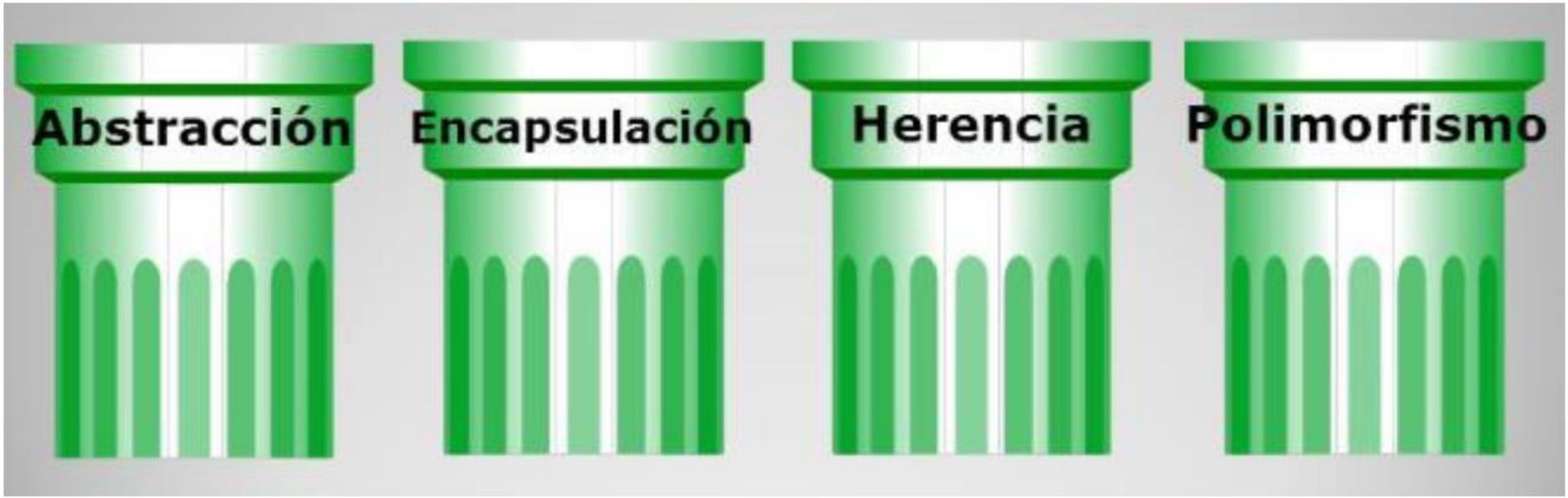
PROFUNDIZACION
Programación Orientada a Objetos

Pilares de la programación orientada a objetos



Clases

- Una clase es la **abstracción** de la realidad en memoria del computador
- Un objeto es una instancia en particular de una clase

Automotor

- Clase:
- Placa:
- Llantas:
- Puertas:
- Sillas:

Clase: Automovil
Placa: AAA001
Llantas: 4
Puertas: 2
Sillas: 5

Clase: Motocicleta
Placa: AAA002
Llantas: 2
Puertas: 0
Sillas: 2

Clase: Autobús
Placa: AAA003
Llantas: 4
Puertas: 2
Sillas: 40

Clases

```
class Automotor:
    clase=""
    placa=""
    llantas=-1
    puertas=-1
    sillas=-1
```

```
a1 = Automotor()
```

```
a1.clase = "automotor"
a1.placa = "AAA001"
a1.llantas = 4
a1.puertas = 2
a1.sillas = 4
```

