Universidad Autónoma de Baja California

Facultad de ingeniería, Arquitectura y diseño

Ingeniería en Software y Tecnologías Emergentes

Paradigmas de la Programación

Docente: Gallegos Mariscal José Carlos

Alumno: Diarte Salas Gilberto

Matricula: 360954

Grupo: 941

Práctica 2 : Banco POO

Fecha de Entrega: 31/mayo/2024

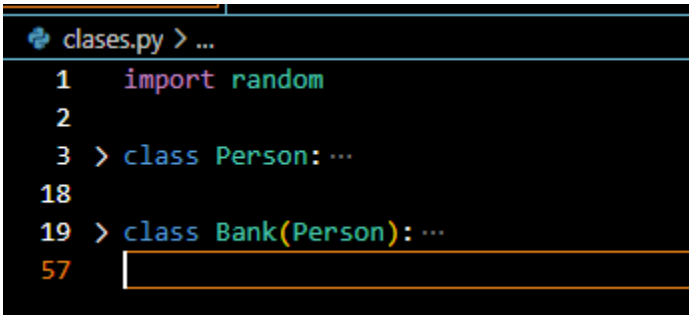
Introducción

El paradigma orientado a objetos es una metodología de programación que se basa en la conceptualización y modelado de entidades del mundo real como objetos, los cuales poseen atributos y comportamientos definidos por su clase. Este enfoque permite organizar y estructurar el código de manera más eficiente y modular, facilitando la reutilización y mantenimiento de este.

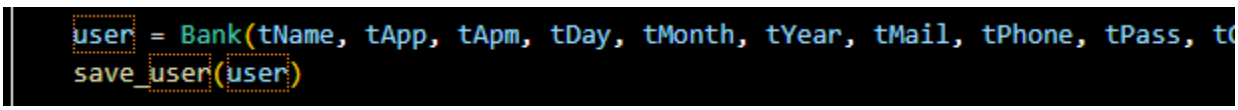
En este trabajo, se desarrollará una aplicación en Python utilizando el paradigma orientado a objetos. El objetivo es aplicar los principales conceptos de este paradigma, tales como clases, objetos, abstracción de datos, encapsulamiento, herencia y polimorfismo, para crear una aplicación funcional y estructurada.

Desarrollo

Clases: En el código, se definen dos clases principales: Person y Bank. La clase Person representa a una persona con atributos como nombre, apellido, fecha de nacimiento, correo, etc. La clase Bank hereda de Person y agrega atributos específicos relacionados con una cuenta bancaria, como número de cuenta, CLABE y saldo.



Objetos: Se crean objetos a partir de las clases definidas. Por ejemplo, en la función main(), se pueden crear múltiples instancias de la clase Bank para representar diferentes usuarios del sistema bancario.



Abstracción de datos: Las clases Person y Bank encapsulan datos y comportamientos relacionados en una única entidad. Por ejemplo, la clase Bank encapsula datos como el saldo y métodos para depositar, retirar y transferir dinero.

```
class Bank(Person):
    def __init__(self, name, app, apm, day, month, year, mail, num
        super().__init__(name, app, apm, day, month, year, mail, n
        self.no_cuenta = no_cuenta if no_cuenta else self.gen_cuen
        self.clabe = clabe if clabe else self.gen_clabe()
        self.saldo = saldo

    def gen_cuenta(self):
        return str(random.randint(1000000000, 9999999999))

    def gen_clabe(self):
        return str(random.randint(100000000000000000, 999999999999
```

Encapsulamiento: En la clase Bank, el atributo __password está encapsulado utilizando el mecanismo de atributos privados. Esto significa que solo se puede acceder a él desde dentro de la clase mediante métodos getter y setter, lo que garantiza la seguridad de la contraseña del usuario.

```
class Person:
    def __init__(self, name, app, apm, day, month, year, mail, nu
        self.name = name
        self.app = app
        self.apm = apm
        self.day = day
        self.month = month
        self.year = year
        self.mail = mail
        self.num_phone = num_phone
        self.__password = password
        self.curp = curp

    def get_password(self):
        return self.__password
```

Herencia: La clase Bank hereda de la clase Person, lo que significa que adquiere todos los atributos y métodos de la clase Person. Esto permite una reutilización eficiente del código y una estructura jerárquica en la definición de clases.

```
class Bank(Person):
    def   init  (self, name, app, apm, day, month,
```

Polimorfismo: El polimorfismo se manifiesta en el código a través de métodos como depositar(), retirar() y transferir(), que pueden comportarse de manera diferente según el objeto que los invoque. Por ejemplo, el método transferir() realiza una transferencia de fondos entre dos cuentas bancarias.

```
class Bank(Person):
    def __init__(self, name, app, apm, da

    def gen_cuenta(self):···

    def gen_clabe(self):···

    def depositar(self, amount):···

    def retirar(self, amount):···

    def transferir(self, other, amount):

    def view_saldo(self):···
```

Conclusión

mediante la aplicación de los conceptos de POO, hemos creado una aplicación bancaria que es fácil de entender, mantener y extender, lo que demuestra la eficacia de este paradigma de programación para el desarrollo de software modular y escalable.