Instituto Politécnico Nacional

Escuela Superior de Cómputo

Profesor: Ituriel Flores

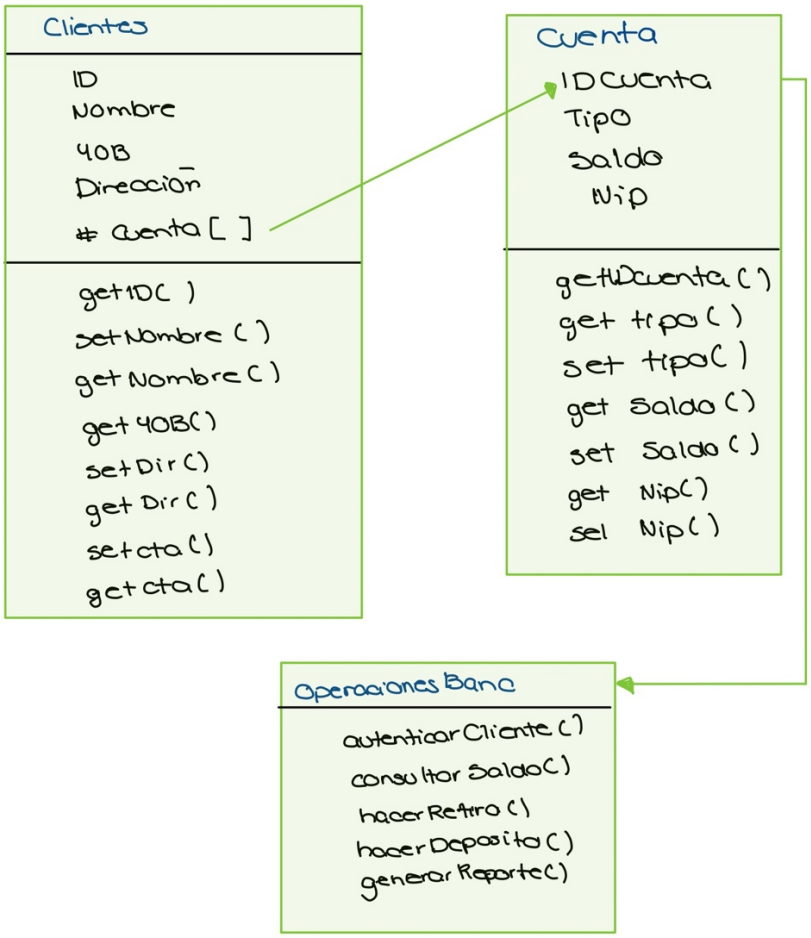
Alumno: De Luna Ocampo Yanina

Grupo: 4CDV1

Carrera: Licenciatura en Ciencia de Datos

Fecha: 12/03/2022

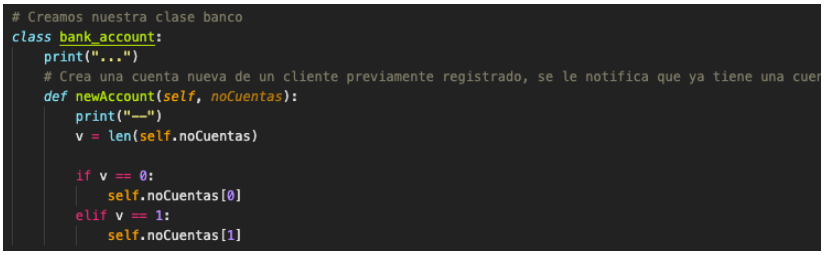
UML



Este programa, es una simulación de todo lo que implica con una cuenta bancaria, como, por ejemplo: crear, ver, modificar, retirar, depositar, creación de cuentas, etc. Se realizó por medio de programación orientada a objetos. A continuación, se describe la cantidad de funciones utilizadas, con sus especificaciones necesarias.

1. Class bank_account()

   Esta clase la creé para nuevas cuentas, aquí nos dirá si su cuenta es considerada como premium o como estándar dependiendo de la cantidad de dinero que esté en su cuenta, tomando a consideración que si se hacen depósitos y retiros, esta va a ir cambiando conforme la cantidad de dinero disponible dentro de esta. Y nos dirá si ya se han creado las cuentas máximas del usuario en marcha.



```
# Creamos nuestra clase banco
class bank_account:
    print("...")
    # Crea una cuenta nueva de un cliente previamente registrado, se le notifica que ya tiene una cuer
    def newAccount(self, noCuentas):
        print("--")
        v = len(self.noCuentas)

        if v == 0:
            self.noCuentas[0]
        elif v == 1:
            self.noCuentas[1]
```

