



# P00 en Java

Diego Iván Oliveros Acosta

# Agenda

- ¿Qué es la POO?
  - Cuáles son las ventajas de este paradigma

# Algunos paradigmas de programación (formas de programar)

---

- Paradigmas de la programación
  - Programación orientada a objetos
  - Programación orientada a procedimientos
    - Fortran, Cobol , Basic, etc...



# Desventajas POO v.s procedimientos :

```
READY
10 FOR X=1 TO 10
20 PRINT "HOLA WIKIPEDIA"
30 NEXT X
RUN
HOLA WIKIPEDIA
HOLA WIKIPEDIA
HOLA WIKIPEDIA
HOLA WIKIPEDIA
HOLA WIKIPEDIA
HOLA WIKIPEDIA
HOLA WIKIPEDIA
HOLA WIKIPEDIA
HOLA WIKIPEDIA
HOLA WIKIPEDIA
READY
■
```

- Unidades de código muy grandes en aplicaciones complejas
- En aplicaciones complejas el código resulta difícil de descifrar
- Poco reusable
- Si existe fallo en una línea de código, es muy probable que el programa caiga.
- Aparición frecuente de código espagueti
- Difícil de depurar por otros programadores en caso de un error.



# POO

¿En qué consiste?

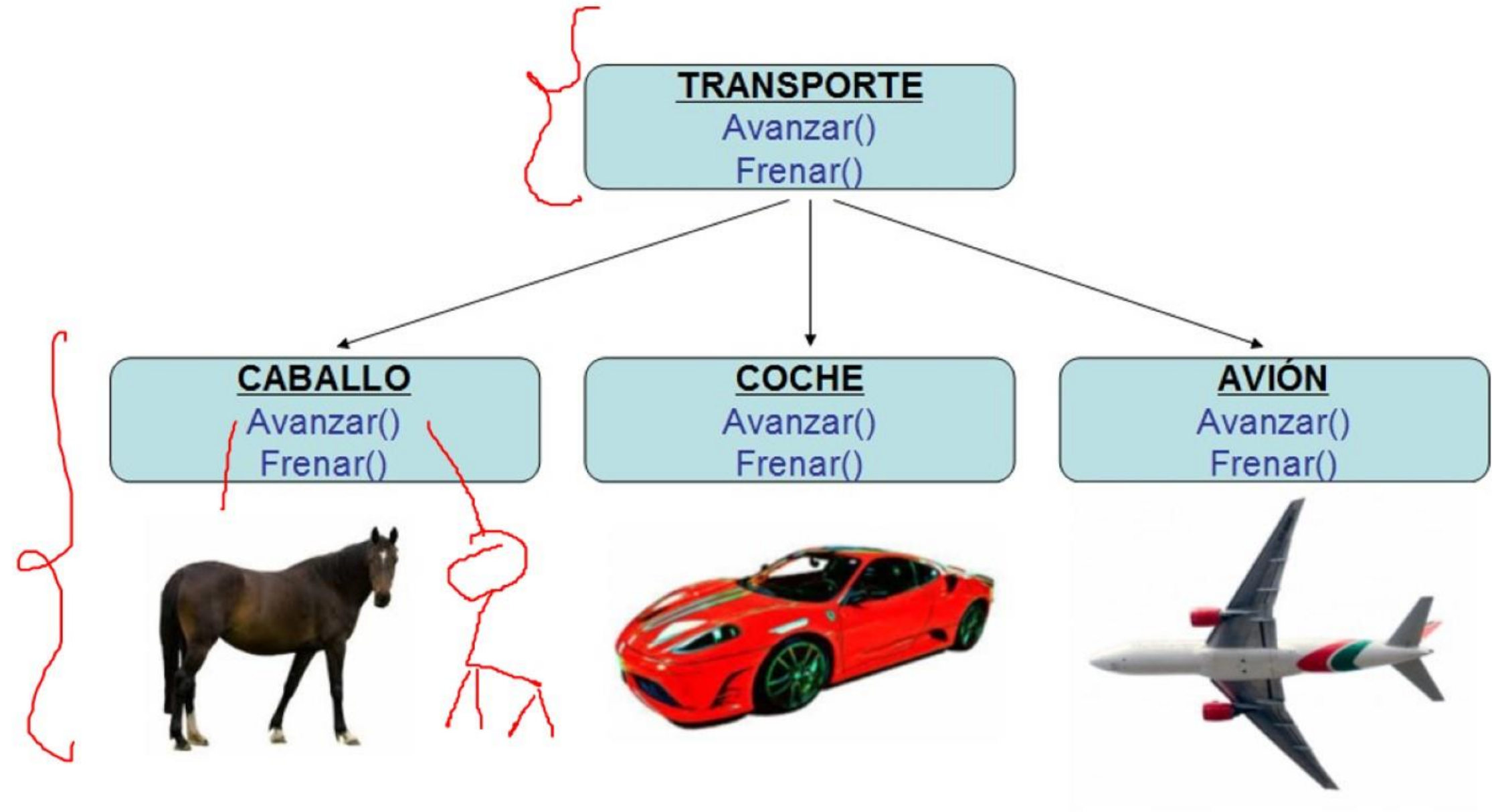
- trasladar la naturaleza de los objetivos de la vida real al código de programación.

