

A graphic on the left side of the slide features four horizontal bars of increasing height, colored purple, orange, yellow, and blue from top to bottom. The text 'Agencia de Aprendizaje a lo largo de la vida' is written across these bars in white. The orange bar is shaped like an arrow pointing to the right.

Agencia de
Aprendizaje
a lo largo
de la vida

DJANGO

Clase 5

Python – Herencia y
encapsulamiento

Les damos la bienvenida

Vamos a comenzar a grabar la clase

Clase 05

Python – Herencia

- Clases y objetos, constructores, variables de instancia y de clase
- Visibilidad de atributos (público y privado)
- Generalización, herencia simple y múltiple
- Polimorfismo
- Clase abstractas

Clase 06

Python – Excepciones

- Manejo de excepciones
- Árbol de herencia de las excepciones
- Excepciones personalizadas
- Lanzando excepciones
- Buenas prácticas en el manejo de excepciones

¿Qué es la Herencia?

Es una técnica de los lenguajes de programación para construir una clase a partir de una o varias clases, compartiendo atributos y operaciones. Básicamente, la **herencia** es la **implementación** de la **generalización** en un **lenguaje de programación**



Es una relación entre clases en la que una clase comparte la estructura y/o el comportamiento definidos en una (herencia simple) o más clases (herencia múltiple)

¿Qué es el Encapsulamiento?

Es la característica que permite asegurar que el contenido de la información de un objeto está oculto al “mundo exterior”



El encapsulamiento, al separar el comportamiento de un objeto de su implementación interna, permite la modificación de este sin que se tengan que modificar las aplicaciones que lo utilizan.

