}
```

Ejercitación