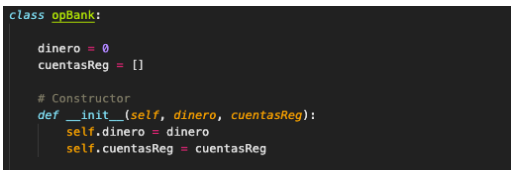
2. Class Cliente()

   Creamos todo acerca de los clientes, como: nombre, apellido, año de nacimiento, dirección, su ID irrepetible.

```
class Cliente:

    nombre = []
    anioNac = []
    dinero = 0
    cuenta = []

    def __init__(self, nombre, anioNac):
        self.nombre = nombre
        self.anioNac = anioNac

    # Definimos las funciones que utilizaremos, todas las operaciones de los menus que aparezcan
    def createID(self):
        miID = []
        miID = random.random()
        print(miID)

    def nombre(self, nombre):
        nombre = input("Digite su nombre: ")
        apellido = input("Digite su apellido: ")
        print(nombre + apellido)

    def modNombre(self, nombre):
        nombre = input("Digita correctamente tu nombre: ")
        print("Tu nuevo nombre es: ", nombre)

    def anioNac(self, anioNac):
        anioNac = input("Digita tu anio de nacimiento: ")
        print(anioNac)

    def direccion(self):
        direccion = input("Digita tu direccion: ")
        print(direccion)
```

3. Class opBank()

   Dentro de esta, haremos todas las operaciones pedidas dentro del enunciado, que son depósitos, retiros, consultar saldoy generar el reporte de cuenta.



```
class opBank:

    dinero = 0
    cuentasReg = []

    # Constructor
    def __init__(self, dinero, cuentasReg):
        self.dinero = dinero
        self.cuentasReg = cuentasReg
```

```python
# OperacionBancaria: retiro.
def wd(self, dinero):
    print("---")
    cantidad = float(input("Digita la cantidad a retirar: "))
    if cantidad > self.dinero:
        print("Saldo insuficiente. :(")
    else:
        self.dinero -= cantidad

    if dinero >= 100000:
        print("Tu cuenta es considerada premium.")
    else:
        print("Tu cuenta es considerada estandar.")

# OperacionBancaria: deposito.
def deposit(self, dinero):
    cantidad = float(input("Digita la cantidad a depositar: "))
    self.dinero += cantidad
    print(dinero)
    print("La cantidad se ha aniadido a su cuenta bancaria.")

    if dinero >= 100000:
        print("Tu cuenta es considerada premium.")
    else:
        print("Tu cuenta es considerada estandar.")

# OperacionBancaria: genera el reporte del cuenta del cliente.
def reportAccount(self, cuentasReg):
```

4. newClient()
   Esta se utiliza para los clientes que recién ingresan a nuestro banco, para así, poder registrarlos. Se les pide nombre (string), apellido (string), año de nacimiento (int), dirección (string), y su CURP (string), para poder llevar un registro completo de ellos. Una vez registradas estas operaciones, se agregan a los arrays creados para almacenar la información de los diferentes clientes que quieran ir ingresando. Finalmente imprime todos los datos dados e ingresados por el usuario para poder verificar su información.
5. modClient()
   El cliente podrá modificar los datos en donde se haya equivocado. Solamente será en su nombre y dirección.
6. newAccount()
   Esta, crea cuentas nuevas de usuarios que ya están registrado con nosotros, al momento de él ingresar el nombre (string), apellido (string), año de nacimiento (int), dirección (string), encontramos las similitudes entre estos y procedemos a informarle que ya tiene una cuenta registrada previamente, por lo que se le asignará la nueva. Imprime todos los datos dados e ingresados por el usuario para poder verificar su información.
7. wd()
   Procedemos a pedir información del usuario para validar que la operación deseada, en este caso de retiro, sea a su cuenta y no a la de alguien más, pedimos tarjeta (int) y NIP (int), después de validar estos, se pide el monto a depositar, imprimiendo su saldo final. Se pide la cantidad que el cliente desea retirar de su cuenta. Dependiendo de la cantidad que quede asignada en su cuenta, se le dirá si esta es premium o estándar.
8. deposit()
   Procedemos a pedir información del usuario para validar que la operación deseada, en este caso de depósito, sea a su cuenta y no a la de alguien más, pedimos tarjeta (int) y tarjeta (int), después de validar estos, se pide el monto a depositar, imprimiendo su saldo final. Si su saldo final supera la cantidad de $100,000, le imprimirá un letrero automáticamente de

que su cuenta esta siendo considerada como premium, de no ser así, se le notificará al usuario que su cuenta es estándar.

