Municipalidad de Tres de Febrero

TECNO 3F | Programá tu futuro

PYTHON INTERMEDIO

¡Les damos la bienvenida!

Municipalidad de Tres de Febrero

TECNO 3F | Programá tu futuro

PROGRAMACIÓN ORIENTADA A OBJETOS II

CLASE 5

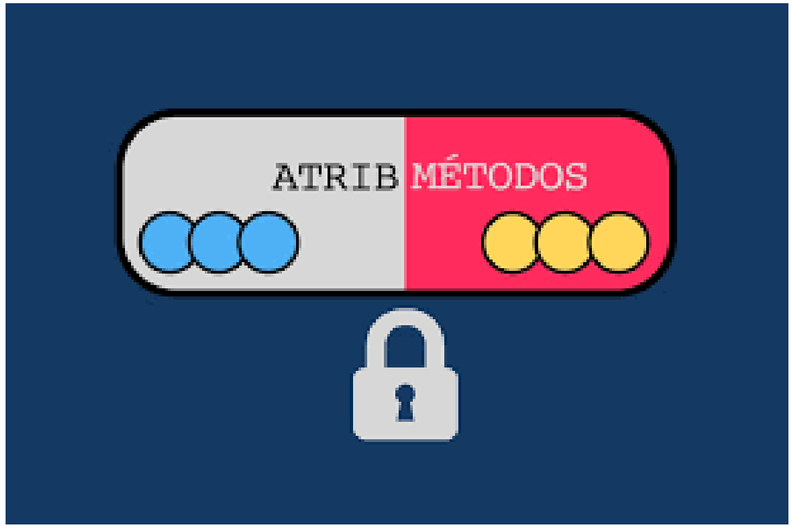
ENCAPSULAMIENTO

TECNO3F | Programá tu futuro

3F | Municipalidad de Tres de Febrero

¿QUÉ ES?

El encapsulamiento es un pilar fundamental de la Programación Orientada a Objetos (POO) que se refiere a la práctica de ocultar los detalles de implementación de una clase y exponer solo una interfaz pública para interactuar con ella.
En otras palabras, permite controlar el acceso a los atributos y métodos de una clase, protegiendo sus datos internos y promoviendo una mejor organización del código



Como se hace en python

Si bien Python no tiene soporte nativo para atributos privados, se utilizan convenciones para simular su comportamiento:

Convención de nomenclatura: Se utiliza una convención de nomenclatura para diferenciar entre atributos públicos y privados. Los atributos públicos no tienen prefijo, mientras que los privados tienen el guión bajo (_).

Veamos un ejemplo

Creemos la clase **CuentaBancaria** que encapsule los atributos , nombre_titular , dni_titular , fecha_nacimiento, saldo .

```python
class CuentaBancaria:
    def __init__(self, nombre_titular, dni_titular, fecha_nacimiento ,saldo = 0):
        self._nombre_titular = nombre_titular     # Atributo privado
        self._dni_titular = dni_titular           # Atributo privado
        self._fecha_nacimiento = fecha_nacimiento # Atributo privado
        self._saldo = saldo                       # Atributo privado
```

ahora vamos a crear unas acciones para esta clase

Veamos un ejemplo

Tambien debe contener los metodos mostrar_saldo , depositar , extraer y calcular _edad (este ultiumo debe ser privado)

```
def obtener_saldo(self):
    return self._saldo

def depositar(self, monto):
    if monto > 0:
        self._saldo += monto
        print(f"Se ha depositado {monto} a la cuenta de {self._nombre_titular}, su saldo actual es de :
        {self.obtener_saldo()}.")
    else:
        print("El monto a depositar debe ser positivo.")

def extraer(self, monto):
    if monto <= self._saldo:
        self._saldo -= monto
        print(f"Se ha extraído {monto} de la cuenta de {self._nombre_titular}, su saldo actual es de :
        {self.obtener_saldo()}.")
    else:
        print("No posee saldo suficiente para realizar esta operación.")

# Implementación del método privado
def _calcular_edad(self):
    fecha_actual = date.today()
    edad = fecha_actual - self._fecha_nacimiento
    return edad.days // 365
```

Veamos un ejemplo

Bien, ya que nuestro metodo **_calcular_edad** es privado , vamos acrear el metodo obtener_edad que sera publico

```
    def obtener_edad(self):
        return self._calcular_edad()
```

