Universidad Andrés Bello

# Lenguajes de programación - Clase 6

# ***Lenguaje Orientado a Objetos***

# Definición

Universidad Andrés Bello

- La Programación Orientada a Objetos (**POO** u **OOP** según sus siglas en inglés) es un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas de computadora.

- Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo, y encapsulamiento.

- Se basa en el modelo objeto, donde el elemento principal es el objeto, el cual es una unidad que contiene todas sus características y comportamientos en sí misma, lo cual lo hace como un todo independiente, pero que se interrelaciona con objetos de su misma clase o de otras clase, como sucede en el mundo real.

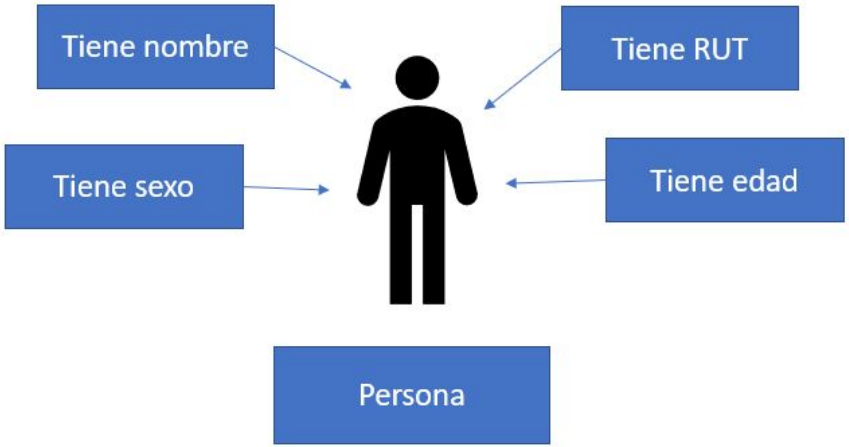
# Objeto

Universidad Andrés Bello

- Son abstracciones de datos.
- Cada objeto tiene una identidad, un tipo y un valor.
- Ejemplo: perro, coche, auto, persona, etc.

# Objeto

- Son abstracciones de datos.
- Cada objeto tiene una identidad, un tipo y un valor.
- Ejemplo: perro, coche, auto, persona, etc.




```
class Persona:
    nombre = "Sam
    sexo = "Femen
    Edad = 28
    RUT = 1802154
```

# Atributos

- Son las características que puede tener un objeto.
- Los atributos describen el estado de un objeto y pueden ser de cualquier tipo de dato.

```
class Persona():
    nombre = 'Samantha'
    sexo = 'Femenino'
    edad = 28
    rut = 180215417
```

```
P1 = Persona
print(P1.nombre)
```

```
Samantha
```

# Métodos

- Es el comportamiento de los objetos de una clase. Estos representan las operaciones que se pueden realizar con los objetos de la clase.

```
class Persona():
    nombre = 'Samantha'
    sexo = 'Femenino'
    edad = 28
    rut = 180215417

    def hablar(self,mensaje):
        return mensaje
```

```
print(Persona().hablar("Hola, mi nombre es {0}.".format(P1.nombre)))
print(type(Persona().hablar))
```

```
Hola, mi nombre es Samantha.
<class 'method'>
```

# Método constructor

Universidad Andrés Bello

- Método que le da el estado inicial a una clase.

```
class Persona():
    def __init__(self,nombre,sexo):
        self.nombre = nombre
        self.sexo = sexo

Profesor = Persona("Matias", "Masculino")
```

```
print("El profesor de nombre {} tiene sexo {}".format(Profesor.nombre,Profesor.sexo))
```

```
El profesor de nombre Matias tiene sexo Masculino
```

# Método string

- Sirve para dar una representación textual a un objeto en forma de cadena.

```
class Persona():
    def __init__(self,nombre,sexo):
        self.nombre = nombre
        self.sexo = sexo

    def __str__(self):
        return """\
Nombre: {}
Sexo: {}""".format(self.nombre,self.sexo)

Profesor = Persona("Nicolás", "Masculino")

print(Profesor)
```

```
Nombre: Nicolás
Sexo: Masculino
```

# Otros método string

Universidad Andrés Bello

Para operadores

- __add__(): Define el comportamiento al aplicar el operador +
- __sub__(): Define el comportamiento al aplicar el operador -
- __mul__(): Define el comportamiento al aplicar el operador *
- __truediv__(): Define el comportamiento al aplicar el operador /

Para comparaciones

- __lt__() para a < b.
- __gt__() para a > b.
- __le__() para a <= b.
- __ge__() para a >= b.
- __ne__() para a != b.
- __eq__() para a == b.