¿cuál es la naturaleza de un objeto de la vida real?

Los objetivos tienen un estado, un comportamiento(¿qué puede hacer?), y unas propiedades

Pongamos un ejemplo: el objeto auto

- ¿Cuál es el estado de un auto? Un auto puede estar parado, circulando, aparcado etc.
- ¿Qué propiedades tiene un auto? UN auto tiene un color, un peso, un tamaño,
- ¿Qué comportamiento tiene un auto? Arrancar, frenar, acelerar, frenar, girar



# Objeto ¿Qué es?



**Objeto:**



**Tiene propiedades(atributos):**

Color  
Peso  
Alto  
Largo



**Tiene un comportamiento (¿qué es capaz de hacer?)**

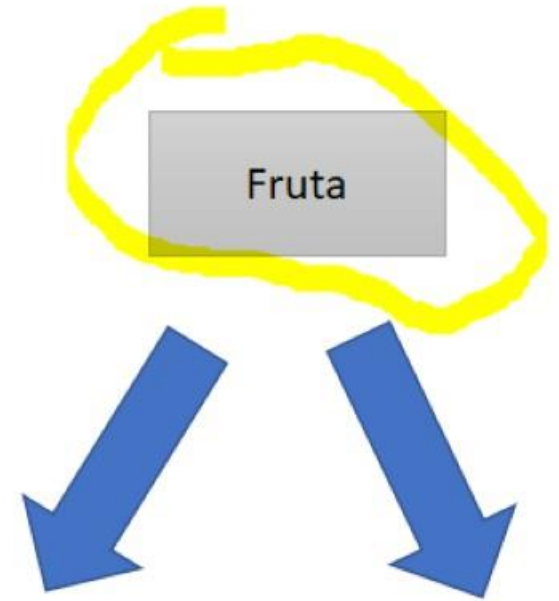
Arrancar  
Frenar  
Girar  
Acelerar

- **Clases**

- Representan conceptos o entidades significativas de un problema
- Se pueden ver como plantillas para definir elementos

- **Objetos**

- Pueden estar directamente relacionadas unas con otros objetos
- Elementos con comportamiento definido en la clase y estado concreto
- Instancias de clase
- Interactúan por medio de mensajes





# Ventajas POO

Algunos ejemplos de lenguajes: C++, Java, Visual.NET etc.

Programas divididos en “trozos”, “partes”, módulos, clases, modularización.

Muy reusable, herencia.

Si existe fallo en alguna línea de código, el programa continua con su funcionamiento. Tratamiento de excepciones.

Encapsulamiento

# Vocabulario de la POO

---

Clase

---

Objeto

---

Ejemplar de clase. Instancia de clase. Ejemplarizar una clase.