a1 es una instancia de Automotor

Clases

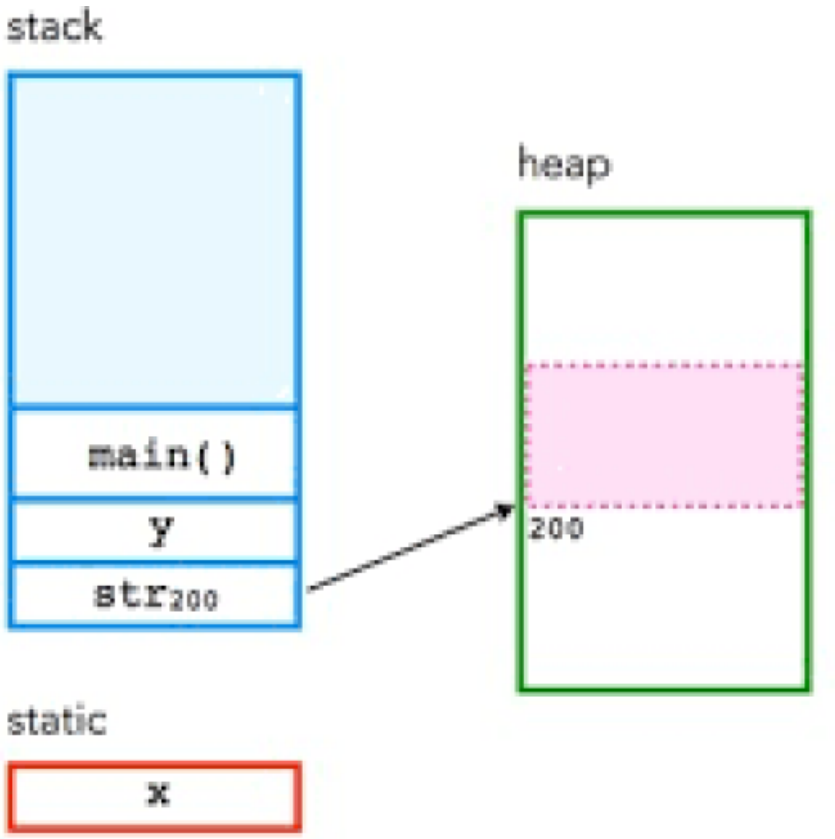
```
a1.clase = "automotor"
a1.placa = "AAA001"
a1.llantas = 4
a1.puertas = 2
a1.sillas = 4
```

```
a2.clase = "motocicleta"
a2.placa = "AAA002"
a2.llantas = 2
a2.puertas = 0
a2.sillas = 2
```

```
a3.clase = "autobus"
a3.placa = "AAA003"
a3.llantas = 4
a3.puertas = 2
a3.sillas = 40
```

Garbage collector

```
1  class Persona():
2      nombre = ""
3      apellido = ""
4
5  p = Persona()
6  p.nombre = "Guillermo"
7  p.apellido = "De Mendoza"
8
9  print("Persona: %s %s"%(p.nombre,p.apellido))
10
11 p = None
12
```



Si un objeto no es referencia, será recuperable por el garbage collector para liberar memoria RAM

Metodos

- Todos los métodos de una clase deben iniciar con **self**
- **self** indica que se quiere acceder a un elemento de la misma clase/objeto

```
class Persona:

    __nombre = "Guillermo"

    def getNombre(self):
        return self.__nombre

    def setNombre(self,nombre):
        self.__nombre = nombre

p = Persona()
print(p.getNombre())
```

```
Guillermo
```

```
class Persona:

    __nombre = "Guillermo"

    def getNombre(self):
        return self.__nombre

    def setNombre(self,nombre):
        self.__nombre = nombre

p = Persona()
p.setNombre("Mario")
print(p.getNombre())
```

```
Mario
```

Constructor

Un constructor es el llamado de un método de la clase al momento de instanciar un nuevo objeto

Por defecto toda clase tien un constructor vacio, a no ser que se defina uno:

__init__

```
class Persona:

    __nombre = "Guillermo"

    def getNombre(self):
        return self.__nombre

    def setNombre(self,nombre):
        self.__nombre = nombre

p = Persona()
```

Campos privados - encapsulación

- Un parámetro o método privado inicia con __
- Solo pueden ser accedidos por la clase

```
class Persona:
    __nombre = "Guillermo"

p = Persona()
print(p.__nombre)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-10-2d0c5b7c41e0> in <module>
      3
      4 p = Persona()
----> 5 print(p.__nombre)

AttributeError: 'Persona' object has no attribute '__nombre'
```