Ahora instanciemos la clase y hagan algunos movimientos en sus cuentas

```
cuenta = CuentaBancaria("Gabriel",33333333,date(1990,3,2),500)
print(cuenta.obtener_edad())
```

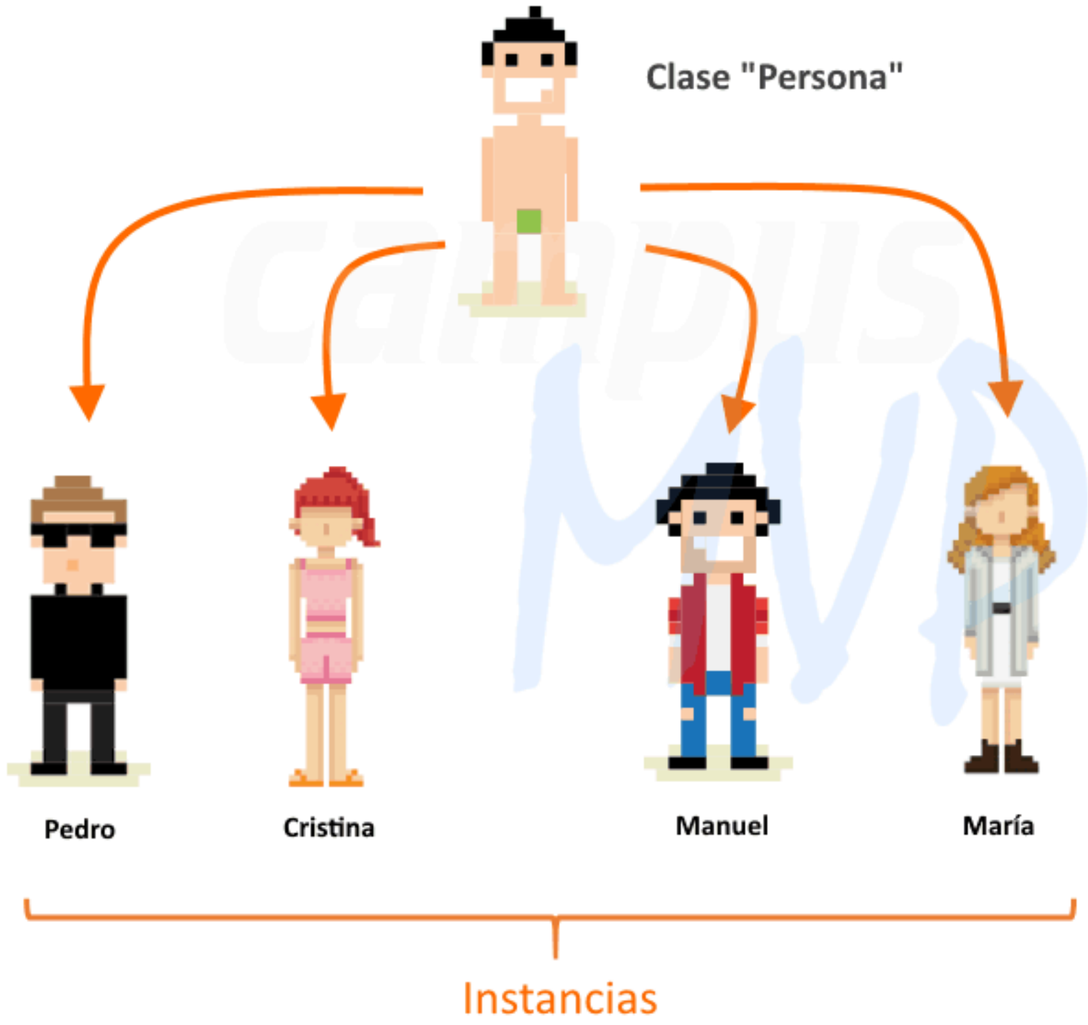
HERENCIA

TECNO3F | Programá tu futuro

3F | Municipalidad de Tres de Febrero

¿QUÉ ES?

La herencia es una técnica de POO que permite crear clases a partir de otras clases existentes.
Las clases hijas heredan las propiedades y métodos de la clase padre, lo que facilita la reutilización del código y la creación de jerarquías de clases.



¿Como heredamos?

