TITULO DE LABORATORIO
PARCIAL (NEQUI)

HANNA KATHERINE ABRIL GÓNGORA
KAROL ASLEY ORJUELA MAPE

UNIVERSIDAD MANUELA BELTRÁN

PROGRAMACIÓN ORIENTADA A OBJECTOS

DOCENTE

DIANA MARCELA TOQUICA RODRÍGUEZ

FECHA

1. **EXPLICACIÓN DEL CÓDIGO EN PYTHON**

- En esta sección, se importa el módulo *datetime* para trabajar con fechas respecto a las metas de ahorro y se define la clase *MetaRecaudo*. En el método *__init__*, se inicializan los atributos de una meta de ahorro, como el monto objetivo, la fecha de establecimiento de dicho monto, el estado (inicializado como "No cumplida, esta cambia dependiendo del estado del ahorro") y el ahorro actual (inicializado en 0).

```
1   import datetime
2
3   class MetaRecaudo:
4       def __init__(self, monto_objetivo):
5           self.monto_objetivo = monto_objetivo
6           self.fecha_establecimiento = datetime.date.today()
7           self.estado = "No cumplida"
8           self.ahorro_actual = 0
T 9
```

- En esta parte, se define la clase *BancoDigital*, que administra las operaciones bancarias y las metas de ahorro. Se inicializan los atributos del banco, como el saldo total, la lista de metas de ahorro, el diccionario de usuarios y contraseñas, y el usuario actual (inicialmente *None*). Ya que se implemento un sistema de autenticación para que no sea tan fácil ejecutar acciones del menú principal si no fue inscrito anteriormente.

```
13  class BancoDigital:
14      def __init__(self):
15          self.saldo_total = 0
16          self.metas_recaudo = []
17          self.usuarios = {}
18          self.usuario_actual = None
T 19
```

- Este método *autenticar_usuario* verifica las credenciales del usuario para iniciar sesión. Si el usuario ya está autenticado, devuelve True; de lo contrario, solicita al usuario que ingrese sus credenciales y las compara con las almacenadas en el diccionario de usuarios y contraseñas. Si la autenticación es exitosa, establece el usuario actual y devuelve True; de lo contrario, devuelve False.

```
13  class BancoDigital:
14      def __init__(self):
15          self.saldo_total = 0
16          self.metas_recaudo = []
17          self.usuarios = {}
18          self.usuario_actual = None
19
20      def autenticar_usuario(self):
21          if self.usuario_actual is None:
22              usuario = input("Ingrese su usuario: ")
23              contraseña = input("Ingrese su contraseña: ")
24              if usuario in self.usuarios and self.usuarios[usuario] == contraseña:
25                  self.usuario_actual = usuario
26                  print("Autenticación exitosa.")
27                  return True
28              else:
29                  print("Usuario o contraseña incorrectos. Intente nuevamente.")
30                  return False
31          else:
32              return True
```

- El método *inscribir_usuario* permite registrar nuevos usuarios. Muestra un menú de opciones donde el usuario puede ingresar un nuevo usuario y contraseña, ingresar al menú principal o salir del sistema. Dependiendo de la opción seleccionada, se ejecuta la acción correspondiente.

```
34      def inscribir_usuario(self):
35          while True:
36              print("----- Menú de Inscripción -----")
37              print("")
38              print("1. Ingresar nuevo usuario")
39              print("2. Ingresar al menú principal")
40              print("3. Salir del sistema")
41              print("")
42              print("-------------------------------")
43              print("")
44              opcion = input("Seleccione una opción: ")
45              if opcion == "1":
46                  usuario = input("Ingrese el nuevo usuario: ")
47                  contraseña = input("Ingrese la contraseña del nuevo usuario: ")
48                  self.usuarios[usuario] = contraseña
49                  print("Usuario inscrito correctamente.")
50              elif opcion == "2":
51                  print("Ingresando al menú principal.")
52                  if self.autenticar_usuario():
53                      self.menu_principal()
54              elif opcion == "3":
55                  print("Saliendo del sistema.")
56                  exit()
57              else:
58                  print("Opción no válida. Por favor, seleccione una opción válida.")
59
```

