

METODOLOGÍAS DE PROGRAMACIÓN I

Repaso del paradigma de la programación orientada a objetos

- Clases
- Composición
- Herencia

Temario

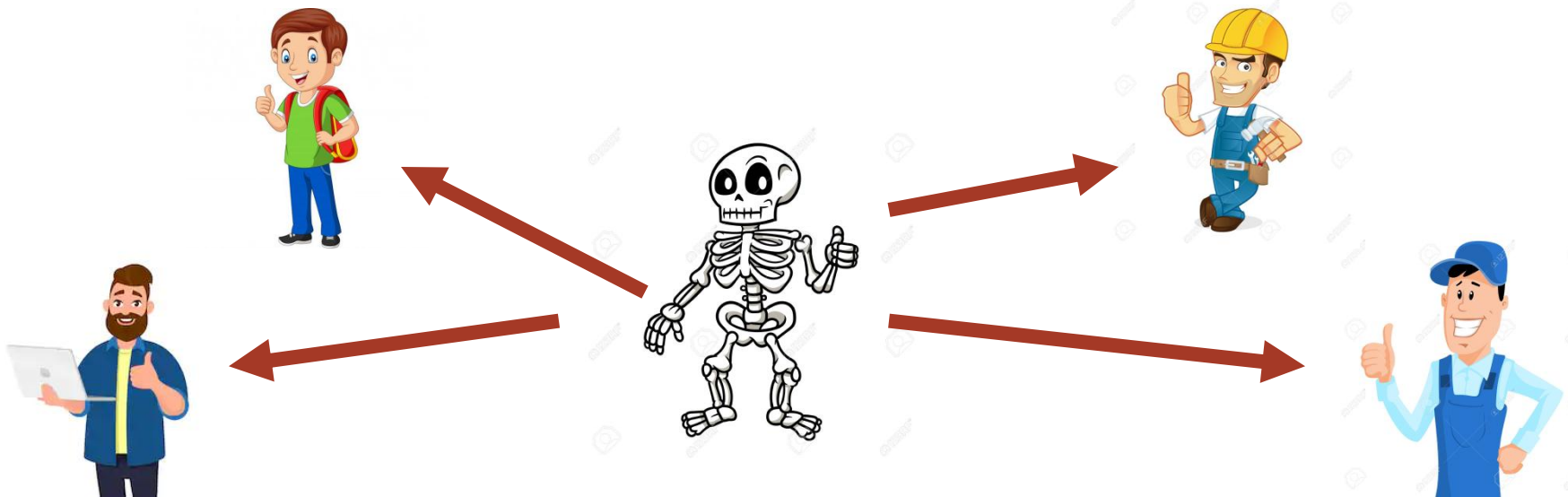
- Programación orientada a objetos
 - Objeto
 - Clase
 - Instancia
 - Composición
 - Herencia

¿Recuerdan qué es un objeto?

- Un objeto es una representación de cualquier entidad del mundo real: un perro, un escritorio, un alumno, una bicicleta, etc.
- Todo objeto posee dos características:
 - Estado (variables)
 - Comportamiento (funciones)

¿Recuerdan qué es una clase?

- Una clase es una construcción estática que describe un comportamiento común y atributos que toman distintos estados.
- Su formalización es a través de una estructura que incluye datos y funciones, llamadas métodos. Los métodos son los que definen el comportamiento.



Clase - Ejemplo

¿Cuál es el estado y el comportamiento que tienen en común todos los objetos (personas)?