Vamos a crear una clase hija llamada **CuentaCorriente** la cual heredara los atributos y metodos de la clase padre CuentaBancaria agregandole un nuevo atributo privado limite_extraccion

```
class CuentaCorriente(CuentaBancaria):
    def __init__(self, nombre_titular, dni_titular, fecha_nacimiento, saldo=0, limite_extraccion = 500):
        super().__init__(nombre_titular, dni_titular, fecha_nacimiento, saldo)
        self._limite_extraccion = limite_extraccion
```

Analicemos el fragmento...

¿Como heredamos?

Al crear la calse hija debemos indicarle de que clase hereda

```
class CuentaCorriente(CuentaBancaria):
    def __init__(self, nombre_titular,
```

```
f __init__(self, nombre_ti
  super().__init__(nombre_
```

luego accedemos al metodo contrusctor del padre utilizando la funcion super()

Si nosotros queres darle atributos propios lo haremos en el constructur propio de la clase

POLIMORFISMO

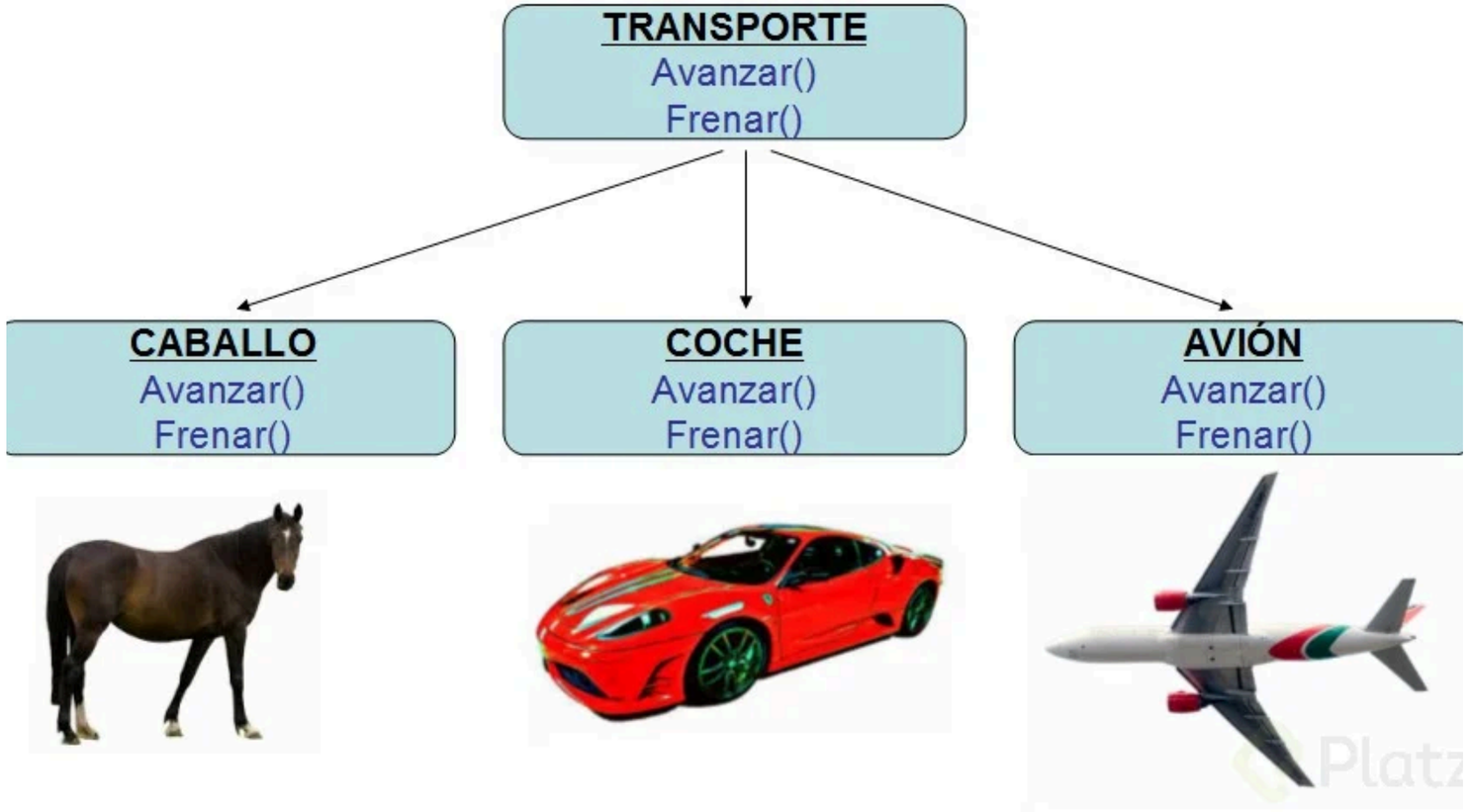
TECNO3F | Programá tu futuro

3F | Municipalidad de Tres de Febrero

¿QUÉ ES?