- El método *menu_principal* muestra un menú con opciones para que el usuario interactúe con el banco. Dependiendo de la opción seleccionada, se llaman a los métodos correspondientes para realizar operaciones bancarias o gestionar las metas de ahorro. Este menú se realizó con el método Match – Case en Python, que es su similar switch en Java, ya que en este lenguaje no había como tal un método switch

```
60      def menu_principal(self):
61          while True:
62              print("--------- Menú Principal ----------")
63              print("")
64              print("1. Ver saldo total")
65              print("2. Ingresar dinero")
66              print("3. Enviar dinero")
67              print("4. Retirar dinero en cajero")
68              print("5. Retirar dinero en punto físico")
69              print("6. Establecer meta de ahorro")
70              print("7. Ver metas de ahorro")
71              print("8. Salir")
72              print("")
73              print("-----------------------------------")
74              print("")
75              opcion = input("Seleccione una opción: ")
76
77              match opcion:
78                  case "1":
79                      self.ver_saldo_total()
80                  case "2":
81                      monto = float(input("Ingrese el monto a ingresar: "))
82                      self.ingresar_dinero(monto)
83                  case "3":
84                      monto = float(input("Ingrese el monto a enviar: "))
85                      self.enviar_dinero(monto)
86                  case "4":
87                      monto = float(input("Ingrese el monto a retirar en cajero: "))
88                      self.retirar_dinero(monto, "cajero")
89                  case "5":
90                      monto = float(input("Ingrese el monto a retirar en punto físico
91                      self.retirar_dinero(monto, "punto físico")
92                  case "6":
93                      monto_objetivo = float(input("Ingrese el monto objetivo de ahor
94                      self.establecer_meta_de_ahorro(monto_objetivo)
95                  case "7":
96                      self.ver_metas_de_ahorro()
97                  case "8":
98                      print("Saliendo del programa.")
99                      return
100                 case _:
101                     print("Opción no válida. Por favor, seleccione una opción válid
102
```

- Estos métodos permiten realizar operaciones bancarias como ver el saldo total, ingresar dinero, enviar dinero, retirar dinero, establecer metas de ahorro, ver las metas de ahorro, etc. Para acceder a estas funciones en el menú es necesario hacer una autenticación previa con el usuario y la contraseña.

```
103         def ver_saldo_total(self):
104             print(f"Saldo total: COP {self.saldo_total}")
105 
106         def ingresar_dinero(self, monto):
107             self.saldo_total += monto
108             print(f"Se ingresó correctamente ${monto}. Saldo actual: {self.saldo_total}
109 
110         def enviar_dinero(self, monto):
111             if self.saldo_total ⩾ monto:
112                 self.saldo_total -= monto
113                 print(f"Se envió correctamente ${monto}. Saldo actual: {self.saldo_tota
114             else:
115                 print("Saldo insuficiente para realizar la transacción.")
116 
117         def retirar_dinero(self, monto, tipo):
118             if tipo == "cajero":
119                 if self.saldo_total ⩾ monto:
120                     self.saldo_total -= monto
121                     print(f"Se retiró correctamente ${monto} en cajero. Saldo actual: 
122                 else:
123                     print("Saldo insuficiente para realizar la transacción.")
124             elif tipo == "punto físico":
125                 print("Retiro en punto físico aún no implementado.")
126             else:
127                 print("Tipo de retiro no válido.")
128 
129         def establecer_meta_de_ahorro(self, monto_objetivo):
130             nueva_meta = MetaRecaudo(monto_objetivo)
131             self.metas_recaudo.append(nueva_meta)
132             print(f"Meta de ahorro establecida correctamente. Monto objetivo: ${monto_o
133 
134         def ver_metas_de_ahorro(self):
135             if self.metas_recaudo:
136                 print("Metas de ahorro:")
137                 for meta in self.metas_recaudo:
138                     print(meta)
139             else:
140                 print("No hay metas de ahorro establecidas.")
141 
142         def ejecutar(self):
143             self.inscribir_usuario()
144 
```

- Finalmente, se crea una instancia de *BancoDigital* llamada banco y se ejecuta el programa llamando al método *ejecutar()*, que inicia el flujo de interacción con el usuario.