BAJO ACOPLAMIENTO



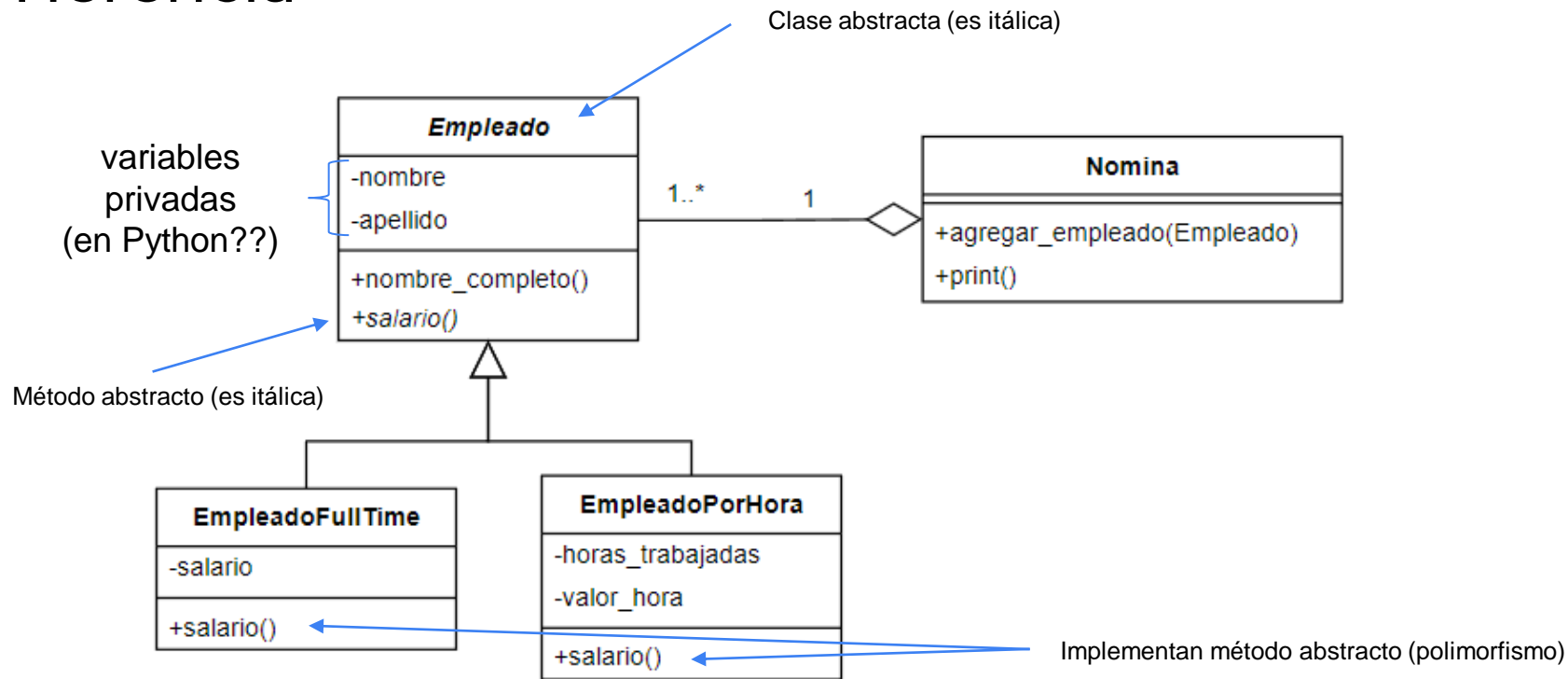
ALTA COHESIÓN



Herencia y encapsulamiento en Python



Herencia



Herencia y Polimorfismo

```
class Empleado(ABC):
    def __init__(self, nombre, apellido):
        self.__nombre = nombre
        self.__apellido = apellido

    @property
    def nombre_completo(self):
        return f"{self.__nombre} {self.__apellido}"

    @property
    @abstractmethod
    def salario(self):
        pass
```

```
class EmpleadoFullTime(Empleado):
    def __init__(self, nombre, apellido, salario):
        super().__init__(nombre, apellido)
        self.__salario = salario

    @property
    def salario(self):
        return self.__salario
```

```
class EmpleadoPorHora(Empleado):
    def __init__(self, nombre, apellido, horas_trabajadas, valor_hora):
        super().__init__(nombre, apellido)
        self.__horas_trabajadas = horas_trabajadas
        self.__valor_hora = valor_hora

    @property
    def salario(self):
        return self.__horas_trabajadas * self.__valor_hora
```

Encapsulamiento

NO HAY VARIABLES
PRIVADAS EN PYTHON..
CONVENCIÓN "_" o "__"

Empleado
-nombre
-apellido
+nombre_completo()
+salario()

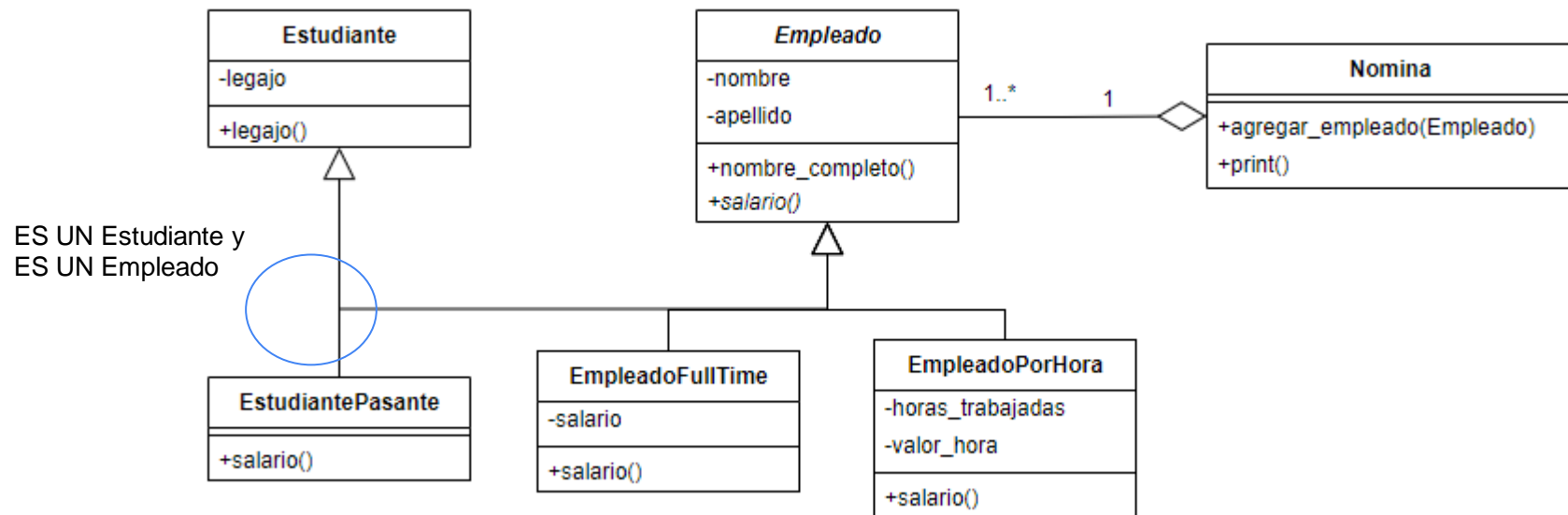
```
from abc import ABC, abstractmethod

class Empleado(ABC):
    def __init__(self, nombre, apellido):
        self.__nombre = nombre
        self.__apellido = apellido

    @property
    def nombre_completo(self):
        return f"{self.__nombre} {self.__apellido}"

    @property
    @abstractmethod
    def salario(self):
        pass
```

Herencia Múltiple



Herencia Múltiple

El orden importa

```
class Estudiante():  
    def __init__(self, legajo):  
        self.__legajo = legajo  
  
    @property  
    def legajo(self):  
        return self.__legajo
```

Llamo a los constructores de las clases base

```
class EstudiantePasante(Empleado, Estudiante):  
    def __init__(self, nombre, apellido, legajo):  
        Empleado.__init__(self, nombre, apellido)  
        Estudiante.__init__(self, legajo)  
  
    # Tengo que implementar la propiedad salario porque hereda de empleado  
    @property  
    def salario(self):  
        return 0
```

**No te olvides de completar la
asistencia y consultar dudas**

Recordá:

- Revisar la Cartelera de Novedades.
- Hacer tus consultas en el Foro.

TODO EN EL AULA VIRTUAL