Estado

Nombre

Edad

DNI

Peso

Comportamiento

Saludar

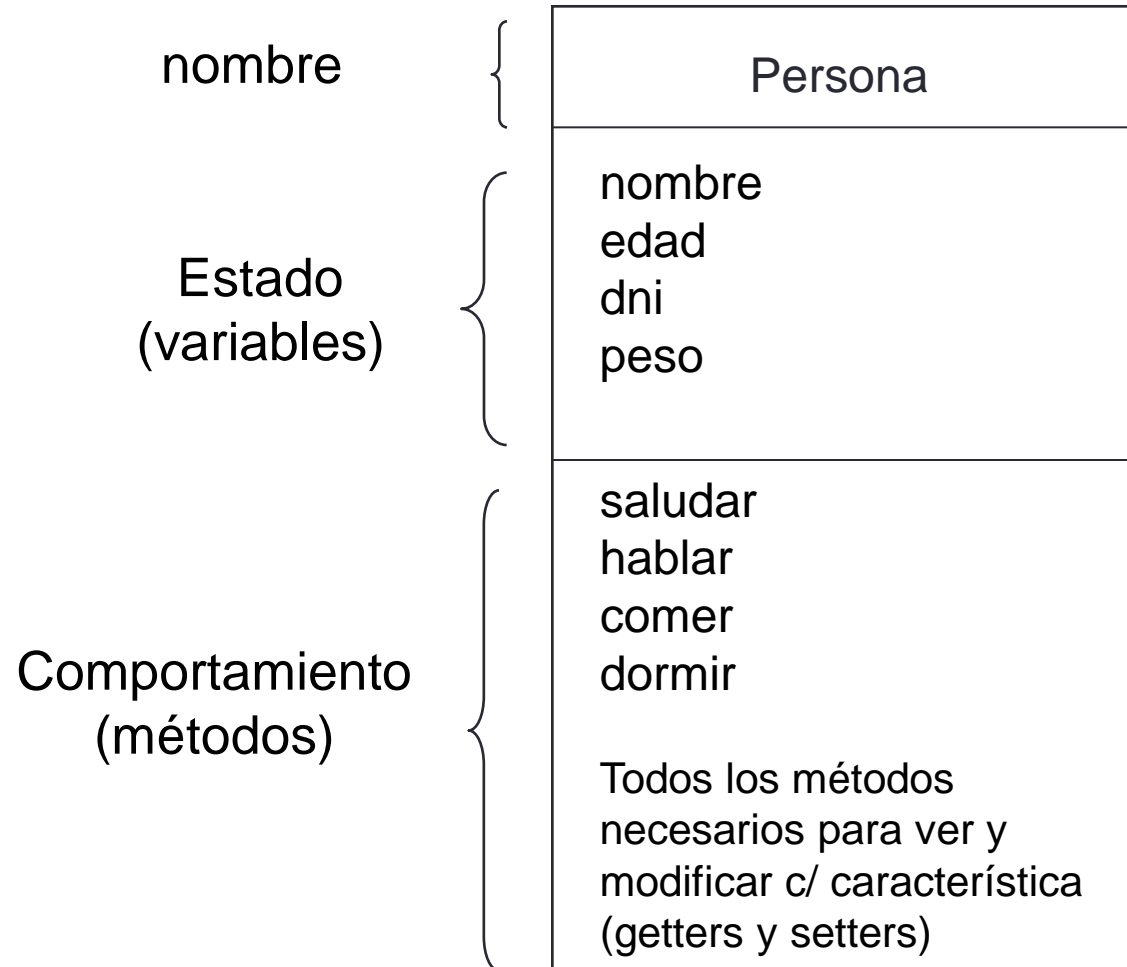
Hablar

Comer

Dormir

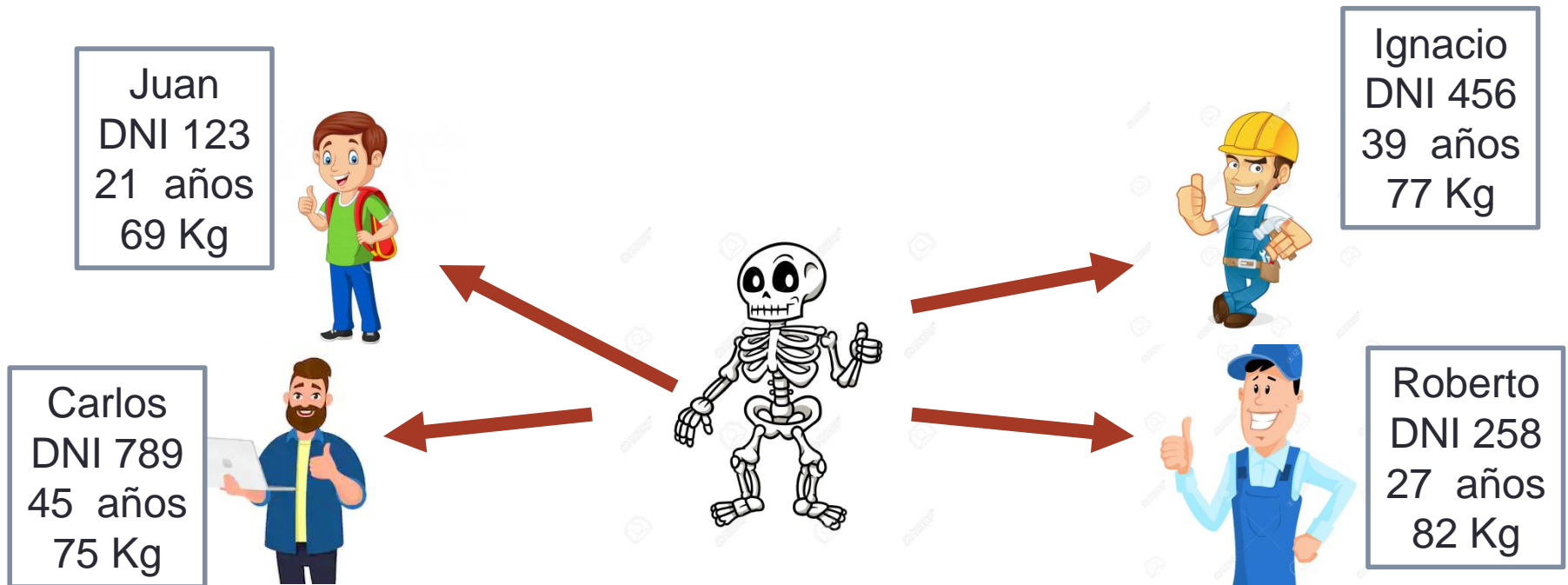
Clase de objetos

Diagrama de clases UML



¿Qué es una instancia?

- Instancia es la estructura de datos que tiene "vida".
- Todas las instancias de una misma clase tienen el mismo estado y el mismo comportamiento.
- Cada instancia tiene sus propios valores para cada estado



Instanciando objetos

- Para poder usar objetos hay que ***instanciar*** una clase.
- Las **instancias** se crean haciendo referencia a la clase que se desee instanciar y deben ser almacenadas en una variable para luego poder referenciarla, por ejemplo:

per1 = **new** Persona(); // ← En C#

per1 = Persona() # ← En Python

- Se pueden crear tantas instancias de una clase como se necesite.

Trabajando con instancias

per1.saludar()

per2.hablar()

~~per3.nombre = "Alberto"~~

per3.setNombre("Alberto")

nombre = per3.getNombre()

per4.setNombre(per5.getNombre())

Constructores

- Los constructores son funciones especiales que permiten la inicialización interna de una instancia.
- En los constructores se pueden asignar valores por defecto al estado interno de un objeto.
- También pueden recibir valores por parámetro para ser utilizados como estado inicial.

```
// En c#  
Persona(string nom) {  
    this.nombre = nom;  
    this.edad = 18;  
}  
per1 = new Persona("Oscar");
```

```
# En Python  
def __init__(self, nom):  
    self.nombre = nom  
    self.edad = 18  
  
per1 = Persona("Oscar")
```

- Es un buen lugar para crear estructuras de datos más complejas (arreglos, listas, etc.).