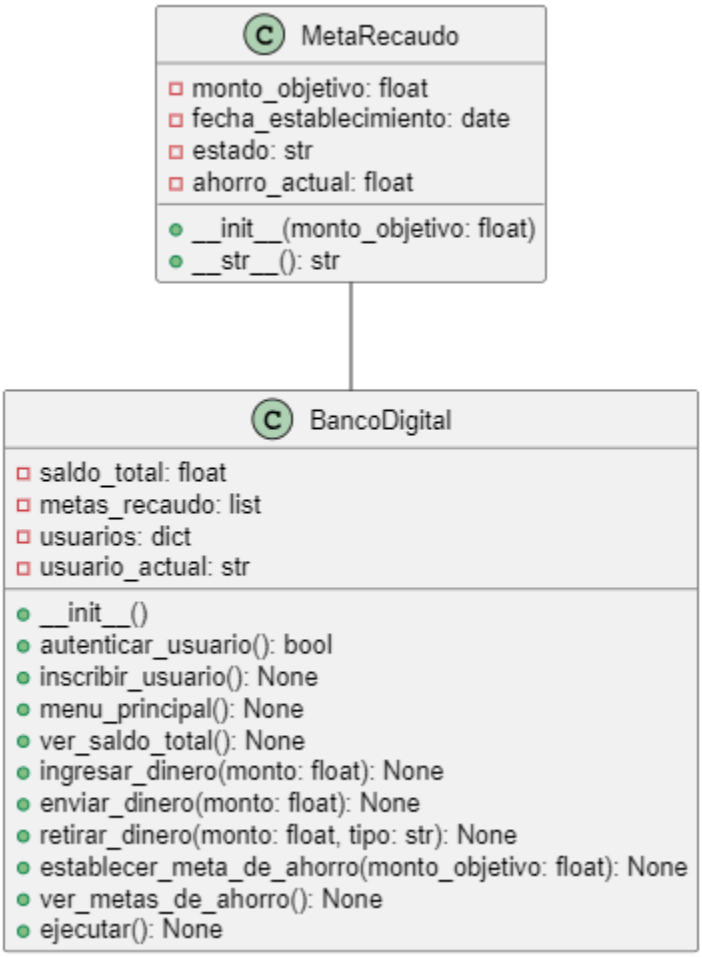
```
145  banco = BancoDigital()
146  banco.ejecutar()
```

2. **EXPLICACION DEL CODIGO UML**

```
@startuml Nequi

class MetaRecaudo {
    - monto_objetivo: float
    - fecha_establecimiento: date
    - estado: str
    - ahorro_actual: float
    + __init__(monto_objetivo: float)
    + __str__(): str
}

class BancoDigital {
    - saldo_total: float
    - metas_recaudo: list
    - usuarios: dict
    - usuario_actual: str
    + __init__()
    + autenticar_usuario(): bool
    + inscribir_usuario(): None
    + menu_principal(): None
    + ver_saldo_total(): None
    + ingresar_dinero(monto: float): None
    + enviar_dinero(monto: float): None
    + retirar_dinero(monto: float, tipo: str): None
    + establecer_meta_de_ahorro(monto_objetivo: float): None
    + ver_metas_de_ahorro(): None
    + ejecutar(): None
}

MetaRecaudo -- BancoDigital

@enduml
```

- **Línea 3-4:** Se define la clase MetaRecaudo con atributos como el monto objetivo (monto_objetivo: float) y la fecha de establecimiento (fecha_establecimiento: date).
- **Línea 5-6:** También tiene atributos para el estado (estado: str) y el ahorro actual (ahorro_actual: float).
- **Línea 7-8:** Se define un método constructor __init__ que inicializa la clase con un monto objetivo específico.
- **Línea 9:** Hay otro método __str__() que probablemente se utiliza para devolver una representación en cadena de la instancia de la clase.
- **Línea 12:** Se introduce otra clase llamada BancoDigital, con un atributo de saldo total (saldo_total: float).
- **Línea 13-14:** También gestiona las metas de recaudo en una lista (metas_recaudo: list) y los usuarios en un diccionario (usuarios: dict).
- **Línea 15-16:** Tiene un atributo para el usuario actual (usuario_actual: str) y varios métodos.
- **Líneas 18,19:** Los métodos incluyen autenticar usuario, inscribir usuario, y mostrar un menú principal.
- **Líneas 21-26:** Otros métodos permiten ver el saldo total, ingresar dinero, enviar dinero y retirar dinero.
- **Líneas 27-28:** También se pueden establecer metas de ahorro y verlas.
- **Linea 30:** Indica una relación entre las dos clases; parece ser parte del diagrama UML pero no está claro sin más contexto.

