



# Programación Orientada a Objetos

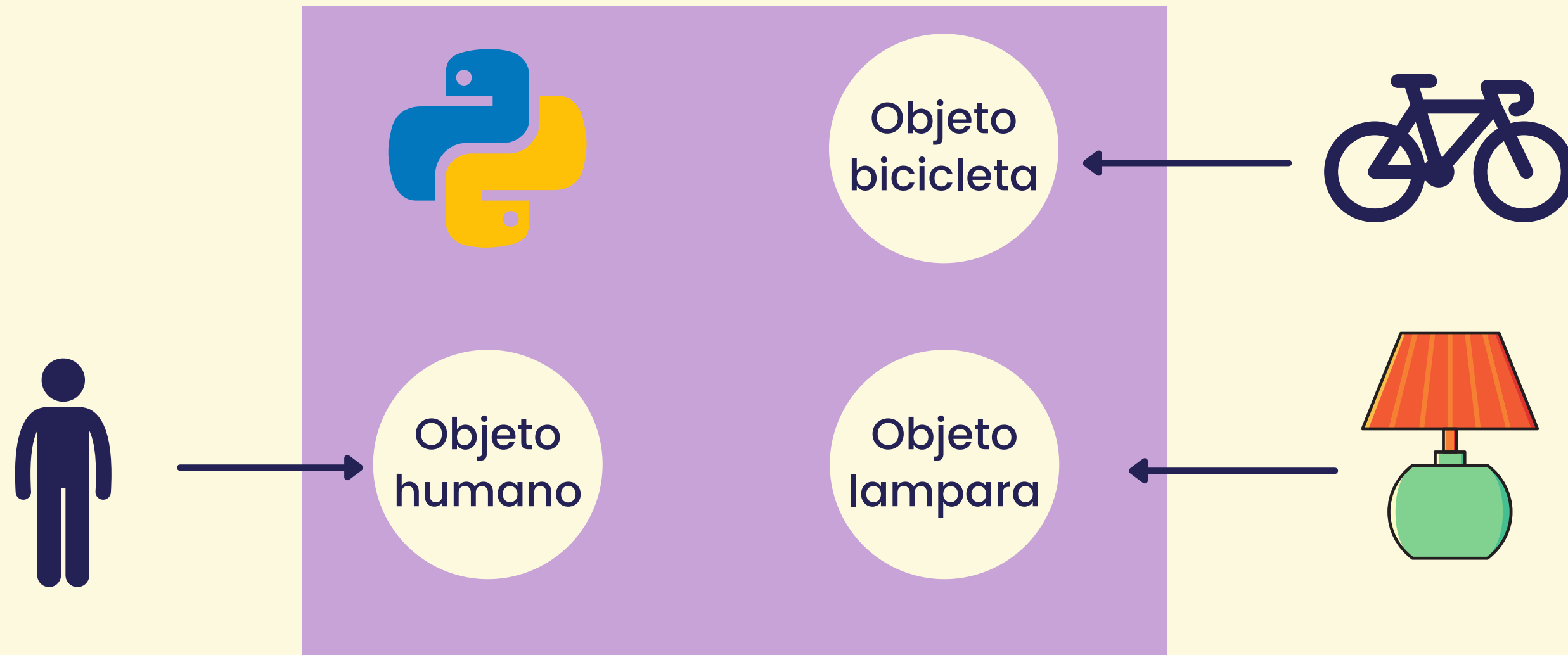
Gutiérrez Cruz Abel Isaías

# ¿En qué consiste?

Se basa en crear clases y objetos para la construcción del software

## ¿Qué es un objeto?

- Son cosas que pueden ser percibidas por algunos de nuestros sentidos cuando censamos nuestro ambiente
- Estos objetos tienen características específicas tales como su tamaño, forma, color, material, etc. Además de esto también tienen funciones específicas



# Composición de un objeto

Se componen por características del objeto y por métodos que lleva a cabo ese objeto.