# Herencia

- Permite crear un objeto a partir de uno ya existente.
- El nuevo objeto hereda todas las cualidades del objeto del que deriva y además puede añadir nuevas funcionalidades o modificar las ya existentes.

```python
class Animal():

    def __init__(self, nombre, color, sexo):
        #Constructor clase animal
        self.nombre = nombre
        self.color = color
        self.sexo = sexo

    def getNombre(self):
        return self.nombre

    def getColor(self):
        return self.color

    def getSexo(self):
        return self.sexo

class Gato(Animal):

    #Invoca al constructor de la clase Gato
    def __init__(self,nombre,color,sexo,raza):

        # Invoca al constructor de clase Persona
        Animal.__init__(self, nombre, color, sexo)
        #Nuevos atributos
        self.raza = raza

    def Descripcion(self):
        return self.getNombre(), self.getColor(), self.getSexo(),self.raza

a1 = Gato("Mocka","Multicolor","Femenino","Calico")

print(a1.Descripcion())
print(a1.getNombre())
```

```
('Mocka', 'Multicolor', 'Femenino', 'Calico')
Mocka
```

# Herencia

- Permite crear un objeto a partir de uno ya existente.
- El nuevo objeto hereda todas las cualidades del objeto del que deriva y además puede añadir nuevas funcionalidades o modificar las ya existentes.

Universidad Andrés Bello

```python
class Gato(Animal):

    #Invoca al constructor de la clase Gato
    def __init__(self,nombre,color,sexo,raza):

        # Invoca al constructor de clase Persona
        Animal.__init__(self, nombre, color, sexo)
        #Nuevos atributos
        self.raza = raza

    def Descripcion(self):
        return self.getNombre(), self.getColor(), self.getSexo(),self.raza

    def Maullar(self):
        print("Miau.")
        if self.sexo == "Masculino":
            print ("No puelo toy chikito.")
        else:
            print ("No puelo toy chikita.")

a1 = Gato("Mocka","Multicolor","Femenino","Calico")

print(a1.Maullar())
```

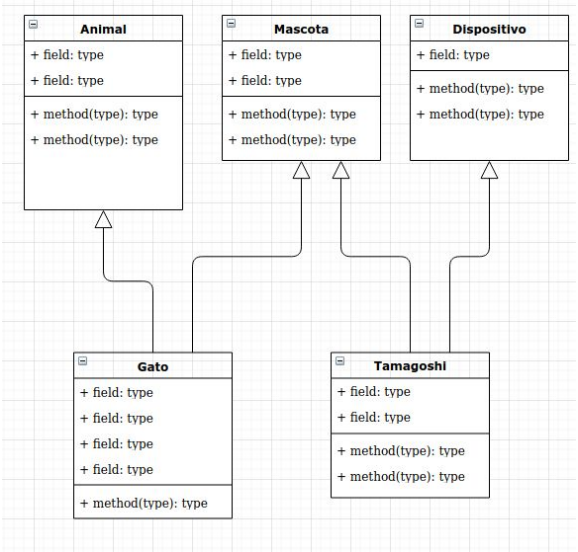
```
Miau.
No puelo toy chikita.
```

# Herencia Múltiple

- Podríamos hacer que una clase adquiera el comportamiento y las propiedades de otras dos clases
-

```
class Gato(Animal, Mascota):
    def __init__(self, nombre, color, sexo, raza):
        #invoca al constructor de la clase Animal
        Animal.__init__(self, nombre, color, sexo)
        Mascota.__init__(self, nombre)

    .........
```



# Polimorfismo

Universidad Andrés Bello

- Es la capacidad de tomar más de una forma.
- Una operación puede presentar diferentes comportamientos en diferentes instancias.
- El comportamiento depende de los tipos de datos utilizados en la operación.

```
class Ayudante1():
    nombre = "Samantha"
    sexo = "Femenino"

    def hablar(self):
        print("Hola, mi nombre es Samantha.")

class Ayudante2():
    nombre = "Marco"
    sexo = "Masculino"

    def hablar(self):
        print("Hola, mi nombre es Marco.")

def escucharPersona(persona):
    persona.hablar()

p1 = Ayudante1()
p2 = Ayudante2()
escucharPersona(p1)
escucharPersona(p2)
```

```
Hola, mi nombre es Samantha.
Hola, mi nombre es Marco.
```