Miembros

- En POO hay dos tipos de miembros (variables, métodos)
 - De instancia
 - De clase

- En C# si queremos declarar miembros de clase usamos la palabra reservada **static**

```
public static void metodoDeClase() { . . . }
```

- En Python podemos usar la directiva **@staticmethod**

```
@staticmethod
```

```
def metodoDeClase():
```

```
    . . .
```

Miembros de instancia

- Los miembros de instancia, ya sean variables o métodos son utilizados cuando se trabaja con instancias

`per1.saludar()`

`per2.hablar()`

`per3.setEdad(50)`

- Los métodos de instancia pueden hacer referencia a las variables de instancia de la propia instancia.

Miembros de clase

- Los miembros de clase no pertenecen a ninguna instancia, pertenecen a la clase
- La referencia a un miembro de clase se hace mediante el nombre de clase

`Persona.metodoDeClase()`

- Los métodos de clases NO pueden hacer referencia a ningún miembro de instancia
- Una variable de clase es compartida por todas las instancias.

Modificadores de acceso

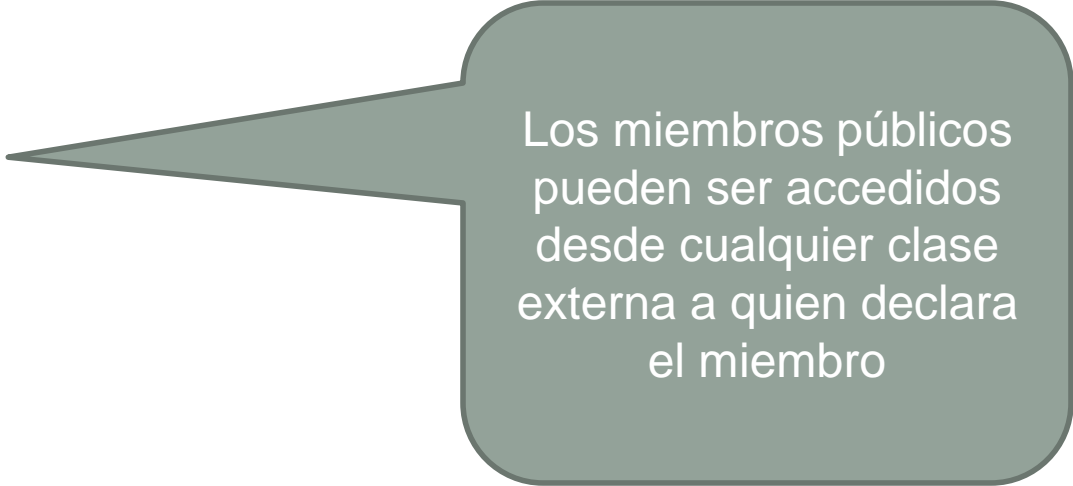
- En POO se habla de dos mecanismos para asegurar el encapsulamiento mediante los modificadores de acceso a miembros
- Los miembros de una clase pueden declararse como:
 - Públicos (public)
 - Privados (private)
 - Protegidos (protected)