3. **DIAGRAMA DEL CODIGO EN UML**

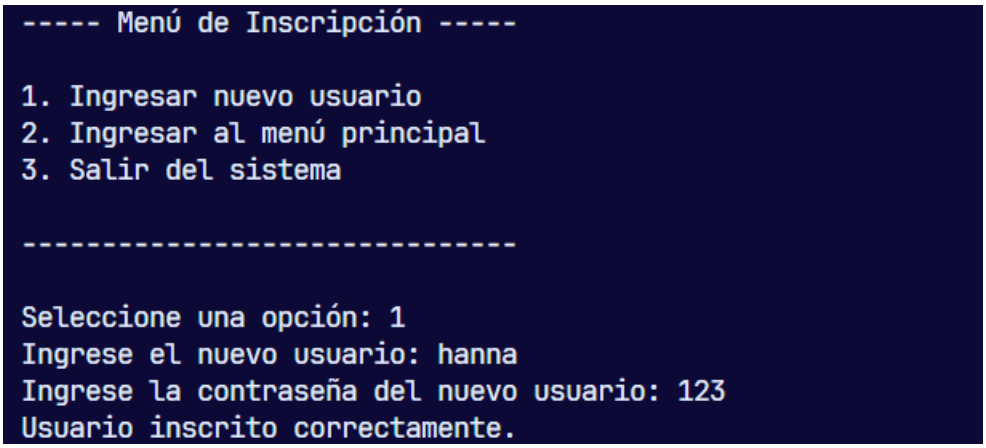


4. **EXPLICACION Y USO**

El código implementa un sistema básico de banco digital en este caso tipo Nequi. La clase MetaRecaudo representa una meta de ahorro con un monto objetivo, fecha de establecimiento, estado y ahorro actual. Por otro lado, la clase BancoDigital maneja operaciones bancarias como autenticación de usuarios, ingreso y envío de dinero, retiros en cajeros o puntos físicos, establecimiento de metas de ahorro y visualización de metas establecidas. El programa permite al usuario interactuar a través de un menú donde puede realizar diferentes operaciones bancarias y administrativas. Por ejemplo, puede inscribirse como nuevo usuario, autenticarse, establecer metas de ahorro, realizar transacciones financieras y verificar su saldo total. Este código demuestra conceptos clave de la Programación Orientada a Objetos (POO) como abstracción, encapsulamiento y uso de clases y métodos para modelar y gestionar entidades y acciones en un sistema de banco digital.

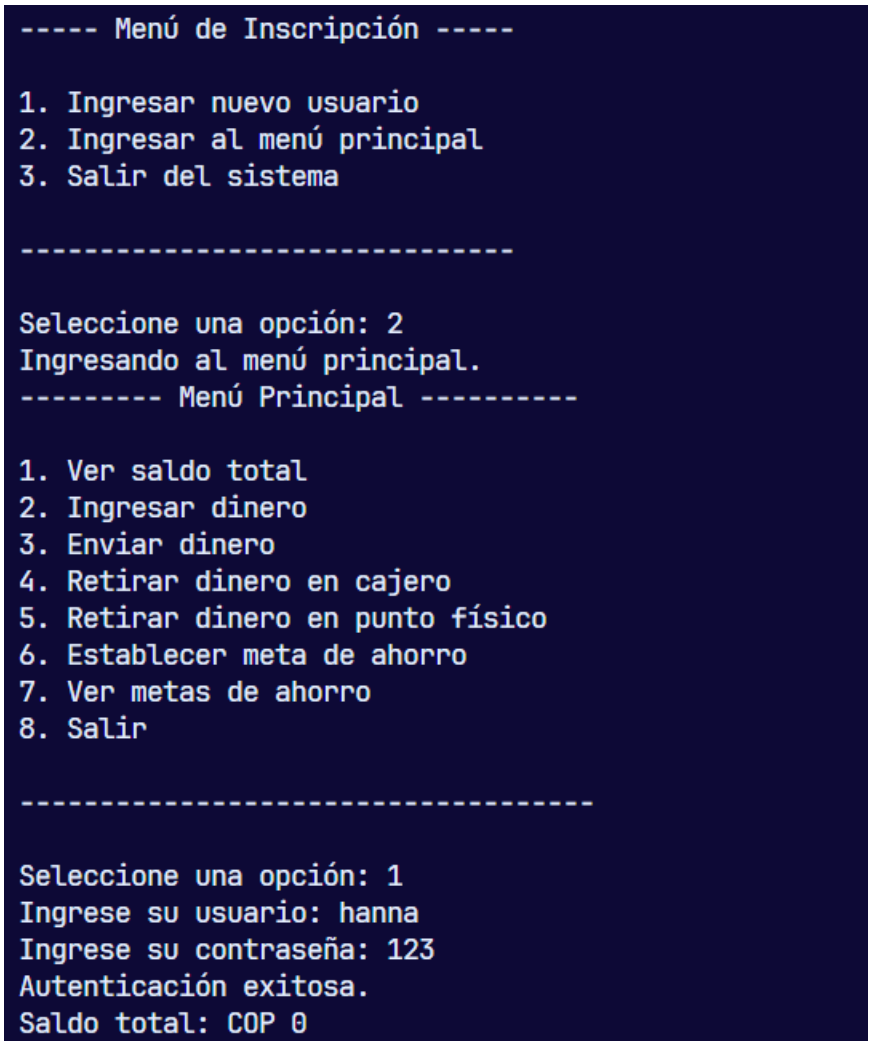
5. DEMOSTRACION DE PRUEBAS Y FUNCIONAMIENTO DEL CODIGO