---

Instanciar una clase

---

Modularización

---

Encapsulamiento/ encapsulación

---

Herencia

---

Polimorfismo



# Clase

Modelo donde se redactan las características comunes de un grupo de objetos

# Instancia



Ejemplar perteneciente a una clase



Clase



Objeto



- Programa
  - Clase 1
  - Clase 2
  - Clase 3



Modularizacion

# Creación de Clases y objetos

Objeto en concreto

**Auto**

+marca |  
+cantidadPuertas |  
+color  
+cilindrada  
+consumoCombustible  
+kilometrosRealizados

+Encender()  
+CambioRueda()  
+CombustibleConsumido()  
+CombustibleConsumidoPorMes(pMes)  
+KilometrosRealizados()  
+KilometrosRealizadosPorMes(pMes)

Objeto abstracto

**Rectangulo**

+coordenadaSuperiorIzquierda  
+coordenadaInferiorDerecha  
+tipoDeLinea  
+colorDeLinea  
+colorDeRelleno

+Mostrarse()  
+Ocultarse()  
+CambiarDePosicion()

# Creación de objetos

```
//Para poder usar un objeto hay que crearlo:  
clase identificador = new clase();  
Perro miPerro = new Perro("Pancho");  
miPerro.edad = 5;  
miPerro.ladrear();
```

# Elementos de clase

```
class Circulo {  
    // campos  
    // métodos  
    // constructores  
    // main()  
}
```

Circulo	
-	radio: double = 5
	color: String
	numeroCirculos: int = 0
	PI: double = 3.1416
+	Circulo()
	Circulo(double)
	getRadio(): double
	setRadio(double): void
	getColor(): String
	setColor(String): void
	getCircunferencia(): double
	getCircunferencia(double): double
	getNumeroCirculos(): int
	main(String[]): void



## Objeto(accediendo a propiedades y comportamiento-seudocódigo )

- Objeto:
  - Accediendo a **propiedades** de objeto desde código (seudocódigo)
  - Accediendo a **comportamiento** de objeto desde código (seudocódigo)

```
miauto.arrancar();  
miauto.frena();  
miauto.gira();  
miauto.acelera();
```

```
miauto.color="rojo";  
miauto.peso=1500;  
miauto.ancho=2000  
miauto.alto=900
```

# Métodos



```
{  
tipo_retorno nombre_método (parámetros) {  
código del método  
}  
  
public static void main(String[] args){  
}
```

This

```
public class Perro {  
    String nombre;  
    Perro(String nom) {  
        this.nombre = nom;  
    }  
}
```

## Code:

```
public class Perro {  
    // Atributos  
    String nombre;  
    int edad;  
    // Métodos  
    Perro(String nom) {  
        this.nombre = nom;  
    }  
    void ladrar() {  
        System.out.println("¡Guau!");  
    }  
}
```



# Métodos habituales (Destructor)

- Se ejecuta al crear destruir un objeto
- Se tiene que sobrescribir
- No devuelve nada por definición

```
void finalize()
```

# Encapsulamiento – Encapsulación

- Puede y suele haber distintos niveles de visibilidad

+ **Public:** se puede acceder desde cualquier lugar

— **Private:** solo se puede acceder desde la propia clase

**Protected:** solo se puede acceder desde la propia clase o desde una clase que hereda de ella.

# Visibilidad

```
public String nombre;  
private int edad = 10; (  
  
public void ladar() {  
    System.out.println("¡Guau!");  
}
```