Modificadores de acceso

- Miembros públicos
- Miembros privados
- Miembros protegidos

Modificadores de acceso

- Miembros públicos



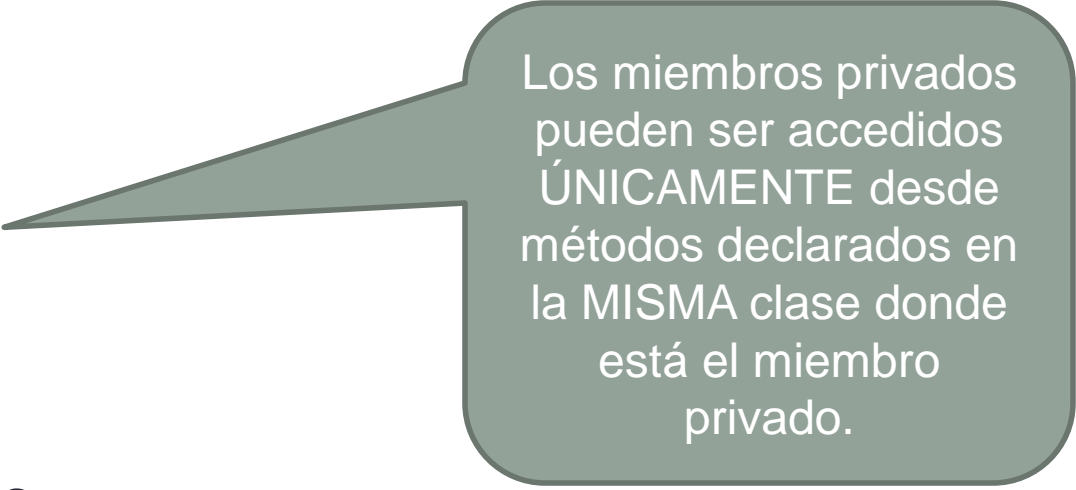
Los miembros públicos pueden ser accedidos desde cualquier clase externa a quien declara el miembro

- Miembros privados

- Miembros protegidos

Modificadores de acceso

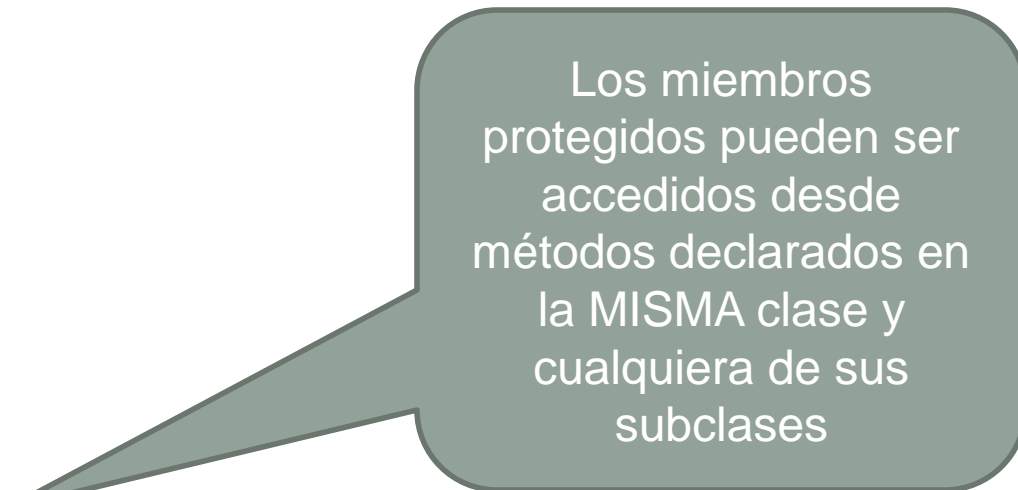
- Miembros públicos
- Miembros privados
- Miembros protegidos



Los miembros privados pueden ser accedidos ÚNICAMENTE desde métodos declarados en la MISMA clase donde está el miembro privado.