Constructor

Ejemplo 1

```
class Persona:

    def __init__(self,nombre):
        self.__nombre = nombre

    def getNombre(self):
        return self.__nombre

    def setNombre(self,nombre):
        self.__nombre = nombre

p = Persona("Mario Bross")
print(p.getNombre())
```

```
Mario Bross
```

Ejemplo 2

```
class Carro:

    def __init__(self,puertas,llantas,puestos):
        self.puertas = puertas
        self.llantas = llantas
        self.puestos = puestos

carro1 = Carro(2,4,5)
print(carro1.puertas)
print(carro1.llantas)
print(carro1.puestos)
```

```
2
4
5
```


Equals

Permite comparar si dos clases son iguales o diferentes

__eq__

```
class Person:

    def __init__(self,name,age):
        self.name = name
        self.age = age

    def __eq__(self,other):
        if isinstance(other, self.__class__):
            if other.name == self.name and other.age == self.age:
                return True
            else:
                return False
        else:
            return False

p1 = Person('Guillermo',32)
p2 = Person('Guillermo',31)

print(p1==p2)

p1 = Person('Guillermo',32)
p2 = Person('Guillermo',32)

print(p1==p2)
```

```
False
True
```


toString

Permite obtener un mensaje personalizado al imprimir una clase

__str__

```
class Person:

    def __init__(self,name,age):
        self.name = name
        self.age = age

    def __eq__(self,other):
        if isinstance(other, self.__class__):
            if other.name == self.name and other.age == self.age:
                return True
            else:
                return False
        else:
            return False

    def __str__(self):
        return "[name: "+str(self.name)+", age: "+str(self.age)+"]"

p1 = Person('Guillermo',32)
print(p1)
```

```
[name: Guillermo, age: 32]
```


Representacion

Permite que al tener un “collection” y este deba ser impreso, la clase retorne un mensaje personalizado

__repr__

```
class Person:

    def __init__(self,name,age):
        self.name = name
        self.age = age

    def __eq__(self,other):
        if isinstance(other, self.__class__):
            if other.name == self.name and other.age == self.age:
                return True
            else:
                return False
        else:
            return False

    def __str__(self):
        return "[name: "+str(self.name)+", age: "+str(self.age)+"]"

    def __repr__(self):
        return "[name: "+str(self.name)+", age: "+str(self.age)+"]"

p1 = Person('A',0)
p2 = Person('B',10)
p3 = Person('C',5)
p4 = Person('D',2)
p5 = Person('F',7)
p6 = Person('G',1)
personList = [p1,p2,p3,p4,p5,p6]

print(personList)
```

```
[[name: A, age: 0], [name: B, age: 10], [name: C, age: 5], [name: D, age: 2], [name: F, age: 7], [name: G, age: 1]]
```


Comparador “less than” o “greather than”

Permite comparar dos clases y saber si una tiene un valor superior o inferior que otra, bastante útil en ordenamientos

