Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

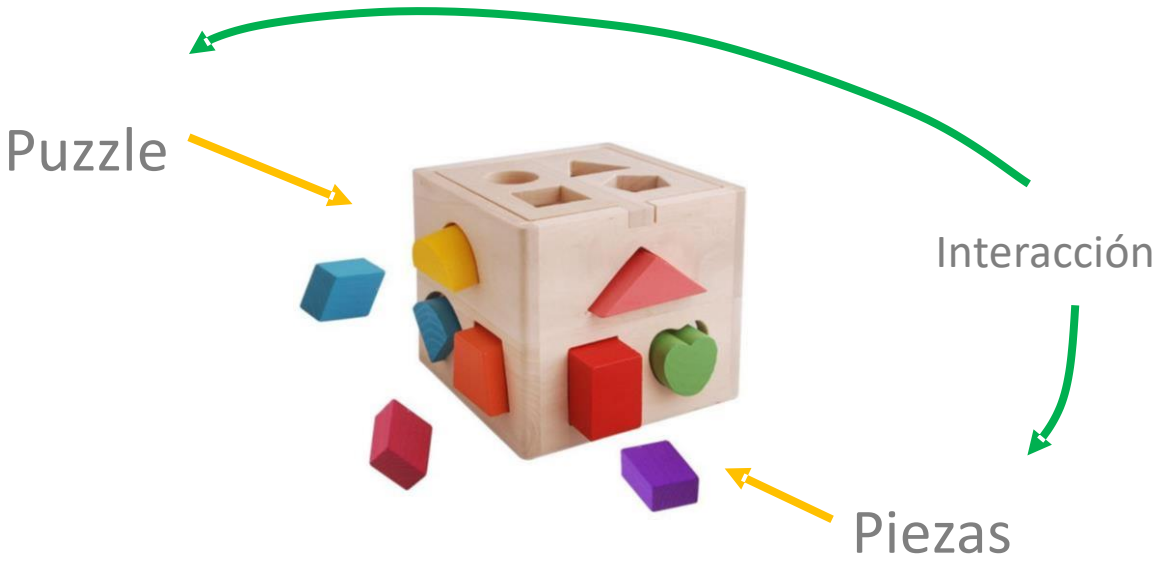
IIC2115 - Programación como herramienta para la ingeniería

Fundamentos de Programación Orientada a Objetos (OOP)

Profesor: Hans Löbel

¿Qué son los objetos?

En el mundo real los objetos son elementos tangibles que se pueden manipular y sentir, que representan algo que tiene significado para nosotros.

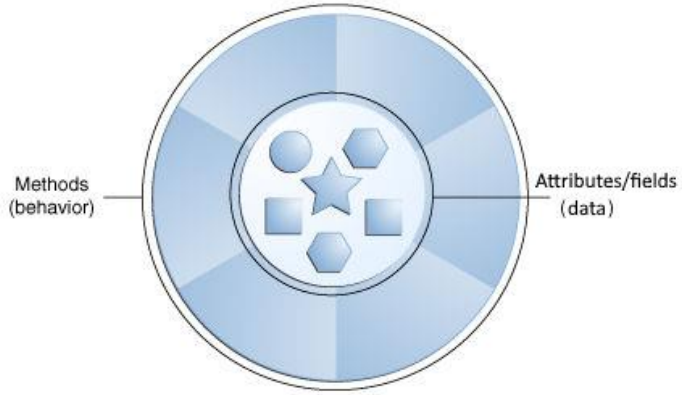


¿Qué son los objetos?

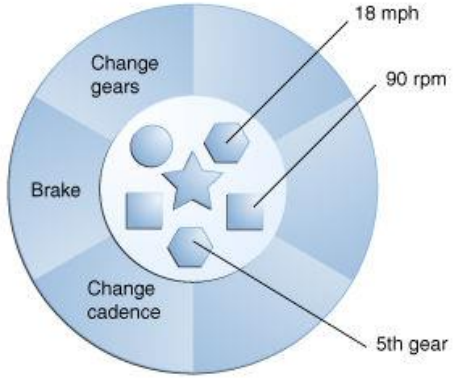
En el área de desarrollo de software, un objeto es una colección de datos que además tiene asociado comportamientos.

- Datos: describen el estado de los objetos. Se les conoce como atributos o campos del objeto.

- Comportamientos: representan acciones que realiza el objeto, o realizan sobre él, que pueden generar cambios en su estado. Se les conoce como métodos del objeto.



(a) A software object



(b) Bicycle modelled as a software object

Ejemplo: datos y comportamiento

Objeto: Auto	
Datos	**Comportamiento**
Marca	Calcular próxima mantención
Modelo	Calcular distancia a alguna dirección
Color	Pintarlo de otro color
Año	Realizar nueva mantención
Motor	
Kilometraje	
Ubicación Actual	
Mantenciones	

¿Qué es entonces OOP?