Modificadores de acceso

- Miembros públicos
- Miembros privados
- Miembros protegidos



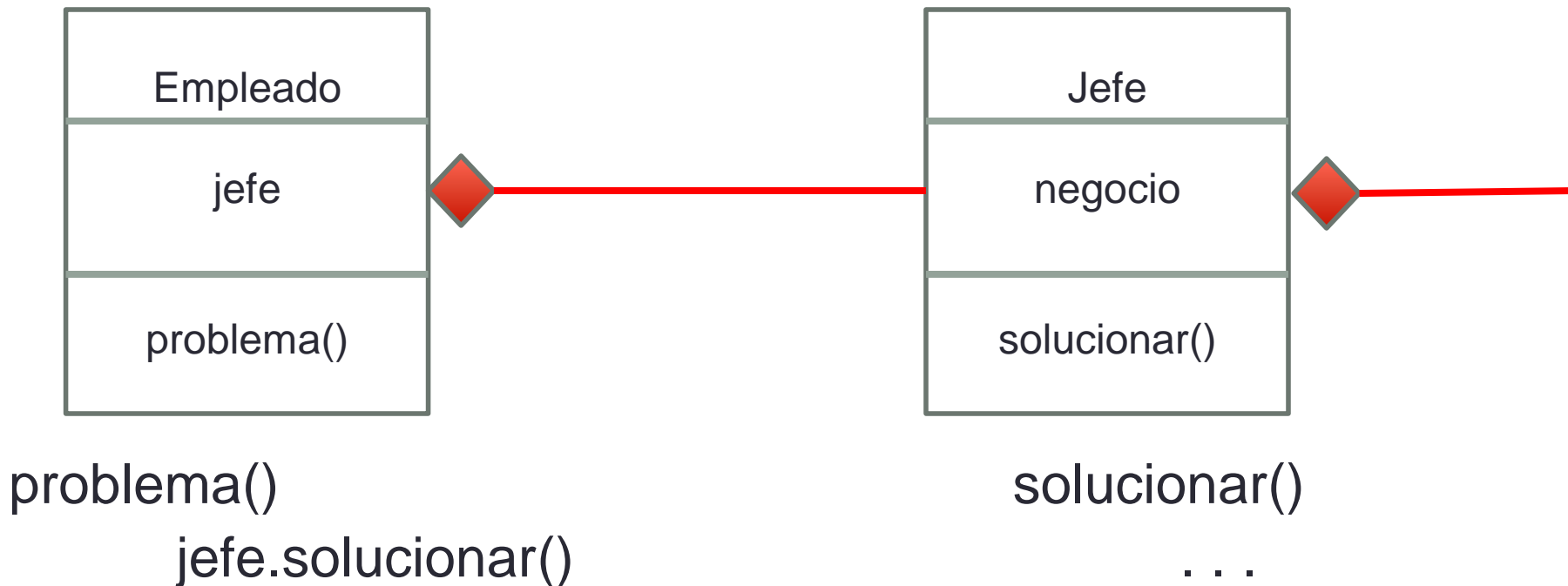
Los miembros protegidos pueden ser accedidos desde métodos declarados en la MISMA clase y cualquiera de sus subclases

Modificadores de acceso

- El estado de un objeto siempre lo vamos a declarar como privado o protegido.
- El caso de los constructores y métodos lo declararemos privados, protegidos o públicos según su función