5.1. Prueba 1



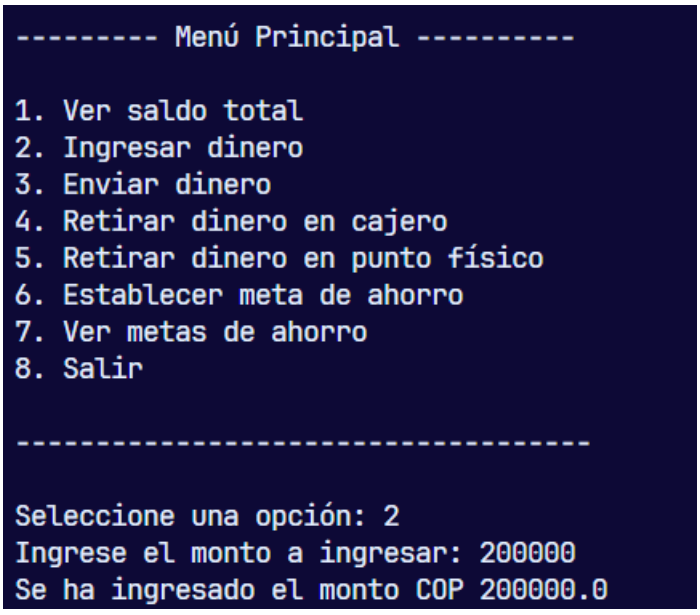
En esta primera prueba se demuestra que el código es funciona y que además de eso permite registrar a un usuario con éxito, con su respectiva contraseña y usuario.

5.2. Prueba 2



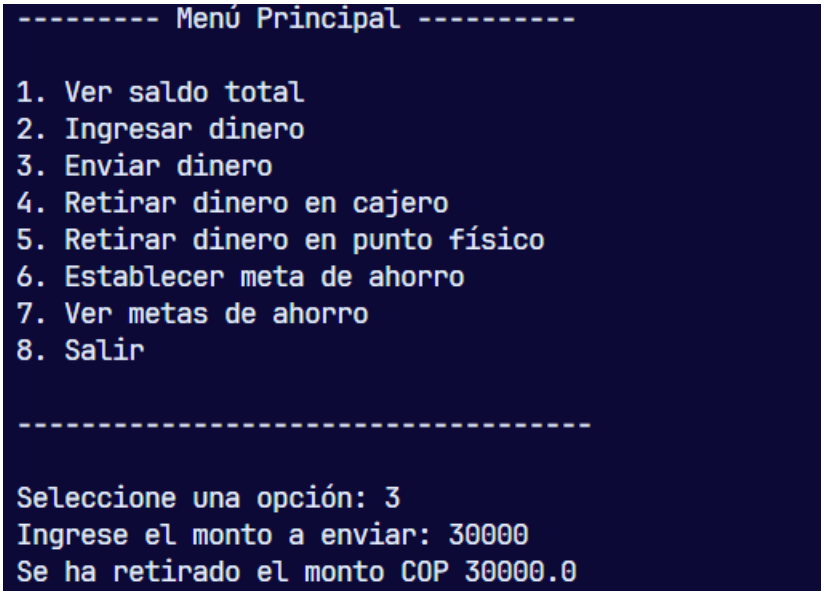
En esta prueba se puede apreciar ya el ingreso del usuario al menú principal en donde se le pide una autenticación previa antes de poder realizar cualquier acción del menú, estas medidas se tomaron debido a que se esta simulando una entidad bancaria digital. Además de eso se demuestra la primera opción del menú para ver saldo el cual esta en 0 ya que no se ha realizado ningún movimiento.

5.3. Prueba 3



En esta prueba se demuestra el funcionamiento de la segunda opción del menú, en donde se puede apreciar que se realizó un ingreso de dinero. Al hacer se actualiza el saldo total, pasa de 0 al monto ingresado.