## ¿De donde vienen esos objetos?

Son creados a través de una plantilla, la cual tiene establecido que características debe tener cada objeto a partir de ella. Estas plantillas son conocidas como clases

Una misma clase puede crear diferentes objetos, cada uno con las mismas características, pero independientes.



class Lampara

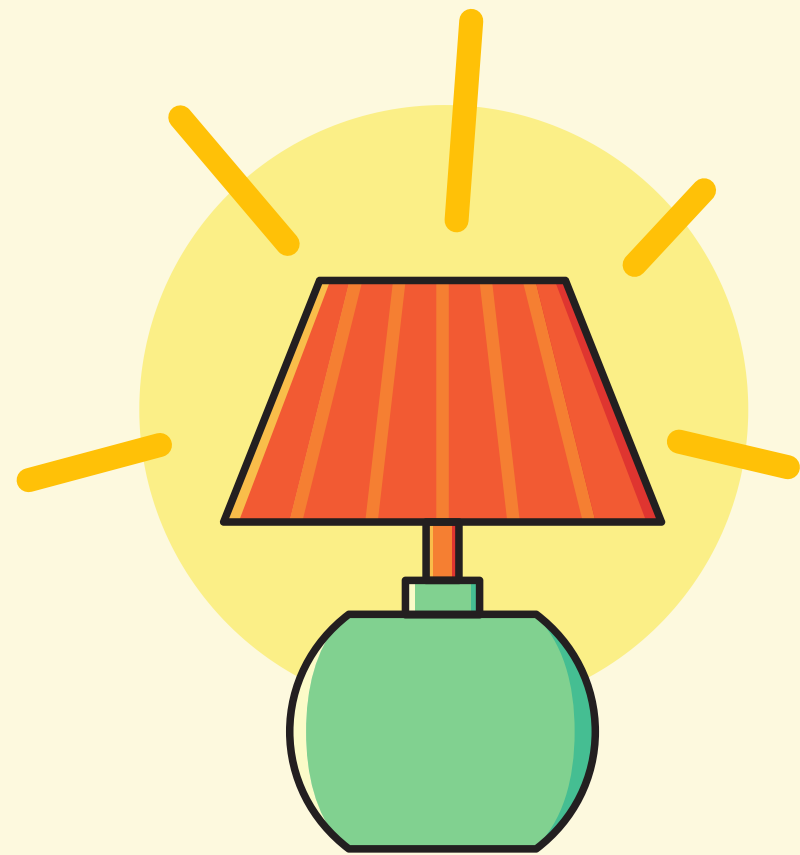
iluminando

encender()

apagar()

Características

Comportamiento de  
la lampara



Encendida

```
class Lampara
```

```
    iluminando
```

```
    encender()
```

```
    apagar()
```

← Guarda True ahora

← Este comportamiento  
es ejecutado



Apagada

```
class Lampara
```

```
    iluminando
```

```
    encender()
```

```
    apagar()
```