La programación orientada a objetos (OOP) quiere decir que los programas estarán orientados a modelar las funcionalidades a través de la interacción entre objetos por medio de sus datos y comportamiento.

```python
class Departamento:
    def __init__(self, _id, mts2, valor, num_dorms, num_banos):
        self._id = _id
        self.mts2 = mts2
        self.valor = valor
        self.num_dorms = num_dorms
        self.num_banos = num_banos
        self.vendido = False

    def vender(self):
        if not self.vendido:
            self.vendido = True
        else:
            print("Departamento {} ya se vendió".format(self._id))
```

Clases vs objetos

En OOP, los objetos son descritos por clases

Cada objeto es una instancia de la clase Auto

Objeto 1

Objeto 2

Objeto 3

Clase Auto

```
1 d1 = Departamento(_id=1, mts2=100, valor=5000, num_dorms=3, num_banos=2)
2 print(d1.vendido)
3 d1.vender()
4 print(d1.vendido)
5 d1.vender()
```

Interfaz

Interface
Turn on Turn off Volume up Volume down Switch to next channel Switch to previous channel
Current channel Volume level

Encapsulamiento

Existen atributos de los objetos que no necesitan ser visualizados ni accedidos por los otros objetos con que se interactúa.

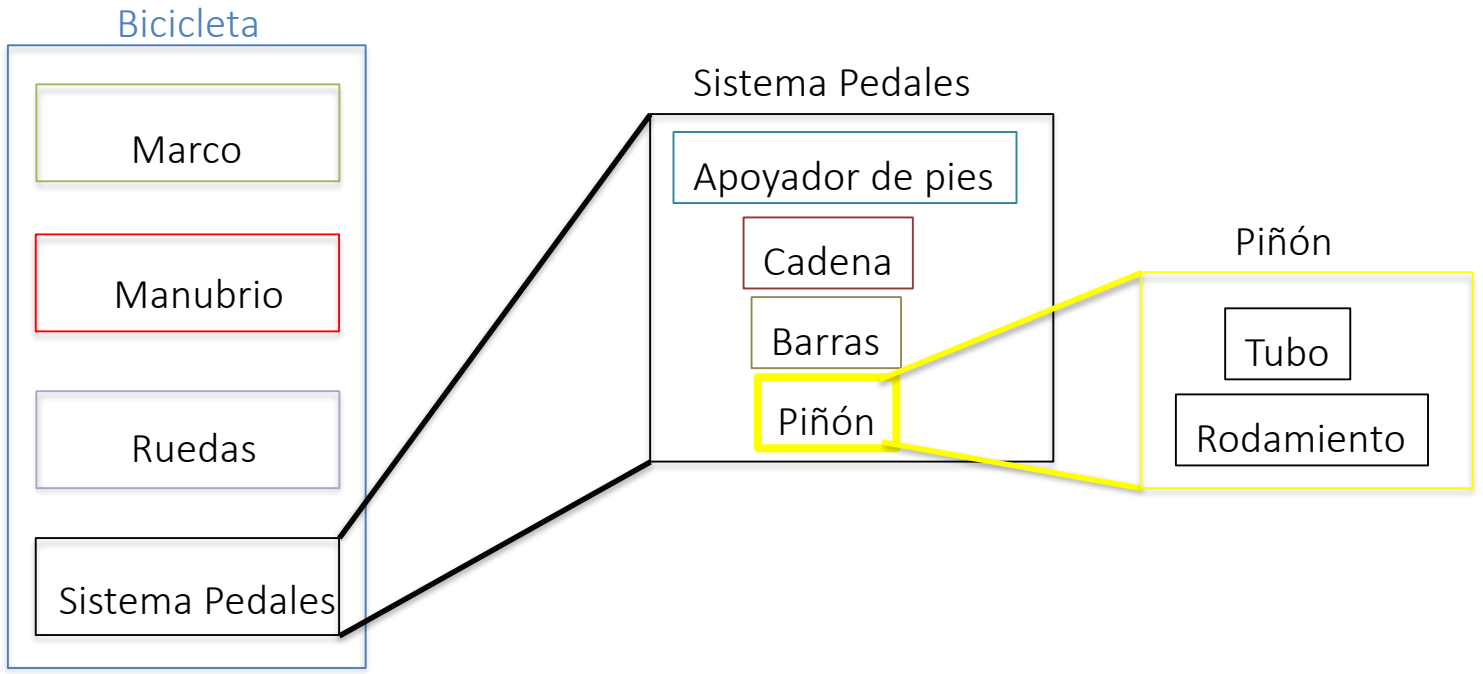
```python
class Televisor:
    ''' Clase que modela un televisor.
    '''

    def __init__(self, pulgadas, marca):
        self.pulgadas = pulgadas
        self.marca = marca
        self.encendido = False
        self.canal_actual = 0

    def encender(self):
        self.encendido = True

    def apagar(self):
        self.encendido = False

    def cambiar_canal(self, nuevo_canal):
        self._codificar_imagen()
        self.canal_actual = nuevo_canal

    def __codificar_imagen(self):
        print("Estoy convirtiendo una señal eléctrica en la imagen que estás viendo.")
```