5.4. Prueba 4



Se realizan transacciones como enviar dinero, la cual se puede complementar y mejorar mas adelante. Por motivos de productividad esta es una versión super simple de eso

5.5. Prueba 5

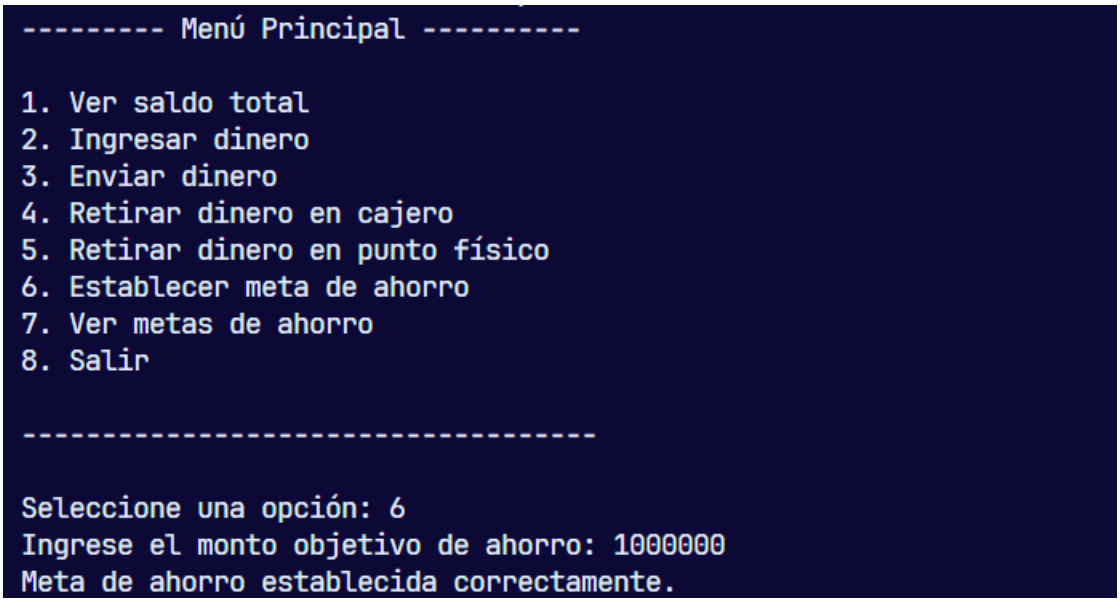
```
--------- Menú Principal ----------

1. Ver saldo total
2. Ingresar dinero
3. Enviar dinero
4. Retirar dinero en cajero
5. Retirar dinero en punto físico
6. Establecer meta de ahorro
7. Ver metas de ahorro
8. Salir

-----------------------------------

Seleccione una opción: 4
Ingrese el monto a retirar en cajero: 15000
Se han retirado COP 15000.0 en cajero.
```

En esta prueba se retira plata en el cajero, e igual que el anterior tiene oportunidades de mejora ya que se puede hacer una lista de cajeros en donde se pueda retirar entre otras.

5.6. Prueba 6

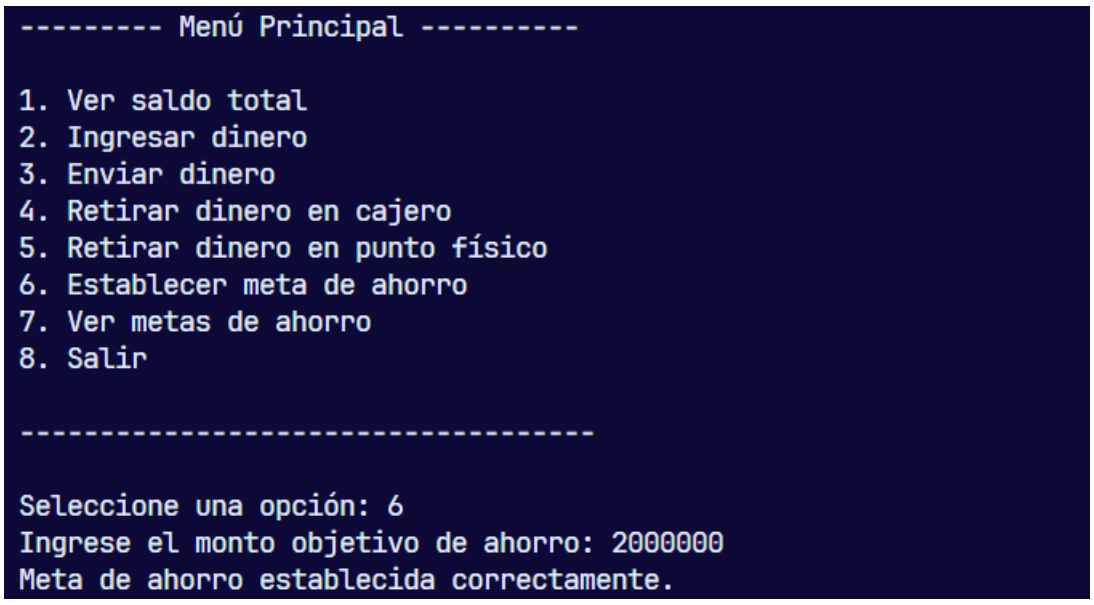
```
--------- Menú Principal ----------

1. Ver saldo total
2. Ingresar dinero
3. Enviar dinero
4. Retirar dinero en cajero
5. Retirar dinero en punto físico
6. Establecer meta de ahorro
7. Ver metas de ahorro
8. Salir

-----------------------------------

Seleccione una opción: 5
Ingrese el monto a retirar en punto físico: 15000
Se han retirado COP 15000.0 en punto físico.
```