# Herencia

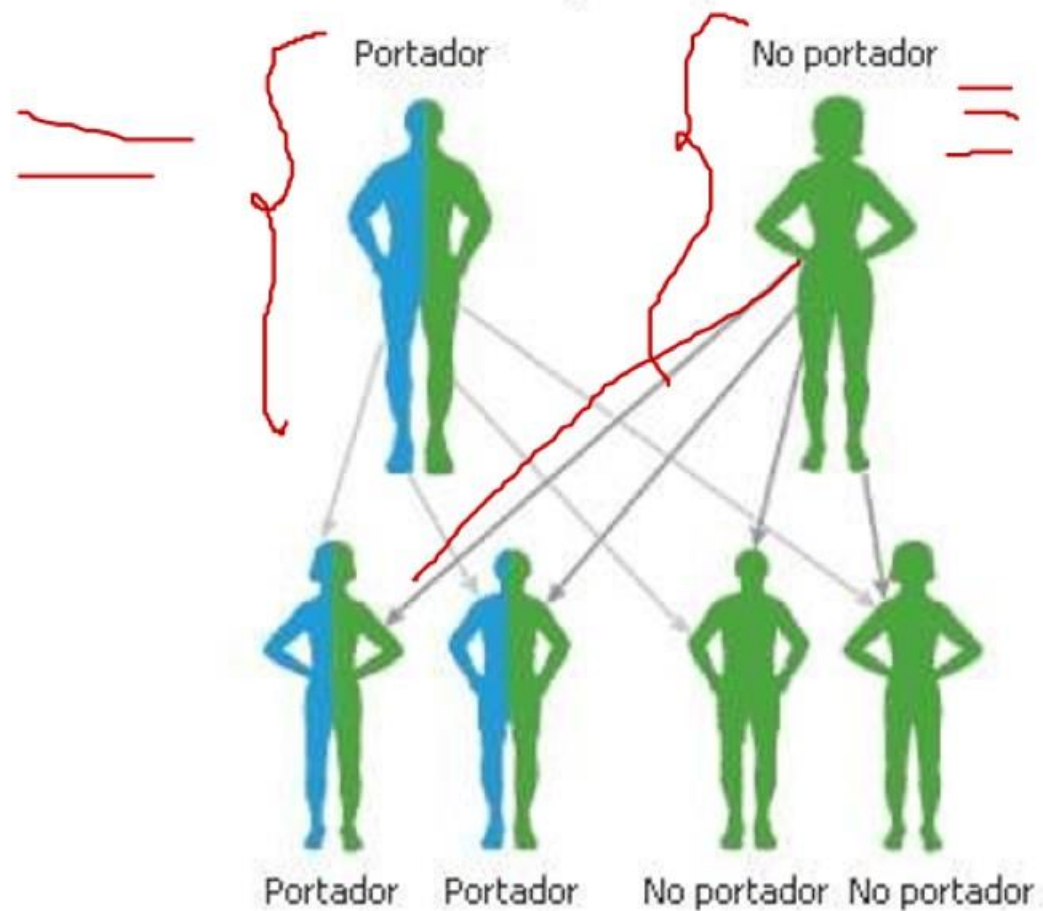
Qué es?

Para qué sirve?

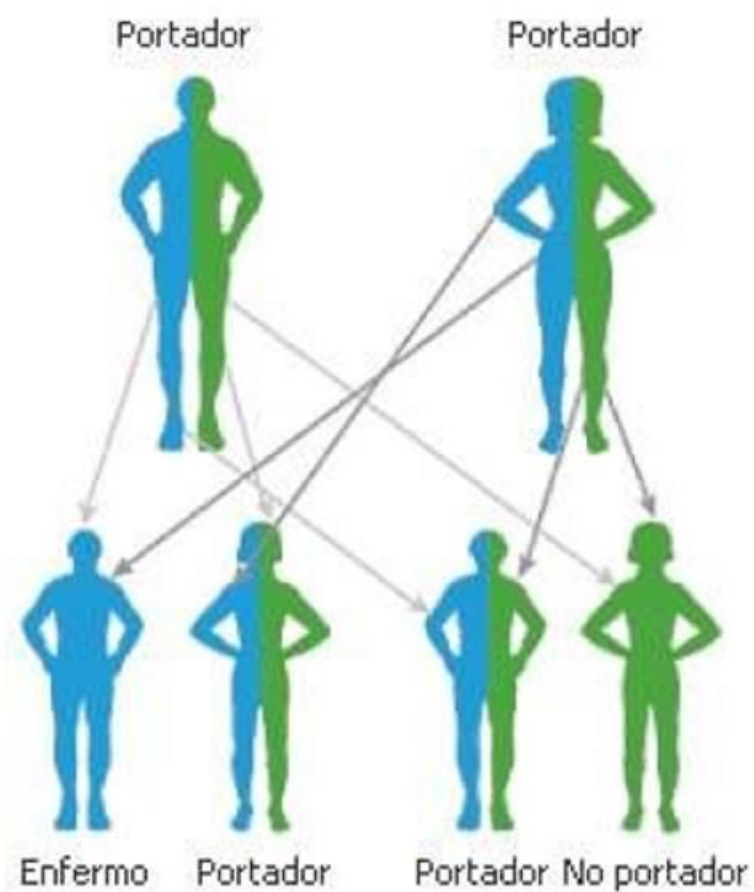
Cómo hacer herencia?



