Paradigmas de la programación

Diego D. Quiros Vicencio

Manual de reportes.

Fecha de Entrega: 30 de mayo de 2024.

Reporte Practica 2 de la Clase Paradigmas de Programacion

Alumno Diego Demian Quiros Vicencio - 372688

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 acordo que se creara una aplicacion que simule un sistema de un banco donde se podran crear diferentes cuentas que puedan realizar distintas operaciones como ingresar y retirar dinero, transferir y mostrar informacion de la cuenta

Desarrollo de la practica

Abstraccion de Datos

Una de las cosas mas importantes a la hora de programar utilizando el paradigma orientado a objetos es como modelas las entidades y las interacciones que tendran entre ellas. En este programa se observa la abstraccion en las clases que creamos ya que pensamos en los elementos que tienen las cuentas de banco y como estan relacionadas entre clases. > Recordemos que la abstraccion es mas un proceso de analisis menta.

Clases

A continuacion se mostraran las clases que se crearon

```
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 realizado, su saldo actual es de: {self.cantidad}")

    def retirar(self,cantidad):
        if self.cantidad < cantidad:
            print("No cuenta con saldo suficiente")
        else:
            self.cantidad -= cantidad
            print(f"Retiro realizado, su saldo actual es de: {self.cantidad}")
```

```
    def transferir(self,destino,monto):
        if self.cantidad < monto:
            print("No cuenta con saldo suficiente")
        else:
            self.cantidad -= monto
            destino.cantidad += monto
            print(f"Transferencia realizada, su saldo actual es de: {self.cantidad} \n")

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

Clase Cuenta de Ahorros

Herencia

En esta clase se aplica la herencia ya que surge de su clase padre la cual es Cuenta, esta tiene sus mismos atributos pero agrega uno mas, el cual es "interes" #### Polimorfismo Se aplica el polimormismo en el metodo "**str**(self)" ya que sobre-escribe esta funcion que fue definida en la clase cuenta para que se adapte a los atributos de esta funcion

```
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 interes: {self.calc
```

Clase Banco

Esta es la clase principal que contiene a las otras dos

```
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, su numero de cuenta es: {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
```

Encapsulamiento

Este es un ejemplo donde se aplica el encapsulamiento, no se especifica que tipo de encapsulamiento se utiliza por lo que python hace los datos de tipo private como predeterminado lo que significa que los atributos solo son manipulables por la misma clase

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

Objetos

En el programa instanciamos objetos de una clase al momento que creamos las cuentas de banc

Ejemplo ~

```
tipo_cuenta = input("Ingrese el tipo de cuenta que desea crear (1. Cuenta normal, 2. Cuenta
            if tipo_cuenta == "1":
                titular = input("Ingrese el nombre del titular: ")
                cantidad = float(input("Ingrese la cantidad inicial: "))
                cuenta = banco.crear_cuenta(titular, cantidad)
```