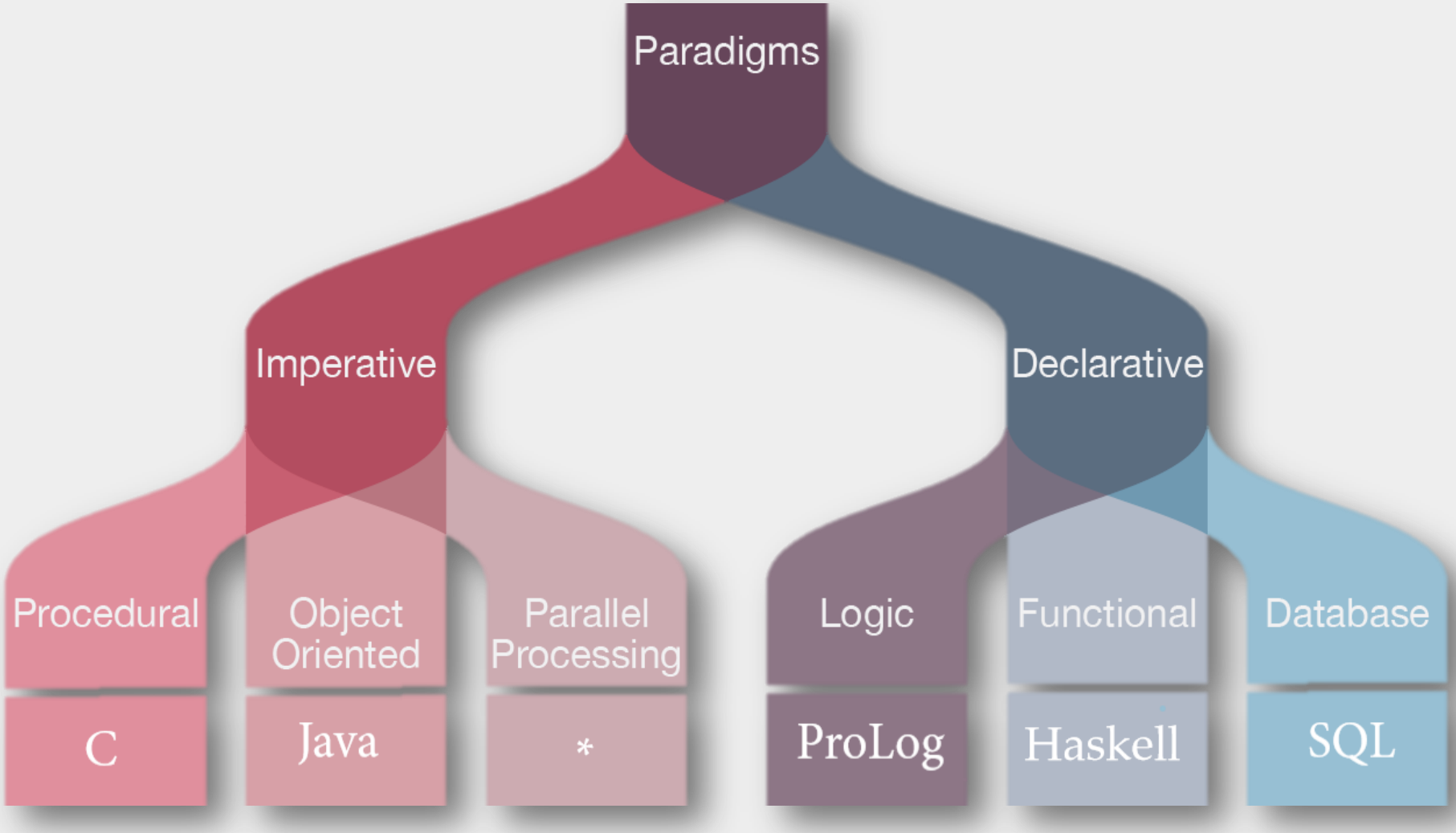
Programación orientada a objetos (POO)

Paradigmas de programación



Ventajas de POO

- Modularización
- Reutilización
- Abstracción
- Encapsulamiento
- Herencia

Objeto

- Estado, propiedades y comportamiento.
- *Ejemplo*:
  - **Coche**
    - **Estado**: parado, circulando, aparcado…
    - **Propiedades**: color, peso, motor…
    - **Comportamiento**: acelerar, frenar, girar…

En Java

- **Declaración** de objeto: **`class`**
- **Instanciación** de objeto: **`new`** `MiObjeto()`**`;`**
- **Estado** y **propiedades**: variables y constantes.
- **Comportamiento**: métodos.