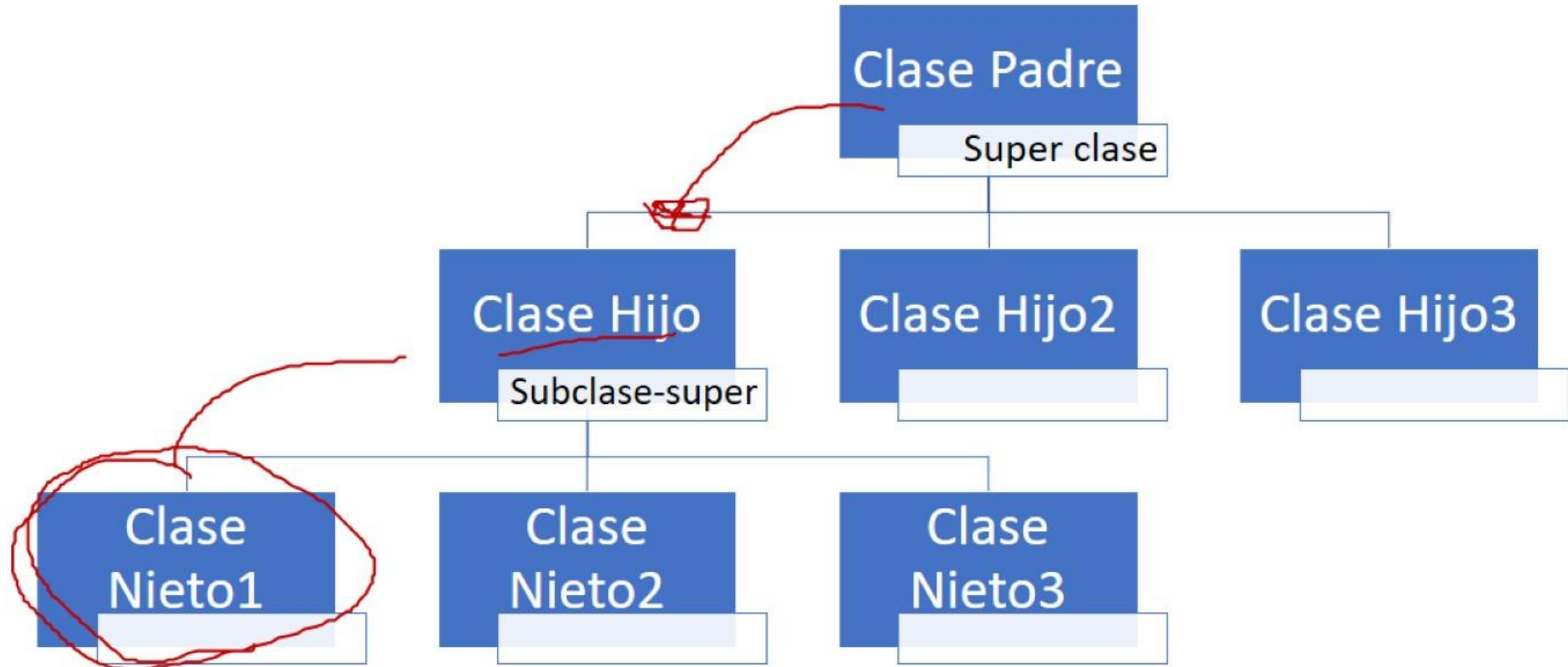
### Uno de los padres portador



### Ambos padres portadores



# Herencia



# Ventajas

- Reutilización ✓
- Super
- ¿Qué características en común tienen todos los objetos?
- ¿Qué comportamientos en común tienen todos los objetos?

```
import MiClase;
public class MiNuevaClase extends MiClase {
    public void Suma_a_i( int j ) {
        i = i + ( j/2 );
        super.Suma_a_i( j );
    }
}
```