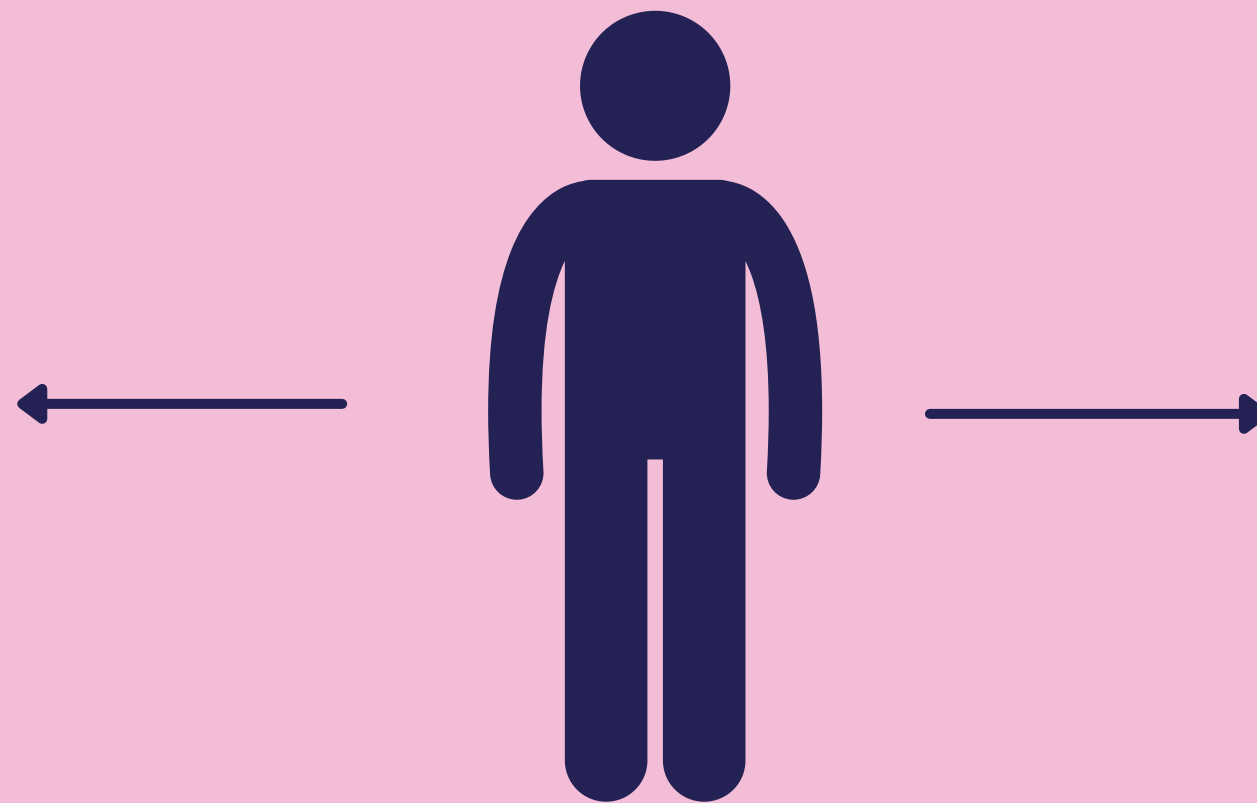
← Guarda False ahora

← Este comportamiento  
es ejecutado

# Empleado en una compañía

Plantilla general

- Atributos
- ID
  - Salario
  - Departamento



- Comportamiento
- Calculo de salario por día
  - Calculo de impuestos

Propiedades

ID

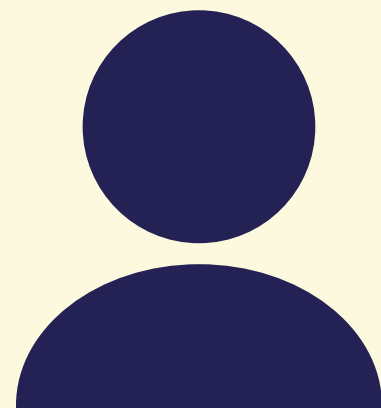
Salario

Departamento

Métodos

salarioPorDia()

impuestos()



Ricardo

ID: 568912

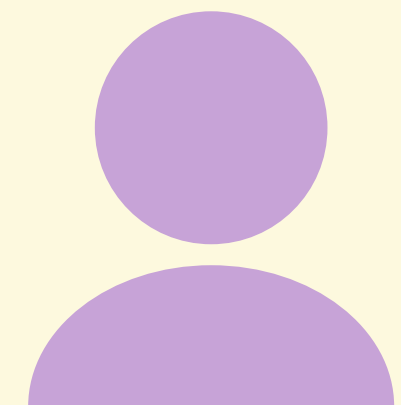
Salario: 300

Departamento: Finanzas

ID: 894536

Salario: 400

Departamento: Marketing



Efren




# Propiedades

Son variables que contienen información perteneciente al objeto de la clase.

# Métodos

Funciones que tienen acceso a las propiedades (y a otros métodos) de la clase. Los métodos también pueden aceptar parámetros y devolver valores.

# Declarar una clase



```
class <nombre_de_la_clase>:  
    # atributos  
  
    #métodos
```

## Reglas para definir el nombre

- Debe de comenzar con una letra o con un guion bajo
- Debe estar compuesto solo por letras, guiones bajos o números

# Implementación de la clase empleado

## Propiedades

ID  
Salario  
Departamento

## Métodos



```
class Empleado:  
    # atributos  
    ID = None  
    salario = None  
    departamento = None
```

# Crear un objeto (instanciar)



```
Ricardo = Empleado()
```

# Acceder a las propiedades



```
Ricardo.ID = 568912
```

```
print("ID = ", Ricardo.ID)
```

# Inicializar una clase

Método que se encarga establecer las propiedades del objeto cuando este es creado (instanciado).