5.7. Prueba 7



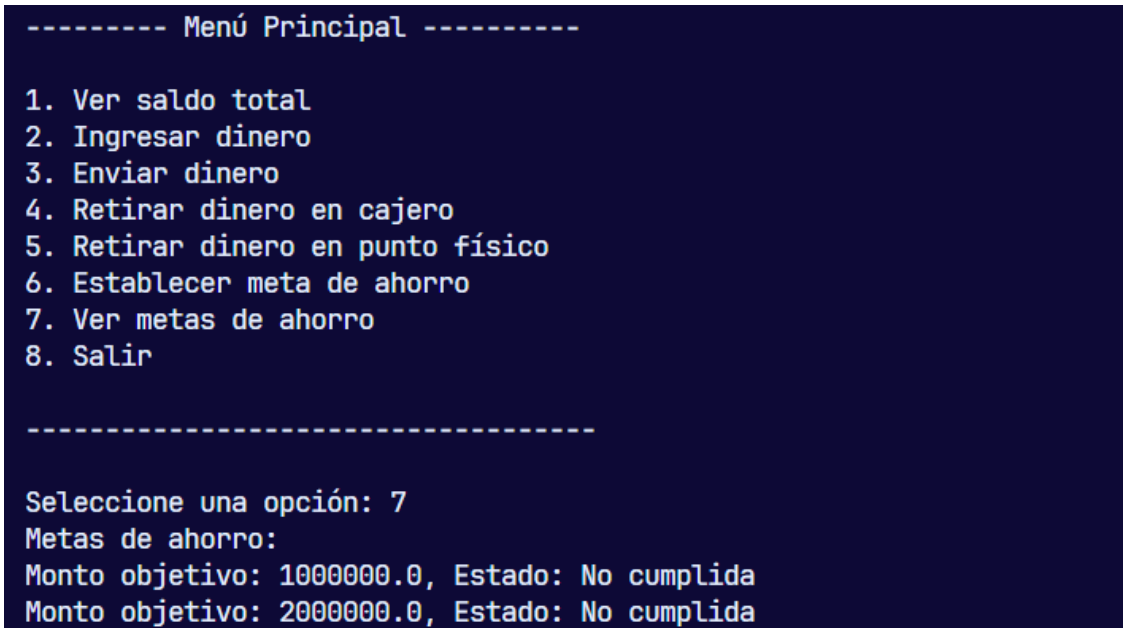
En esta prueba se ingresa el monto objetivo de ahorro, se realizan dos metas para propósito de demostración de funcionalidad de código.