# Herencia

```
//Clase para objetos de dos dimensiones
class DosDimensiones{
    private double base;
    private double altura;

    void mostrarDimension(){
        System.out.println("La base y altura es: "+base+"
y "+altura);
    }
}
```



# Herencia

CP

```
//Una subclase de DosDimensiones para Triangulo
class Triangulo extends DosDimensiones{
    String estilo;
    double area(){
        return base*altura/2; //Error! no se puede acceder
    }

    void mostrarEstilo() {
        System.out.println("Triangulo es: "+estilo);
    }
}
```



# Interfaces

```
public interface Receta {  
    void preparaReceta();  
}  
...  
public class Paella implements Receta {  
    ...  
public class LubinaAlHorno implements Receta {  
    ...
```

```

6     def estado(self):
7         print("Marca: ", self.marca, "\nModelo", self.modelo,
8             "\nEn marcha", self.enmarcha, "\nAcelerando", self.acelera, "\n")
9
10    class Furgoneta(Vehiculos):
11
12        def carga(self, cargar):
13            self.cargado=cargar
14            if(self.cargado):
15                return "La furgoneta está cargada"
16            else:
17                return "La furgoneta no está cargada"
18
19    class Moto(Vehiculos):
20        hcaballito=""
21        def caballito(self):

```

```

miMoto=Moto("Honda", "CBR")

miMoto.caballito()

miMoto.estado()

miFurgoneta=Furgoneta("Renaut", "Kangoo")

miFurgoneta.arrancar()

```

```

55
56 miMoto=Moto("Honda", "CBR")
57
58 miMoto.caballito()
59
60 miMoto.estado()
61
62 miFurgoneta=Furgoneta("Renaut", "Kangoo")
63
64 miFurgoneta.arrancar()
65
66 miFurgoneta.estado()
67
68 print(miFurgoneta.carga(True))
69
70 class BicicletaElectrica(Ve electricos, Vehiculos):
71     pass
72
73 miBici=BicicletaElectrica("", "")
74

```

# Herencia multiple

# Sobrecarga de Operadores

```
public class Punto3D {  
    private int x;  
    private int y;  
    private int z;  
    public Punto3D() {  
        x = y = z = 0;  
    }  
    public Punto3D(int x) {  
        this.x = x;  
    }  
}
```

```
✓ public Punto3D(int x, int y) {  
    this.x = x;  
    this.y = y;  
}  
✓ public Punto3D(int x, int y, int z) {  
    this.x = x;  
    this.y = y;  
    this.z = z;  
}
```

# Polimorfismo de clases

```
class Animal {  
    public void makeSound() {  
        System.out.println("Grr...");  
    }  
}  
class Cat extends Animal {  
    public void makeSound() {  
        System.out.println("Meow");  
    }  
}  
class Dog extends Animal {  
    public void makeSound() {  
        System.out.println("Woof");  
    }  
}
```

```
public static void main(String[] args) {  
    Animal a = new Dog();  
    Animal b = new Cat();  
}
```

```
a.makeSound();  
//Outputs "Woof" ✓
```

```
b.makeSound();  
//Outputs "Meow" ✓
```