9. reportAccount()
   Procedemos a pedir información del usuario para validar que la operación deseada, en este caso de depósito, sea a su cuenta y no a la de alguien más, pedimos tarjeta (int) y NIP (int), después de validar estos, se pide el monto a depositar, imprimiendo su saldo final.
10. consultarSaldo()
    Te muestra el saldo final de la cuenta después de las operaciones necesarias hechas por el usuario.

Muestra del programa funcionando:

Menú bancario

```
----------------------Bienvenido a tu banco.----------------------

Las posibles operaciones dentro de tu cuenta bancaria son:
        1. Crear nuevo cliente.
        2. Modificar datos de cliente.
        3. Crear una nueva cuenta.
        4. Operaciones bancarias.
        5. Salir.

Digita la opción que desees para este módulo:
```

1. Función nombre, direccion, anioNac, createID

```python
def createID(self):
    miID = []
    miID = random.random()
    print(miID)

def nombre(self, nombre):
    nombre = input("Digite su nombre: ")
    apellido = input("Digite su apellido: ")
    print(nombre + apellido)

def modNombre(self, nombre):
    nombre = input("Digita correctamente tu nombre: ")
    print("Tu nuevo nombre es: ", nombre)

def anioNac(self, anioNac):
    anioNac = input("Digita tu anio de nacimiento: ")
    print(anioNac)

def direccion(self):
    direccion = input("Digita tu direccion: ")
    print(direccion)
```

```python
if mist == 1:
    nom = "Yanina"
    yob = 2001
    clnt = Cliente(nom, yob)
    print(clnt.nombre)
    clnt.createID()
```

```
Digita la opcion que desees para este modulo: 1
Yanina
0.438074479408
```

2. Función modClientes
   Intenté acceder a los datos del usuario mediante un diccionario para poder modificarlos.
3. Función newAccount()

```python
def newAccount(self, noCuentas):
    print("--")
    v = len(self.noCuentas)

    if v == 0:
        self.noCuentas[0]
    elif v == 1:
        self.noCuentas[1]
```

```
Crearas una nueva cuenta bancaria.
```

4. Función OperacionesBancarias

```
Digita la opcion que desees para este modulo: 4

Las operaciones posibles dentro de este modulo son:
        1. Consultar saldo.
        2. Hacer retiros.
        3. Hacer depositos.
        4. Generar reporte de cuenta.
        5. Atras.

Digita la operacion deseada para este modulo:
```

```python
def deposit(self, dinero):
    cantidad = float(input("Digita la cantidad a depositar: "))
    self.dinero += cantidad
    print(dinero)
    print("La cantidad se ha aniadido a su cuenta bancaria.")

    if dinero >= 100000:
        print("Tu cuenta es considerada premium.")
    else:
        print("Tu cuenta es considerada estandar.")
```

```python
def wd(self, dinero):
    print("---")
    cantidad = float(input("Digita la cantidad a retirar: "))
    if cantidad > self.dinero:
        print("Saldo insuficiente. :(")
    else:
        self.dinero -= cantidad

    if dinero >= 100000:
        print("Tu cuenta es considerada premium.")
    else:
        print("Tu cuenta es considerada estandar.")
```

```python
# OperacionBancaria: genera el reporte del cuenta del cliente.
def reportAccount(self, cuentasReg):
    print("---")
    print(cuentasReg)

    for a in range(len(cuentasReg)):
        for b in range(len(cuentasReg)):
            print(cuentasReg[a][b])
```

```python
# Mostramos el saldo dentro de la cuenta
def mostrarSaldo(self, dinero):
    print("\nTu saldo disponible neto: ", dinero)
```

Código dentro del menú:

```
if mist == 4:
    try:
        bankSln = 0
        while bankSln != 5:
            print("\nLas operaciones posibles dentro de este modulo son: ")
            print("\t1. Consultar saldo.")
            print("\t2. Hacer retiros.")
            print("\t3. Hacer depositos.")
            print("\t4. Generar reporte de cuenta.")
            print("\t5. Atras.")

            bankSln = int(input("\nDigita la operacion deseada para este modulo: "))

            if bankSln == 1:
                print("---")
                operaciones.mostrarSaldo(dinero=cantidad)

            if bankSln == 2:
                cantidad = 500
                operaciones = opBank(cantidad)
                operaciones.wd(dinero=cantidad)

            if bankSln == 3:
                cantidad = 500000
                operaciones = opBank(cantidad)
                operaciones.deposit(dinero=cantidad)

            if bankSln == 4:
                operaciones.reportAccount()

    except ValueError:
        print("\nOperacion incorrecta, intentelo nuevamente.")
```