5.8. Prueba 8



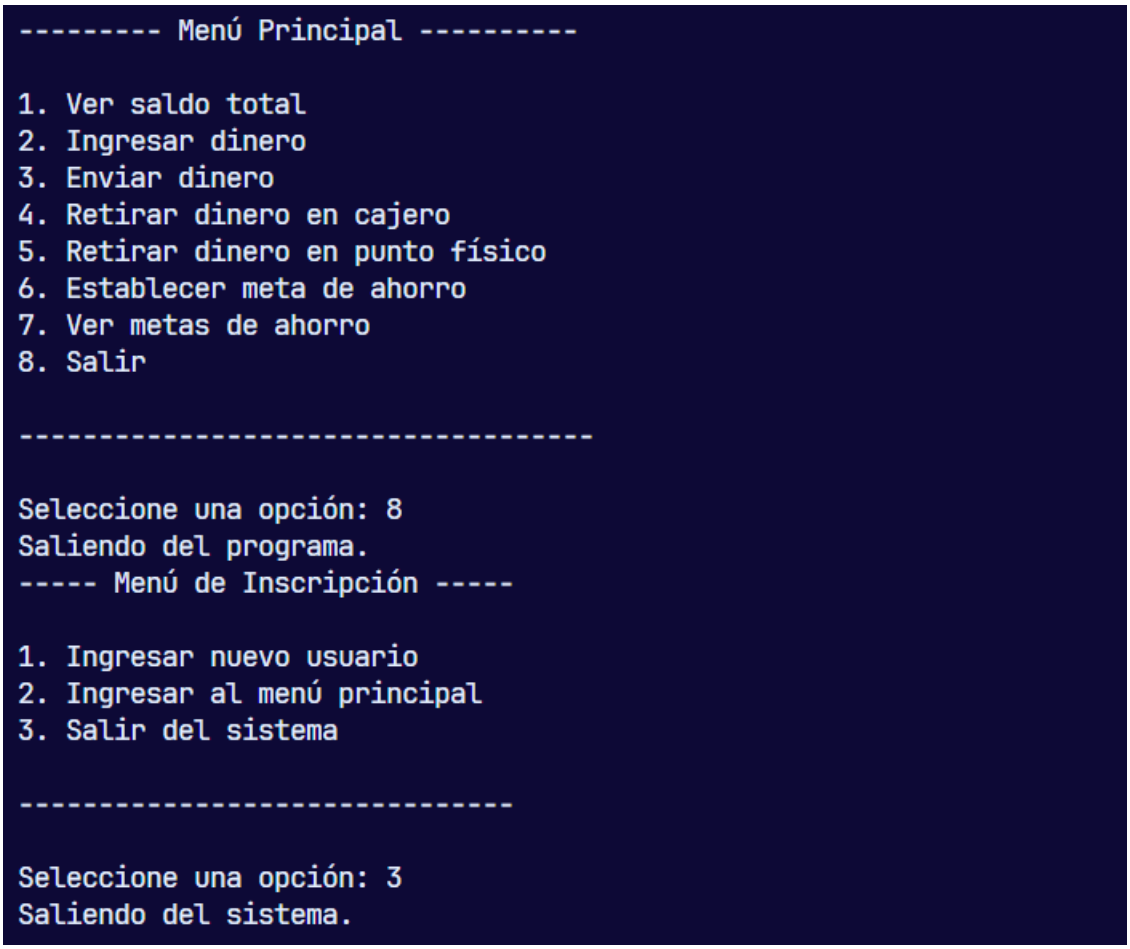
En esta prueba se ingresa el monto objetivo de ahorro diferente al anterior, se realizan dos metas para propósito de demostración de funcionalidad de código.

5.9. Prueba 9



En esta prueba se puede verificar los montos objetivos, las metas de ahorro disponibles para este usuario además de el estado de estas.

5.10. Prueba 10



En esta prueba se sale del menú principal para entrar al menú de inscripciones nuevamente, en donde se puede inscribir un usuario diferente o salir del programa o incluso volver a ingresar al menú principal.

6. DECISIÓN DE DISEÑO O CONSIDERACION IMPORTANTE

En este código una de las principales decisiones de diseño fue crear un menú de inscripción de usuario para que no fuera tan fácil hacer cambios en la cuenta, y que fuera solo posible únicamente si se ingresaba con un usuario y una contraseña, que fueron colocadas en el momento de la inscripción del usuario.

7. PILARES DE POO IMPLEMENTADOS

7.1. Abstraccion: Se define la clase MetaRecaudo, que representa una meta de ahorro con atributos como monto_objetivo, fecha_establecimiento, estado, y ahorro_actual.

Se define la clase BancoDigital, que representa un banco digital con atributos y métodos para manejar operaciones bancarias y usuarios.

7.2. Encapsulamiento: Los atributos de las clases están encapsulados dentro de ellas y se acceden a través de métodos de la clase (self.atributo).

8. CONCLUCIONES

Al analizar y trabajar en el código anterior, aprendimos varios conceptos importantes de la Programación Orientada a Objetos (POO). En primer lugar, comprendimos la importancia de la abstracción al modelar entidades como metas de ahorro y bancos digitales como clases con atributos y métodos específicos. El encapsulamiento me permitió organizar la lógica del programa de manera estructurada, asegurando que los datos y funcionalidades estén encapsulados dentro de las clases correspondientes. También pudimos apreciar cómo el uso de métodos permite la reutilización de código y facilita la modularidad y mantenimiento del sistema. Aunque el código no utiliza herencia ni polimorfismo, la estructura general del programa proporcionó una base sólida para comprender estos conceptos y su potencial en proyectos más complejos. En resumen, trabajar con este código nos proporcionó una comprensión práctica de los principios fundamentales de la POO y cómo aplicarlos en el desarrollo de sistemas software.