# Polimorfismo paramétrico

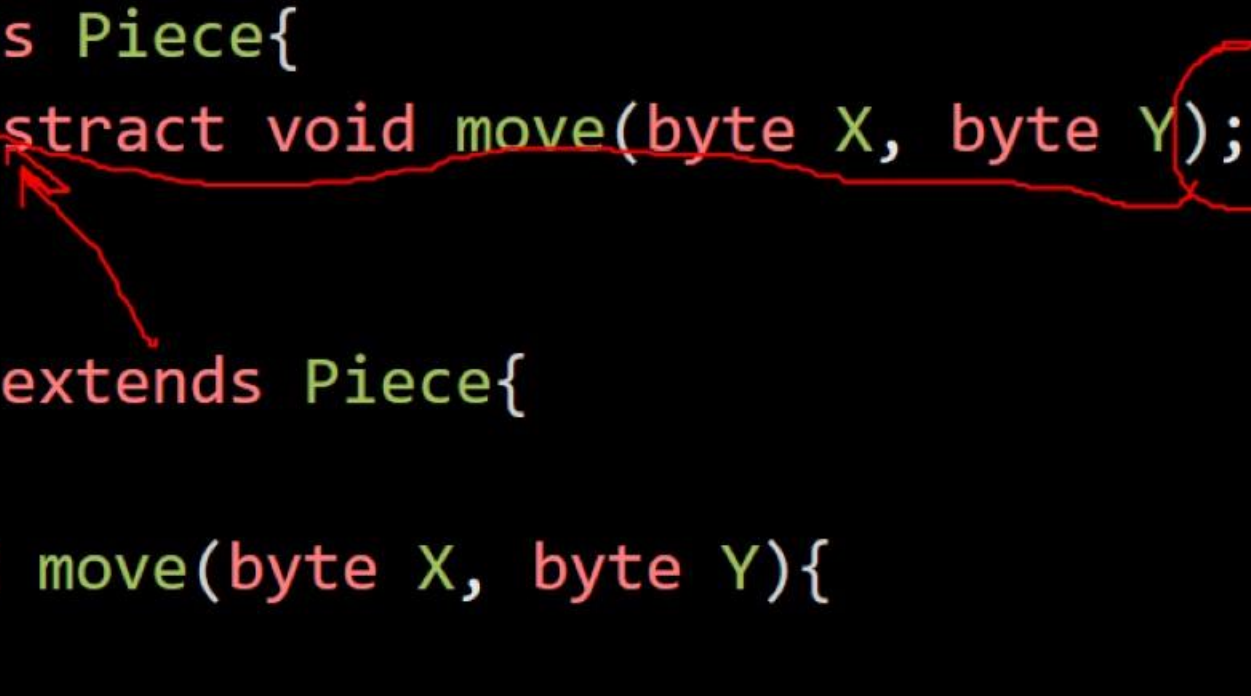
```
class Overload {  
    void demo (int a) {  
        System.out.println ("a: " + a);  
    }  
    void demo (int a, int b){  
        System.out.println  
("a and b: " + a + ", " + b);  
    }  
    double demo(double a){  
        System.out.println("double a: " +  
a);  
        return a*a;  
    }  
}
```

```
class MethodOverloading  
{  
    public static void main  
(String args [])  
    {  
        Overload Obj = new Overload();  
        ✓ double result;  
        Obj .demo(10); ✓  
        Obj .demo(10, 20); ✓  
        result = Obj .demo(5.5);  
        System.out.println  
("O/P : " + result);  
    }  
}
```



# Polimorfismo de inclusión

```
abstract class Piece{  
    public abstract void move(byte X, byte Y);  
}  
  
class Bishop extends Piece{  
    @Override  
    public void move(byte X, byte Y){  
    }  
}
```



# Referencias:

- **BOOCH.** (2013). Orientación a objetos. Pearson Education.
- **GRADY, B.** (2001). Analisis de diseño orientado a objetos con aplicaciones. Mexico: S.A. Alhambra Mexicana.
- **HERNANDEZ.** (2013). <https://www.fdi.ucm.es>. Obtenido de <https://www.fdi.ucm.es/profesor/luis/fp/FP.pdf>
- **JOYANES.** (2003). Fundamentos de Programación. Libro de problemas (Segunda ed.). Interamericana de España. (2006). Programación en Pascal (Cuarta ed.). Interamericana de España. (2008). Fundamentos de la programación. Madrid: Interamericana de España. (2008). Fundamentos de Programación. Algoritmos, estructura de datos y objetos. Madrid: Interamericana de España