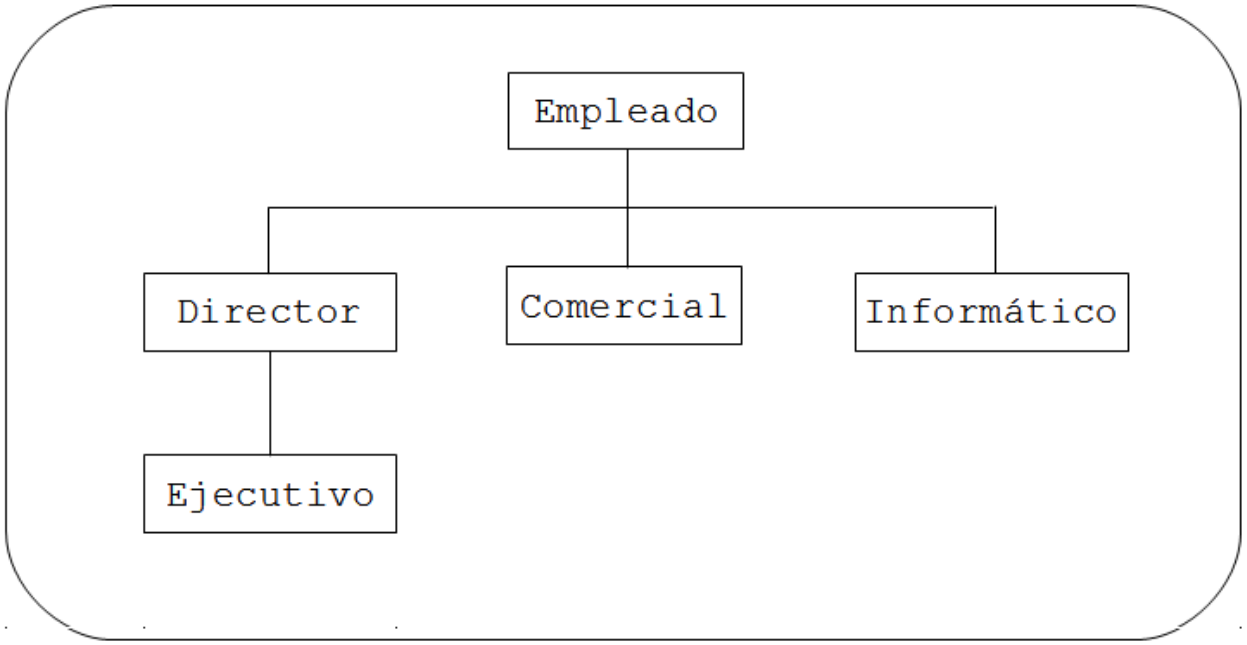
__lt__ , __gt__

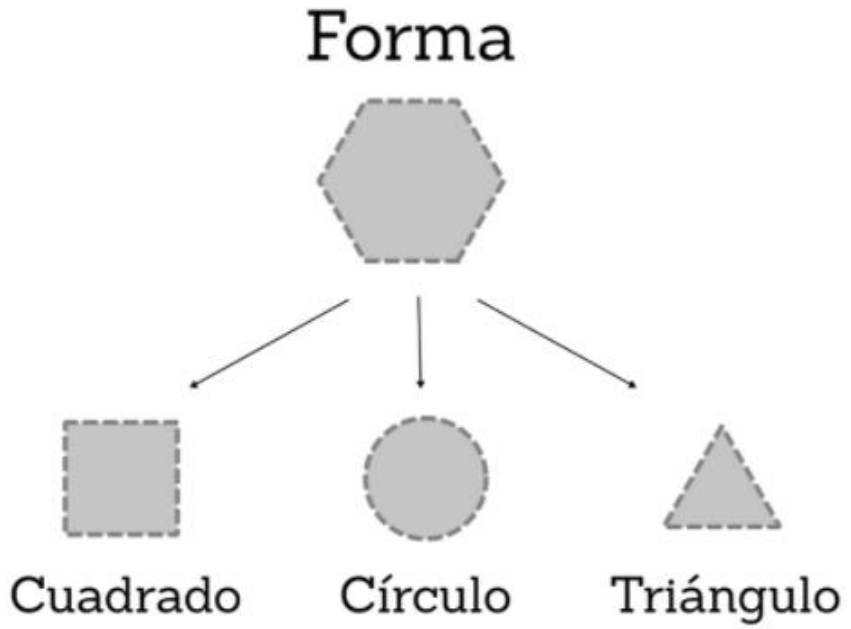
```python
class Person:
    def __init__(self,name,age):
        self.name = name
        self.age = age
    def __eq__(self,other):
        if isinstance(other, self.__class__):
            if other.name == self.name and other.age == self.age:
                return True
            else:
                return False
        else:
            return False
    def __str__(self):
        return "[name: "+str(self.name)+", age: "+str(self.age)+"]"
    def __repr__(self):
        return "[name: "+str(self.name)+", age: "+str(self.age)+"]"
    def __lt__(self,other):
        if(self.age - other.age < 0):
            return True
        return False
    def __gt__(self,other):
        if(self.age - other.age > 0):
            return True
        return False
```

```python
p1 = Person('A',0)
p2 = Person('B',10)
p3 = Person('C',5)
p4 = Person('D',2)
p5 = Person('F',7)
p6 = Person('G',1)

personList = [p1,p2,p3,p4,p5,p6]

print("Normal: %s"%(personList))

personList.sort()

print("Sorted: %s"%(personList))
```

Herencia

Permite reutilizar código, al hacer que una clase “hijo” herede todo el código de una clase “padre





Cada clase puede incluso tener su propia definición de sus métodos !