Conceptos de POO en Java

- **`this`**: Acceso al objeto dentro del objeto.
- **Constructor**: Creación de una instancia del objeto.
- **`public`**: Acceso externo a un atributo o método.
- **`private`**: Acceso restringido a un atributo o método.
- **`protected`**: Acceso restringido a un atributo o método excepto a hijos de la clase y dentro del paquete.

Getters / Setters

- **Getter**: Obtener atributo del objeto.
- **Setter**: Modificar atributo del objeto.
- **Ventajas**:
  - Atributos de sólo lectura.
  - Validación parámetros de entrada.
  - Inicialización de atributos.
  - Modificación de funcionalidad interna sin afectar al usuario de la clase.

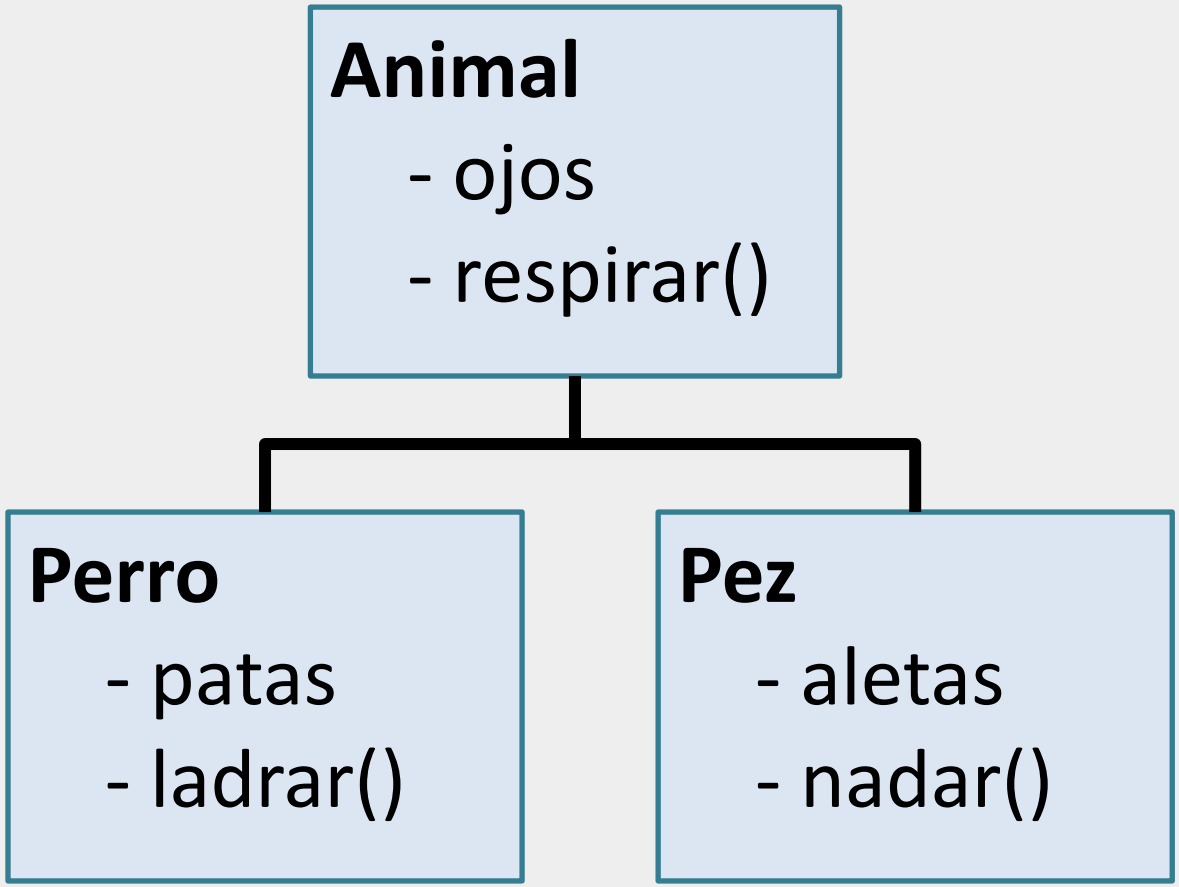
Conceptos avanzados de POO

- Herencia
- Contexto (estático o dinámico)
- Sobrecarga de métodos
- Polimorfismo
- Sobreescritura de métodos
- Métodos abstractos

Herencia

Padre:

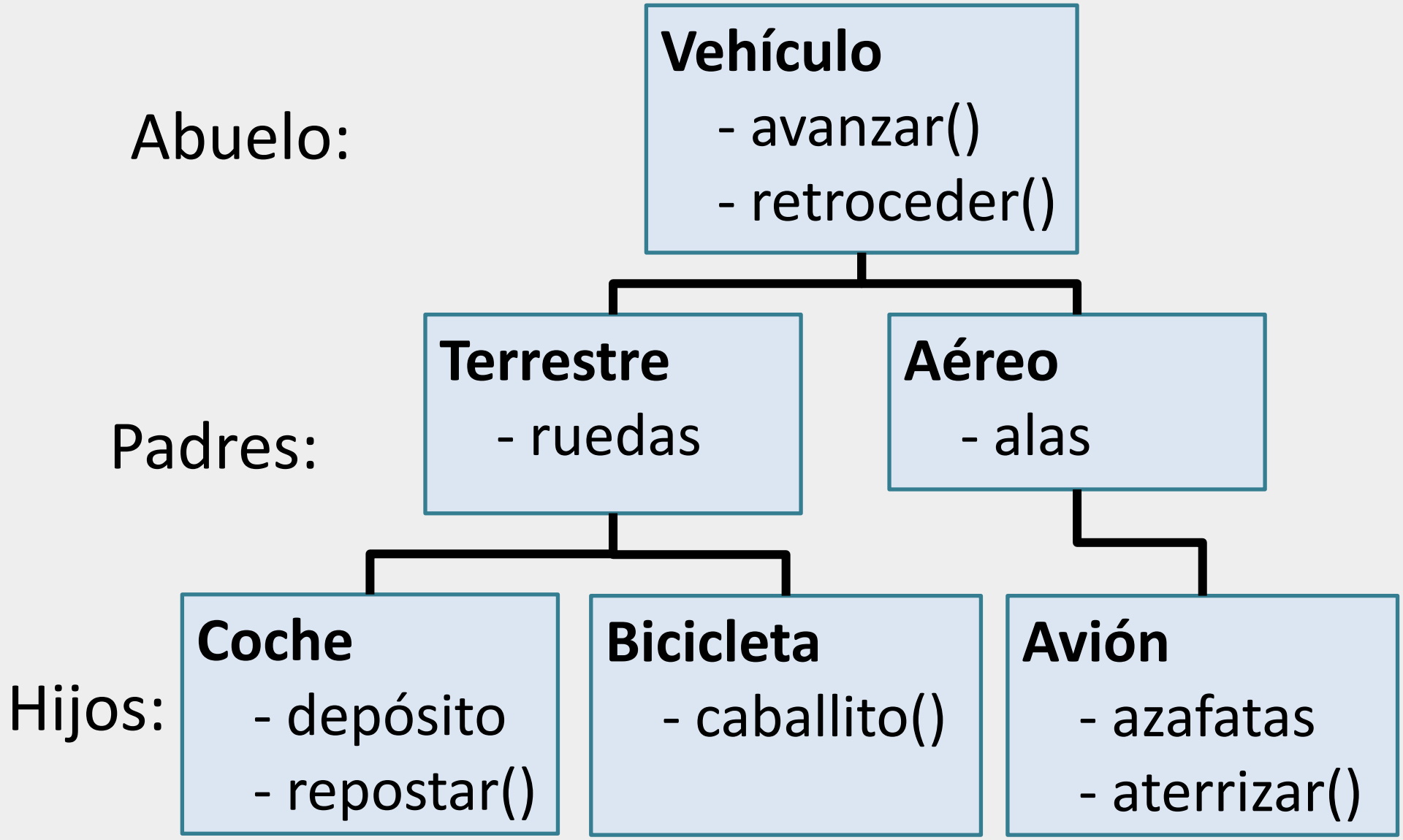
Hijos:



Herencia en Java

```
class Hijo extends Padre{
    Hijo() {
        super();
    }
}
```

Herencia


Abuelo:
Vehículo
- avanzar()
- retroceder()
Padres:
Terrestre
- ruedas
Aéreo
- alas
Hijos:
Coche
- depósito
- repostar()
Bicicleta
- caballito()
Avión
- azafatas
- aterrizar()

Contexto

Contexto estático:

clase **Coche**
- color = **?**
- ruedas = **4**

Contexto dinámico:

Instancias de la clase Coche:

cocheDePepe
- color = rojo

cocheDePaco
- color = blanco

cocheDeLola
- color = azul

cocheDeRupert
- color = negro

Sobrecarga de métodos

Declaración:

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

void comer(String comida){
    System.out.println("Estoy comiendo " + comida);
}

void comer(String comida, int cantidad){
    System.out.println("Estoy comiendo " + cantidad + " " + comida);
}
```

Sobrecarga de métodos

Llamada:

```
comer();
comer("Zanahorias");
comer("Zanahorias", 4);
```

Resultado:

```
Estoy comiendo
Estoy comiendo Zanahorias
Estoy comiendo 4 Zanahorias
```

Polimorfismo

Ejemplo:

```
Animal animal = new Animal();
Perro perro = new Perro();
Animal animal2 = new Perro();
```

animal es un **Animal**

perro es un **Perro** y un **Animal**

animal2 es un **Perro** y un **Animal**

Métodos:

```
perro.comer();
perro.ladrar();
animal.comer();
animal.ladrar();
```

Asignación:

```
animal = perro;
animal2 = perro;
perro = animal;
perro = animal2;
```

Casting

Ejemplo:

```
Animal animal = new Perro();
Perro perro;

perro = animal;
perro = (Perro) animal;
```

instanceof

Ejemplo:

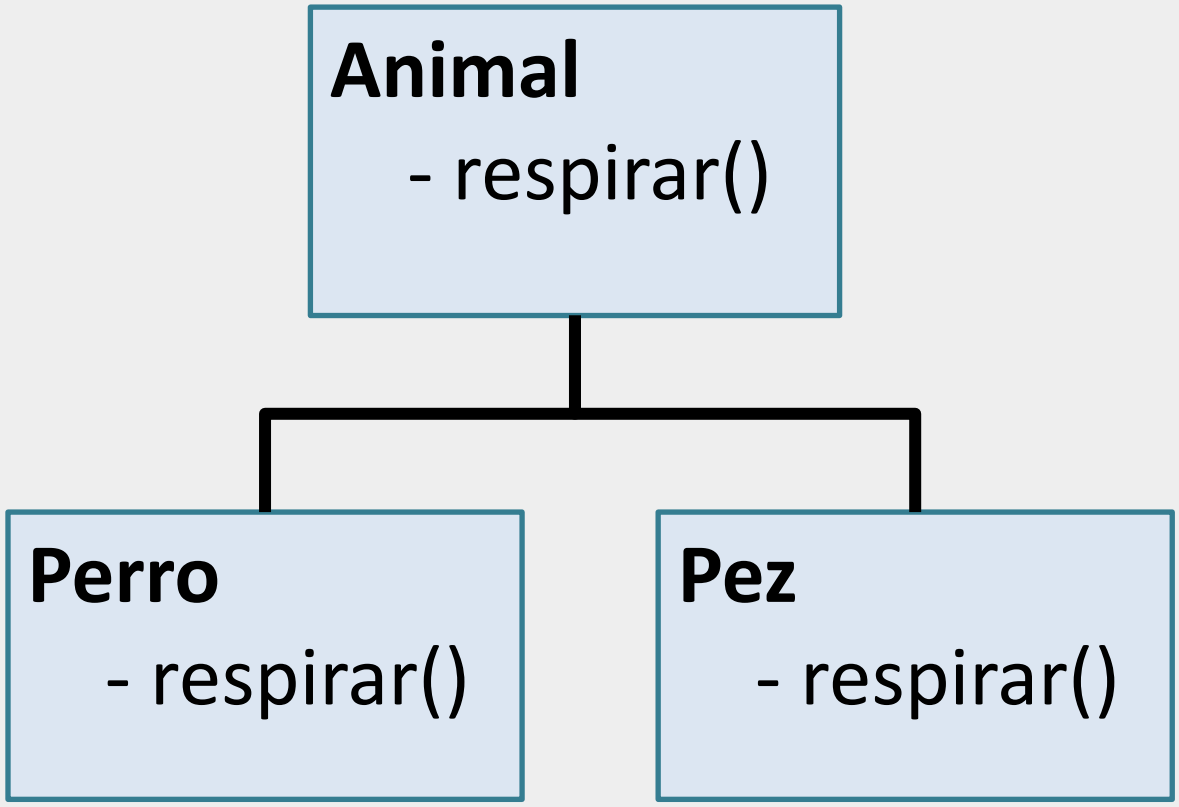
```
Animal animal = new Animal();
Perro perro = new Perro();
```

perro ***intanceof*** *Animal*	TRUE
perro ***intanceof*** *Perro*	TRUE
animal ***intanceof*** *animal*	TRUE
animal ***intanceof*** *Perro*	FALSE

Sobreescritura de métodos

Padre:

Hijos:



Sobreescritura de métodos

Animal

```
public void respirar(){
    System.out.println("Estoy respirando");
}
```

Perro

```
@Override
public void respirar(){
    System.out.println("Estoy respirando con pulmones");
}
```

Pez

```
@Override
public void respirar(){
    System.out.println("Estoy respirando con branquias");
}
```

Sobreescritura de métodos

Llamada:

```
animal.respirar();
perro.respirar();
pez.respirar();
```

Resultado:

```
Estoy respirando
Estoy respirando con pulmones
Estoy respirando con branquias
```

Métodos abstractos

Animal:

```
public abstract void respirar(); // No tiene implementación
```

Perro:

```
@Override
public void respirar(){
    System.out.println("Estoy respirando con pulmones");
}
```

Pez:

```
@Override
public void respirar(){
    System.out.println("Estoy respirando con branquias");
}
```