Composición

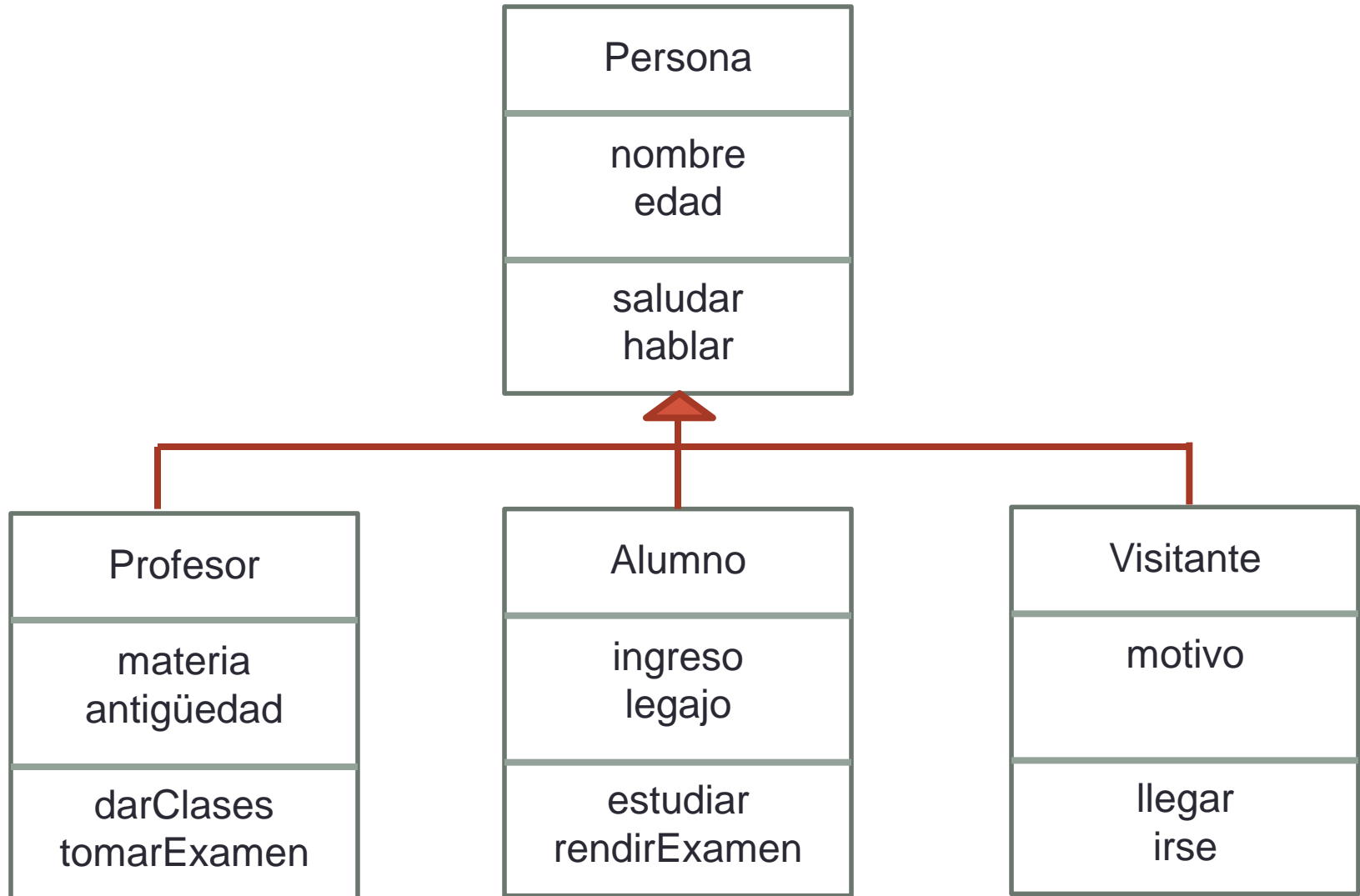
- La composición de objetos es la capacidad de que el estado de un objeto puede estar formado por otros objetos.



Herencia de clases

- La herencia permite el reuso de código facilitando la ampliación y mantenimiento de un diseño de objetos, minimizando la posibilidad de cometer errores.
- Todo el estado y comportamiento que tienen en común varios objetos se implementan en una clase "A" que denominamos **superclase**.
- Las características específicas de un objeto (que no tienen otros) se implementan en diferentes clases "B", "C", "D", etc. que denominamos **subclases**.
- La herencia de clases permite que "B", "C" y "D" **herede** todo el estado y comportamiento de la clase "A".

Herencia de clases - Ejemplo



Clases abstractas

- Una clase abstracta representa una abstracción de un objeto
- Con una clase abstracta tenemos un mecanismo de decir "QUE" es lo que tiene que hacer un objeto, sin especificar el "COMO" debe hacerlo
- Las clases abstractas NO se pueden instanciar

Método abstracto

- Un método abstracto especifica "QUE" es lo que debe hacer un objeto, sin decir "COMO" debe hacerlo.
- Los método abstractos NO tienen implementación.