El polimorfismo es una característica fundamental de la Programación Orientada a Objetos (POO) que permite que objetos de diferentes clases respondan al mismo mensaje de manera diferente.

En otras palabras, un mismo método puede ser invocado en diferentes objetos y producir resultados distintos según la clase a la que pertenece el objeto.



Casos de uso del polimorfismo:

Implementar interfaces comunes:

- Se define un método con el mismo nombre en diferentes clases, permitiendo que se invoque desde cualquier instancia de esas clases.
- Cada clase implementa el método de forma específica, adaptándolo a su comportamiento particular.

Abstraer funcionalidades:

- Se crea una clase base con un método abstracto, definiendo la firma del método pero no su implementación.
- Las clases derivadas implementan el método abstracto de forma específica, adaptándolo a sus necesidades.

Implementando...

Ahora vamos a crear el método extraer el cual al recibir un monto debe verificar que tengamos el saldo suficiente y que nuestra cuenta este habilitada para retirar dicho monto.

```
def extraer(self, monto):
    if monto <= self._saldo and monto <= self._limite_extraccion:
        super().extraer(monto)
    else:
        if monto > self._limite_extraccion:
            print("Ustedes no puede extraer ese monto")
        else:
            print("Ustedes no posee saldo suficiente para realizar la operación")
```

CLASE ABSTRACTA

TECNO 3F | Programá tu futuro

3F | Municipalidad de Tres de Febrero

¿Como la creamos?

En python utilizaremos el modulo ABC para determinar que una clase es abstracta

```
from abc import ABC , abstractmethod
```

¿Como la creamos?

Bien ahora , creamos la clase abstracta FiguraGeometica y por parametro le enviamos ABC (Abstrac Basic Class) tendra un atributo privado : nombre , tendra 2 metodos abstractos para calcular el area y el perimetro y un metodo para obtener el nombre

```python
class FiguraGeometrica(ABC):
    def __init__(self, nombre):
        self._nombre = nombre #atributo privado

    @abstractmethod
    def calcular_area(self):
        pass

    @abstractmethod
    def calcular_perimetro(self):
        pass

    def obtener_nombre(self):
        return self._nombre
```

¿Que pasa si...?

Bien ahora , intentemos instanciar esta clase e intentemos acceder al metodo de obtener nombre

```
figura = FiguraGeometrica("casa")

figura.obtener_nombre()
```

¿Que pasa si...?

Bien ahora , intentemos instanciar esta clase e intentemos acceder al metodo de obtener nombre

```python
figura = FiguraGeometrica("casa")

figura.obtener_nombre()
```

```
Traceback (most recent call last):
  File "c:\Users\magog\Documents\Tecno3F\python\inter\1s 2024\clase 6\abstracta.py", line 18, in <module>
    figura = FiguraGeometrica("casa")
             ^^^^^^^^^^^^^^^^^^^^^^^^
TypeError: Can't instantiate abstract class FiguraGeometrica without an implementation for abstract methods 'calcular_area', 'calcular_perimetro'
```

No podemos instanciar una clase abstracta

PARA PRACTICAR

TECNO3F | Programá tu futuro

3F | Municipalidad de Tres de Febrero

Práctica.. Práctica.. Práctica

Se debe modificar la clase **CuentaBancaria** para que sea abstracta , ademas los metodos **extraer** y **depositar** deben volverse abstractos, tambien se debe crear una clase **CuentaAhorro** que herede de **CuentaBancaria** y se le agregue un atributo privado de **tasa de interes**, el cual tendra un valor establecido de **0.001** y un metodo que nos calcule el interes.

¡MUCHAS GRACIAS!

TECNO3F | Programá tu futuro

3F | Municipalidad de Tres de Febrero