Este método es ejecutado en cuanto se crea el objeto y se le conoce como constructor

```
class Empleado:
    # definir las propiedades y asignarlas
    def __init__(self, ID, salary, department):
        self.ID = ID
        self.salary = salary
        self.department = department

# Creación del objeto
Ricardo = Empleado(568912, 300, "Finanzas")

# Imprimir sus propiedades
print("ID :", Ricardo.ID)
print("Salary :", Ricardo.salary)
print("Department :", Ricardo.department)
```

# Agregar métodos a la clase

## Propiedades

ID

Salario

Departamento

## Métodos

salarioPorDia()

impuestos()



```
class Empleado:
    # definiendo el constructor
    def __init__(self, ID, salario, departamento):
        self.ID = ID
        self.salario = salario
        self.departamento = departamento

    def salarioPorDia(self):
        return (self.salario / 30)

    def impuestos(self):
        return (self.salario * 0.2)

# Inicializando un objeto de la clase empleado
Ricardo = Empleado(568912, 300, "Finanzas")

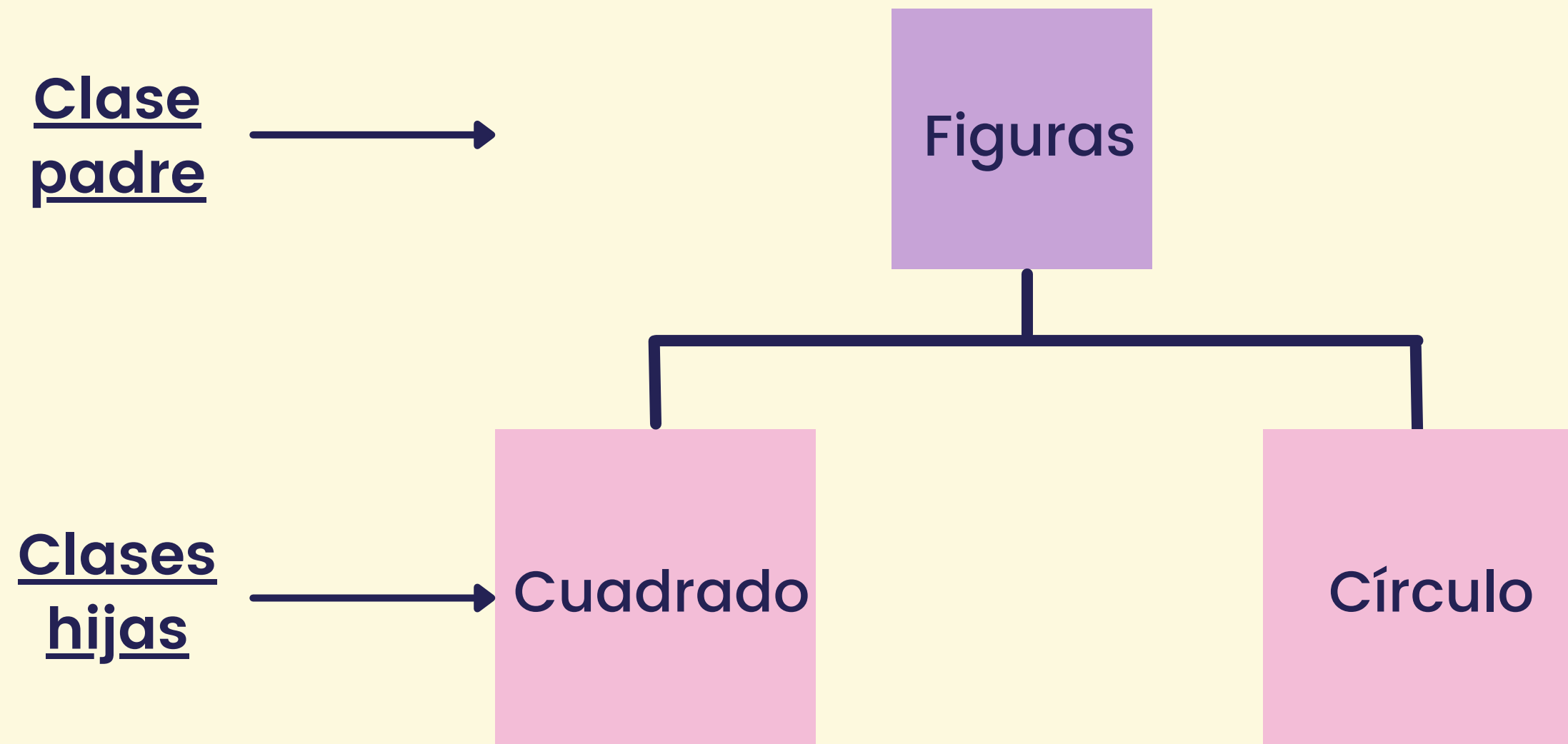
# Imprimir propiedades de Ricardo
print("ID =", Ricardo.ID)
print("Salario", Ricardo.salario)
print("Departamento:", Ricardo.departamento)
print("Impuestos pagados por Ricardo:", Ricardo.impuestos())
print("Salario por día de Ricardo", Ricardo.salarioPorDia())
```