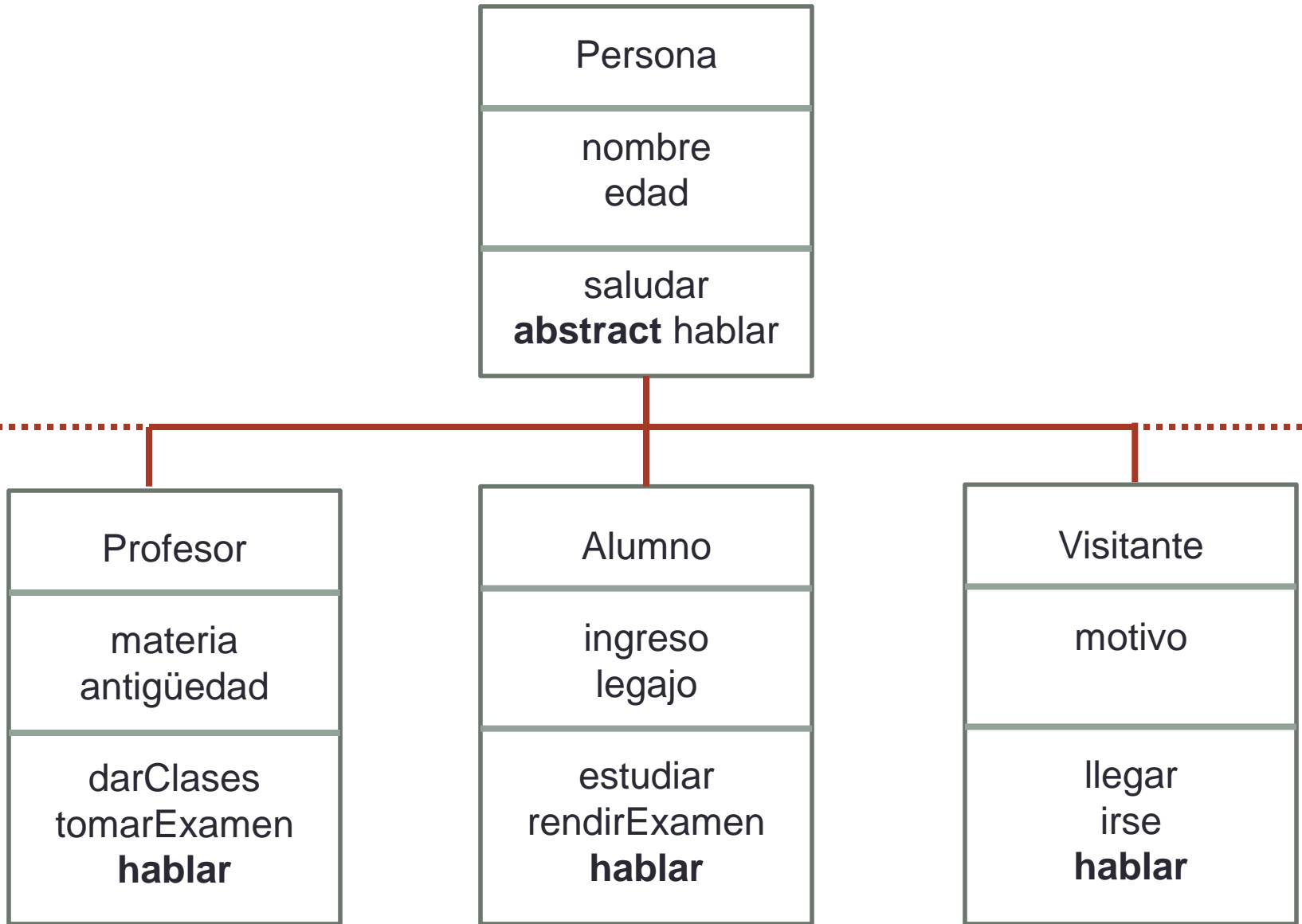
En C#:

```
abstract protected void hablar();
```

En Python

```
@abstractmethod  
def hablar(self):  
    pass
```


Herencia de clases - Ejemplo



Ejemplo