Abstracción en OOP

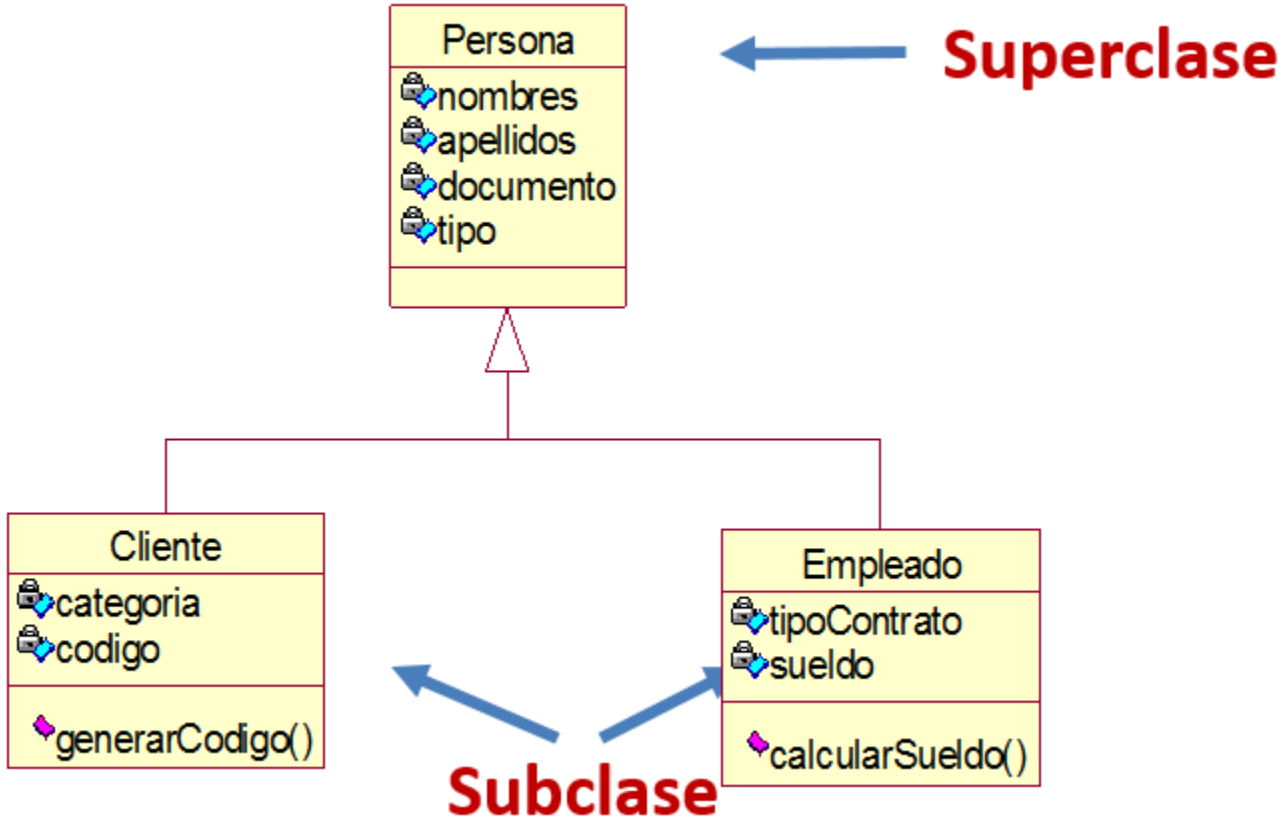
- Proceso de extracción de la interfaz de un objeto a partir de sus detalles internos.
- El nivel de detalle de la interfaz se denomina abstracción.

Vendedor
Nombre
Nº autos vendidos
Comisión asignada

También es posible modelar objetos como atributos de otros objetos, mediante composición



Herencia nos permite modelar clases similares
sin reescribir todo de nuevo



```python
import numpy as np

class Variable:
    def __init__(self, data):
        self.data = np.array(data)

    def representante(self):
        pass

class Ingresos(Variable):
    def representante(self):
        return np.mean(self.data)

class Comuna(Variable):
    def representante(self):
        ind = np.argmax([np.sum(self.data == c) for c in self.data])  # el que mas se repite
        return self.data[ind]

class Puesto(Variable):
    categorias = {'Gerente': 1, 'SubGerente': 2, 'Analista': 3,
                  'Alumno en Practica': 4} # class (or static) variable

    def representante(self):
        return self.data[np.argmin([Puesto.categorias[c] for c in self.data])]#la categoria mas alta acorde con el diccionario
```

Composición y herencia

- NO TIENEN NADA QUE VER!
- Si bien ambos son mecanismo para modelar, estructuralmente difieren de manera fundamental.
- Herencia busca facilitar la especialización de las clases, sin requerir repetir código.
- Composición busca facilitar aumentar el nivel de abstracción de las clases, al permitir tipos de dato complejos (otras clases) como atributos.

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2115 - Programación como herramienta para la ingeniería

Fundamentos de Programación Orientada a Objetos

Profesor: Hans Löbel