# Herencia

Este concepto proporciona la posibilidad de crear nuevas clases de clases ya existentes. La nueva clase que se creé es una versión especializada de la clase ya existente, la cual heredará todos los campos no privados de la clase anterior.

## Terminología

- Clase padre (super clase): Esta clase permite la reutilización de sus propiedades públicas en otra clase.
- Clase hija (sub clase): Esta clase hereda o extiende a la super clase



# Ejemplo en vehículos

Clase  
padre



Vehículo



Clase  
hijas



Carro

```
class Vehiculo:
    def __init__(self, marca, color, modelo):
        self.marca = marca
        self.color = color
        self.modelo = modelo

    def imprimirDetalles(self):
        print("Productor:", self.marca)
        print("Color:", self.color)
        print("Modelo:", self.modelo)

class Carro(Vehiculo):
    def __init__(self, marca, color, modelo, puertas):
        # llamando el constructor de la clase padre
        Vehiculo.__init__(self, marca, color, modelo)
        self.puertas = puertas

    def imprimirDetallesCarro(self):
        self.imprimirDetalles()
        print("Puertas:", self.puertas)

obj1 = Carro("Suzuki", "Grey", "2015", 4)
obj1.imprimirDetallesCarro()
```



# Ejemplo en vehículos

Para inicializar una clase hija se usa el constructor de la clase padre, para no tener que escribir el mismo código

```
class Vehiculo:
    def __init__(self, marca, color, modelo):
        self.marca = marca
        self.color = color
        self.modelo = modelo
```

```
    def imprimirDetalles(self):
        print("Productor:", self.marca)
        print("Color:", self.color)
        print("Modelo:", self.modelo)
```

```
class Carro(Vehiculo):
    def __init__(self, marca, color, modelo, puertas):
        # llamando el constructor de la clase padre
        Vehiculo.__init__(self, marca, color, modelo)
        self.puertas = puertas
```

```
    def imprimirDetallesCarro(self):
        self.imprimirDetalles()
        print("Puertas:", self.puertas)
```

```
obj1 = Carro("Suzuki", "Grey", "2015", 4)
obj1.imprimirDetallesCarro()
```

# Ejercicios

- **Restaurante.** Elabora una clase llamada Restaurante. El método `__init__()` para Restaurante deberá guardar dos atributos: `nombre_restaurante` y `tipo_cocina`. Realiza un método llamado `descripcion_restaurante()` que imprima estas dos piezas de información, y un método llamado `restaurante_abierto()` que imprima el mensaje indicando que el restaurante esta abierto.  
Crea una instancia llamada `restaurante1` desde la clase recién creada. Imprime los dos atributos individualmente y después llama a ambos métodos.
- **Usuarios.** Elabora una clase llamada Usuario. Crea dos atributos llamados `primer_nombre` y `apellido`, y después crea varios atributos que son típicamente guardados en un perfil de usuario. Realiza un método llamado `descripcion_usuario()` que imprima un resumen de la información del usuario. Realiza otro método llamado `recibir_usuario()` que imprima una bienvenida personalizada al usuario.  
Crea múltiples instancias representando diferentes usuarios y llama a ambos métodos por cada usuario.

# Temas recomendados

- Encapsulación
- Polimorfismo