Paradigmas de la programación

Hernández Ceseña Iván Fernando

Práctica 2.

Paradigmas de la programación orientada a objetos

29 de Mayo del 2024

Instrucciones

Crear una aplicación en Python utilizando el paradigma orientado a objetos. En grupo definir la aplicación y sus requerimientos.

Debe de manejar e indicar en el código los siguientes conceptos:

- Clases

- Objetos

- Abstracción de datos

- Encapsulamiento

- Herencia

- Polimorfismo

En el grupo se acordó que se creará una aplicación que simule un sistema de un banco que podrá crear diferentes cuentas, como retirar dinero, transferir y mostrar información de las cuentas

Desarrollo

Para el manejo de cuentas se crearon 2 clases, una clase padre llamada Cuenta, y una cuenta hija llamada CuentaAhorro. Esta clase aplica la herencia ya que es derivada de su clase padre Cuenta. Esta misma cuenta con los atributos de la clase Cuenta, pero agrega un atributo más llamado interés.

Para el manejo del banco se creó la clase Banco la cual contiene a las dos clases antes mencionadas.

Clases

```python
class Cuenta:
    def __init__(self,numero_cuenta, titular, cantidad):
        self.numero_cuenta = numero_cuenta
        self.titular = titular
        self.cantidad = cantidad

    def depositar(self,cantidad):
        self.cantidad += cantidad
        print(f"Deposito exitoso, saldo actual: {self.cantidad}")

    def retirar(self,cantidad):
        if self.cantidad < cantidad:
            print("Saldo insuficiente")
        else:
            self.cantidad -= cantidad
            print(f"Retiro exitoso, saldo actual: {self.cantidad}")

    def transferir(self,destino,monto):
        if self.cantidad < monto:
            print("Saldo insuficiente")
        else:
            self.cantidad -= monto
            destino.cantidad += monto
            print(f"Transferencia exitosa, saldo actual: {self.cantidad} \n")

    def __str__(self):
        return f"Numero de cuenta: {self.numero_cuenta}\nTitular: {self.titular}\nCantidad: {self.cantidad}"
```

```python
class CuentaAhorros(Cuenta):
    def __init__(self, numero_cuenta, titular, cantidad, interes):
        super().__init__(numero_cuenta, titular, cantidad)
        self.interes = interes


    def calcular_interes(self):
        self.cantidad += self.cantidad * self.interes
        return self.cantidad

    def __str__(self):
        informacion_base = super().__str__()
        return f"{informacion_base}\nInteres: {self.interes} \nSaldo con intereses: {self.calcular_interes()}"
```

```python
class Banco:
    def __init__(self,nombre):
        self.nombre = nombre
        self.cuentas = []
        self.numero_cuentas = 0

    def crear_cuenta(self,titular,cantidad):
        self.numero_cuentas += 1
        nueva_cuenta = Cuenta(self.numero_cuentas,titular,cantidad)
        self.cuentas.append(nueva_cuenta)
        print(f"Cuenta creada con exito, numero de cuenta: {self.numero_cuentas}")
        return nueva_cuenta

    def crear_cuenta_ahorros(self,titular,cantidad,interes):
        self.numero_cuentas += 1
        nueva_cuenta = CuentaAhorros(self.numero_cuentas,titular,cantidad,interes)
        self.cuentas.append(nueva_cuenta)
        return nueva_cuenta

    def getCuenta(self,numero_cuenta):
        for cuenta in self.cuentas:
            if cuenta.numero_cuenta == numero_cuenta:
                return cuenta
        return None

    def __str__(self):
        result = f"Banco: {self.nombre}\n"
        for cuenta in self.cuentas:
            result += str(cuenta) + "\n"
            result += "----------------------------\n"
        return result
```

Objetos

Para la creación de los objetos se creó una instancia de la clase Banco, y se crearon cuentas de ahorro y cuentas normales.

```
def crear_cuenta():
    tipo_cuenta = input("Ingrese el tipo de cuenta que desea crear (1. Cuenta normal, 2. Cuenta de ahorros): ")
    if tipo_cuenta == "1":
        titular = input("Ingrese el nombre del titular: ")
        cantidad = float(input("Ingrese la cantidad inicial: "))
        cuenta = banco.crear_cuenta(titular, cantidad)
    elif tipo_cuenta == "2":
        interes = input("Ingrese el interes de la cuenta de ahorros 1 - [0.5] 2 - [0.1] 3 - [0.05]: ")
        if interes == "1":
            interes = 0.5
        elif interes == "2":
            interes = 0.1
        elif interes == "3":
            interes = 0.05
        titular = input("Ingrese el nombre del titular: ")
        cantidad = float(input("Ingrese la cantidad inicial: "))
        cuenta = banco.crear_cuenta_ahorros(titular, cantidad, interes)
    else:
        print("Opcion invalida")
```

Abstracción de datos

En el código se puede observar la abstracción de datos en los métodos de las clases, ya que estos métodos no muestran cómo se realizan las operaciones, sólo se encargan de realizarlas.

También como en las funciones no se especifica el nivel de abstracción, los datos se quedan ocultos y solo se muestran los resultados.

Encapsulamiento

En el código se puede observar que se utilizó el encapsulamiento en los atributos de las clases, ya que estos se encuentran privados y solo se pueden acceder a ellos mediante los métodos de la clase.

```
def __init__(self,numero_cuenta, titular, cantidad):
    self.numero_cuenta = numero_cuenta
    self.titular = titular
    self.cantidad = cantidad
```

Polimorfismo

El polimorfismo se puede observar en el método __str__ de las clases, ya que se sobreescribe el método de la clase padre en la clase hija para que se muestre la información de la cuenta de ahorros con el saldo con intereses.

```
def __str__(self):
    informacion_base = super().__str__()
    return f"{informacion_base}\nInteres: {self.interes} \nSaldo con intereses: {self.calcular_interes()}"
```