Universidad Autonoma de Baja California

Eliel Alfonso Ontiveros Ojeda

368746

Introducción

En la materia de Paradigmas de la Programación, se destaca la importancia de escribir código claro, modular y reutilizable. La modularización no solo facilita la comprensión y el mantenimiento del código, sino que también mejora su escalabilidad y depuración. En este reporte, presentamos la implementación de un sistema bancario en Python, dividido en tres módulos principales: CuentaBancaria, SistemaBancario, y un archivo principal main que actúa como interfaz de usuario. Este enfoque modular permite una mejor organización y manejo del proyecto, ejemplificando principios clave de la programación orientada a objetos.

Desarrollo

Clase CuentaBancaria

La clase CuentaBancaria se encarga de las operaciones básicas de una cuenta bancaria, como depositar, retirar, consultar saldo y transferir fondos a otra cuenta. Esta clase está definida en el archivo cuenta_bancaria.py.

Archivo: cuenta_bancaria.py

class CuentaBancaria:
def **init**(self, numero_cuenta, saldo, tipo_cuenta, pin):
self.numero_cuenta = numero_cuenta
self.saldo = saldo
self.tipo_cuenta = tipo_cuenta
self.pin = pin

```
def depositar(self, monto):
    if monto > 0:
        self.saldo += monto
        print(f"Deposito exitoso. Saldo actual: {self.saldo}")
    else:
        print("El monto debe ser mayor que cero.")

def retirar(self, monto):
    if monto > 0 and monto <= self.saldo:
        self.saldo -= monto
        print(f"Retiro exitoso. Saldo actual: {self.saldo}")
        return True
    else:
        print("Monto invalido o saldo insuficiente.")
        return False

def consultar_saldo(self):
    print(f"Saldo actual: {self.saldo}")
```

```
def transferir(self, cuenta_destino, monto):
    if cuenta_destino and self.retirar(monto):
        cuenta_destino.depositar(monto)
        print(f"Transferencia de {monto} al numero de cuenta
{cuenta_destino.numero_cuenta} completada.")
    else:
        print("Transferencia fallida. Verifique los datos y el saldo.")
```

Clase SistemaBancario

La clase SistemaBancario administra las cuentas bancarias de los clientes. Permite registrar nuevos clientes y gestionar el inicio de sesión. Esta clase está definida en el archivo sistema_bancario.py.

Archivo: sistema_bancario.py

from cuenta_bancaria import CuentaBancaria

class SistemaBancario:
def **init**(self):
self.clientes = {}
self.numero_cuenta_actual = 1000

```
def registrar_cliente(self, nombre):
    pin = input("Elija un PIN para su cuenta: ")
    self.numero_cuenta_actual += 1
    numero_cuenta = self.numero_cuenta_actual
    cuenta_bancaria = CuentaBancaria(numero_cuenta, 0, "Corriente", pin)
    self.clientes[numero_cuenta] = cuenta_bancaria
    print(f"Cuenta creada para {nombre}. Numero de cuenta: {numero_cuenta}")
    return cuenta_bancaria

def iniciar_sesion(self, numero_cuenta, pin):
    if numero_cuenta in self.clientes and self.clientes[numero_cuenta].pin == pin:
        return self.clientes[numero_cuenta]
    else:
        print("Numero de cuenta no valido o PIN incorrecto.")
        return None
```

Archivo principal main.py

El archivo main.py contiene la lógica de la interfaz de usuario, permitiendo a los usuarios interactuar con el sistema bancario para realizar operaciones como consultar saldo, transferir dinero, retirar dinero y depositar dinero.

Archivo: main.py

from sistema_bancario import SistemaBancario

sistema_bancario = SistemaBancario()

while True:
print("\nMenu:")
print("1. Iniciar sesión")
print("2. Crear nueva cuenta")

print("3. Salir")
opcion = input("Seleccione una opcion: ")

```
if opcion == "1":
    numero_cuenta = int(input("Ingrese su numero de cuenta: "))
    pin = input("Ingrese su PIN: ")
    cliente = sistema_bancario.iniciar_sesion(numero_cuenta, pin)
    if cliente:
        while True:
            print("\nOpciones disponibles:")
            print("1. Consultar Saldo")
            print("2. Transferir a otros usuarios")
            print("3. Retirar dinero")
            print("4. Depositar dinero")
            print("0. Volver al menu principal")
            accion = input("Seleccione una accion: ")

            if accion == "1":
                cliente.consultar_saldo()
            elif accion == "2":
                cuenta_destino = int(input("Ingrese el numero de cuenta destino: "))
                monto = float(input("Ingrese el monto a transferir: "))
                if sistema_bancario.iniciar_sesion(numero_cuenta, pin):
                    cuenta_destino_obj =
sistema_bancario.clientes.get(cuenta_destino)
                    if cuenta_destino_obj:
                        cliente.transferir(cuenta_destino_obj, monto)
                    else:
                        print("Numero de cuenta destino no valido.")
                else:
                    print("PIN incorrecto.")
            elif accion == "3":
                monto = float(input("Ingrese el monto a retirar: "))
                cliente.retirar(monto)
            elif accion == "4":
                monto = float(input("Ingrese el monto a depositar: "))
                cliente.depositar(monto)
            elif accion == "0":
                break
            else:
                print("Accion no valida.")

elif opcion == "2":
    nombre = input("Ingrese su nombre: ")
    sistema_bancario.registrar_cliente(nombre)

elif opcion == "3":
    print("¡Hasta luego!")
    break

else:
    print("Opcion no valida. Intente nuevamente.")
```

Conclusión

La modularización del sistema bancario en Python mejora significativamente su claridad y mantenibilidad. Dividir el código en módulos independientes permite un desarrollo más organizado y facilita la identificación de errores y la implementación de nuevas funcionalidades. Esta práctica refleja los principios de la programación orientada a objetos y la importancia de la separación de responsabilidades, aspectos fundamentales en el estudio de Paradigmas de la Programación. La implementación presentada es una base sólida que puede ser expandida con características adicionales y mejoras, demostrando la efectividad y necesidad de un enfoque modular en el desarrollo de software.