Herencia

```
class Empleado:
    def __init__(self,nombre,apellido,puesto,salario):
        self.__nombre = nombre
        self.__apellido = apellido
        self.__puesto = puesto
        self.__salario = salario
    def getNombre(self):
        return self.__nombre
    def getApellido(self):
        return self.__apellido
    def getPuesto(self):
        return self.__puesto
    def getSalario(self):
        return self.__salario
```

```
class Recepcionista(Empleado):
    def __init__(self,nombre,apellido,salario):
        Empleado.__init__(self,nombre,apellido,"recepcionista",salario)
    def pagarSueldo(self):
        salario = Empleado.getSalario(self)
        puesto = Empleado.getPuesto(self)
        print("Pagando %d a %s "%(salario,puesto))

class Ingeniero(Empleado):
    def __init__(self,nombre,apellido,salario,bonificacion):
        Empleado.__init__(self,nombre,apellido,"ingeniero",salario)
        self.__bonificacion = bonificacion
    def pagarSueldo(self):
        salario = Empleado.getSalario(self) + self.__bonificacion
        puesto = Empleado.getPuesto(self)
        print("Pagando %d a %s "%(salario,puesto))

class Vendedor(Empleado):
    def __init__(self,nombre,apellido,salario,ventas,comision):
        Empleado.__init__(self,nombre,apellido,"vendedor",salario)
        self.__ventas = ventas
        self.__comision = comision
    def pagarSueldo(self):
        salario = Empleado.getSalario(self) + self.__ventas*self.__comision
        puesto = Empleado.getPuesto(self)
        print("Pagando %d a %s "%(salario,puesto))
```


Herencia

```
class Recepcionista(Empleado):
    def __init__(self,nombre,apellido,salario):
        Empleado.__init__(self,nombre,apellido,"recepcionista",salario)
    def pagarSueldo(self):
        salario = Empleado.getSalario(self)
        puesto = Empleado.getPuesto(self)
        print("Pagando %d a %s "%(salario,puesto))

class Ingeniero(Empleado):
    def __init__(self,nombre,apellido,salario,bonificacion):
        Empleado.__init__(self,nombre,apellido,"ingeniero",salario)
        self.__bonificacion = bonificacion
    def pagarSueldo(self):
        salario = Empleado.getSalario(self) + self.__bonificacion
        puesto = Empleado.getPuesto(self)
        print("Pagando %d a %s "%(salario,puesto))

class Vendedor(Empleado):
    def __init__(self,nombre,apellido,salario,ventas,comision):
        Empleado.__init__(self,nombre,apellido,"vendedor",salario)
        self.__ventas = ventas
        self.__comision = comision
    def pagarSueldo(self):
        salario = Empleado.getSalario(self) + self.__ventas*self.__comision
        puesto = Empleado.getPuesto(self)
        print("Pagando %d a %s "%(salario,puesto))
```

```
empleado1 = Recepcionista("Maria","Simona",1000)
empleado1.pagarSueldo()
```

```
Pagando 1000 a recepcionista
```

```
empleado2 = Ingeniero("Alejandra","Gomez",2000,300)
empleado2.pagarSueldo()
```

```
Pagando 2300 a ingeniero
```

```
empleado3 = Vendedor("Carolina","Abril",500,30,300)
empleado3.pagarSueldo()
```

```
Pagando 9500 a vendedor
```


Polimorfismo

```
empleado1 = Recepcionista("Maria","Simona",1000)
empleado2 = Ingeniero("Alejandra","Gomez",2000,300)
empleado3 = Vendedor("Carolina","Abril",500,30,300)

empleados = [empleado1,empleado2,empleado3]

for e in empleados:
    e.pagarSueldo()
```

```
Pagando 1000 a recepcionista
Pagando 2300 a ingeniero
Pagando 9500 a vendedor
```