

# Programación Orientada a Objetos (POO)

## Propiedades

### Ejercicio: Crear una cuenta bancaria con propiedades

Realizar lo siguiente:

- Definir una clase *CuentaBancaria* con su constructor, que reciba el titular de la cuenta y el saldo
- Definir las propiedades para obtener el valor de los atributos
- Definir un setter para establecer un nuevo saldo (No se usará setter para el titular de la cuenta, ya que no queremos que el titular cambie). Realizar manejo de excepciones
- Definir métodos de instancia para depositar y retirar. Además, definir otro método para ver la información de la cuenta

### Explicación de la Clase CuentaBancaria

- La clase *CuentaBancaria* modela una cuenta bancaria simple con funcionalidades para gestionar el saldo, realizar depósitos y retiradas, y mostrar la información de la cuenta.

### Atributos Privados

- `__titular`: Atributo privado que almacena el nombre del titular de la cuenta.
- `__saldo`: Atributo privado que almacena el saldo de la cuenta.

### Métodos

- `__init__(self, titular, saldo_inicial)`
  - **Propósito:** Constructor de la clase *CuentaBancaria*. Inicializa los atributos `__titular` y `__saldo` de la cuenta.
  - **Parámetros:**
    - \* `titular`: Nombre del titular de la cuenta.
    - \* `saldo_inicial`: Saldo inicial de la cuenta.

- **Comportamiento:** Establece el titular y el saldo inicial de la cuenta. `__saldo` se establece en 0 inicialmente, y el saldo inicial se asigna mediante el setter `saldo`.
- `@property def titular(self)`
  - **Propósito:** Permite acceder al atributo `__titular` como si fuera un atributo público.
  - **Getter:** Devuelve el valor del atributo privado `__titular`.
- `@property def saldo(self)`
  - **Propósito:** Permite acceder al atributo `__saldo` como si fuera un atributo público.
  - **Getter:** Devuelve el valor del atributo privado `__saldo`.
- `@saldo.setter def saldo(self, nuevo_saldo)`
  - **Propósito:** Permite modificar el valor del atributo `__saldo`.
  - **Setter:** Establece el valor de `__saldo` solo si el nuevo saldo es mayor o igual a 0. Si el saldo es negativo, lanza una excepción `ValueError`.
- `def depositar(self, cantidad)`
  - **Propósito:** Permite realizar un depósito en la cuenta.
  - **Parámetro:**
    - \* `cantidad`: Monto a depositar.
  - **Comportamiento:** Aumenta el saldo de la cuenta si la cantidad es positiva. Lanza una excepción `ValueError` si la cantidad es negativa.
- `def retirar(self, cantidad)`
  - **Propósito:** Permite retirar una cantidad de la cuenta.
  - **Parámetro:**
    - \* `cantidad`: Monto a retirar.
  - **Comportamiento:** Reduce el saldo de la cuenta si la cantidad es positiva y suficiente para cubrir el retiro. Lanza excepciones `ValueError` si la cantidad es negativa o si hay fondos insuficientes.
- `def mostrar_info(self)`
  - **Propósito:** Muestra la información de la cuenta.
  - **Comportamiento:** Imprime el titular y el saldo de la cuenta.

```

class CuentaBancaria:
    def __init__(self, titular, saldo_inicial):
        self.__titular = titular
        self.__saldo = 0
        self.saldo = saldo_inicial

    @property
    def titular(self):
        return self.__titular

    @property
    def saldo(self):
        return self.__saldo

    @saldo.setter
    def saldo(self, nuevo_saldo):
        try:
            if nuevo_saldo >= 0:
                self.__saldo = nuevo_saldo
            else:
                raise ValueError("El saldo no puede ser negativo")
        except ValueError as e:
            print(f"Error al establecer el saldo: {e}")

    def depositar(self, cantidad):
        try:
            if cantidad > 0:
                self.__saldo += cantidad
                print(f"Depositado: {cantidad}. Saldo actual: {self.__saldo}")
            else:
                raise ValueError("La cantidad a depositar debe ser positiva")
        except ValueError as e:
            print(f"Error al depositar: {e}")

    def retirar(self, cantidad):
        try:
            if cantidad > 0:
                if cantidad <= self.__saldo:
                    self.__saldo -= cantidad
                    print(f"Retirado: {cantidad}. Saldo actual: {self.__saldo}")
                else:

```

```

        raise ValueError("Fondos insuficientes para la transacción")
    else:
        raise ValueError("La cantidad a retirar debe ser positiva")
except ValueError as e:
    print(f"Error al retirar: {e}")

def mostrar_info(self):
    print(f"Titular: {self.titular}")
    print(f"Saldo: {self.saldo}")

```

```

cuenta = CuentaBancaria("José Ortega", 1000)
cuenta.mostrar_info()

cuenta.depositar(500)
cuenta.retirar(200)

cuenta.mostrar_info()

```

```

Titular: José Ortega
Saldo: 1000
Depositado: 500. Saldo actual: 1500
Retirado: 200. Saldo actual: 1300
Titular: José Ortega
Saldo: 1300

```