Sistema Bancario en Python

Descripción General

El sistema bancario implementado en Python permite la gestión de usuarios y sus cuentas bancarias. Incluye funcionalidades para registrar usuarios, iniciar sesión, realizar depósitos, retiros y transferencias de dinero. Además, el sistema demuestra el uso de conceptos de programación orientada a objetos como clases, objetos, abstracción de datos, encapsulamiento, herencia y polimorfismo.

Características Principales

Registro de Usuarios

- Los usuarios pueden registrarse proporcionando un nombre, nombre de usuario y contraseña.
- Los usuarios pueden seleccionar el tipo de cuenta (ahorros o corriente).

Inicio de Sesión

- Los usuarios pueden iniciar sesión utilizando su nombre de usuario y contraseña.

Operaciones Bancarias

- **Mostrar Saldo**: Los usuarios pueden ver el saldo de su cuenta.
- **Depositar Dinero**: Los usuarios pueden depositar dinero en su cuenta.
- **Retirar Dinero**: Los usuarios pueden retirar dinero de su cuenta, sujeto a las restricciones del tipo de cuenta.
- **Transferir Dinero**: Los usuarios pueden transferir dinero a otras cuentas dentro del sistema.

Clases y Encapsulamiento

Clase `Usuario`

- **Atributos Privados**:
  - `__nombre`: Nombre del usuario.
  - `__username`: Nombre de usuario.
  - `__password`: Contraseña del usuario.
- **Métodos Públicos**:
  - `verificar_password(password)`: Verifica si la contraseña proporcionada coincide con la almacenada.
  - `get_nombre()`: Devuelve el nombre del usuario.
  - `get_username()`: Devuelve el nombre de usuario.
  - `get_password()`: Devuelve la contraseña del usuario.
  - `to_string()`: Devuelve una representación en cadena del usuario.

Clase `Cuenta` (Hereda de `Usuario`)

- **Atributos Privados**:
  - `__saldo`: Saldo de la cuenta.
- **Métodos Públicos**:
  - `depositar(monto)`: Deposita un monto en la cuenta.
  - `retirar(monto)`: Retira un monto de la cuenta si hay suficiente saldo.
  - `mostrar_saldo()`: Devuelve el saldo de la cuenta.
  - `get_saldo()`: Devuelve el saldo de la cuenta.
  - `to_string()`: Devuelve una representación en cadena de la cuenta.

Subclases de `Cuenta`

Clase `CuentaAhorros`

- **Métodos Públicos**:
  - `retirar(monto)`: Retira un monto de la cuenta si no excede el 80% del saldo disponible.

Clase `CuentaCorriente`

- **Atributos Privados**:
  - `__sobregiro`: Límite de sobregiro de la cuenta.
- **Métodos Públicos**:
  - `retirar(monto)`: Retira un monto de la cuenta si no excede el saldo disponible más el sobregiro.
  - `to_string()`: Devuelve una representación en cadena de la cuenta, incluyendo el sobregiro.

Clase `Banco`

- **Atributos Privados**:
  - `nombre_archivo`: Nombre del archivo donde se almacenan los datos de los usuarios.
  - `usuarios`: Diccionario que almacena los usuarios del banco.
- **Métodos Públicos**:
  - `registrar_usuario(tipo_cuenta, nombre, username, password)`: Registra un nuevo usuario.
  - `iniciar_sesion(username, password)`: Inicia sesión para un usuario existente.
  - `transferir_dinero(cuenta_origen, username_destino, monto)`: Transfiere dinero entre cuentas si hay suficiente saldo.

Polimorfismo en Accion

El sistema demuestra polimorfismo mediante el método `retirar`, que se comporta de manera diferente según el tipo de cuenta (`CuentaAhorros` o `CuentaCorriente`).

Encapsulamiento

Los atributos de las clases `Usuario` y `Cuenta` son privados y se accede a ellos a través de métodos públicos, protegiendo así los datos internos de las clases.

Ejecución del Programa

El programa principal presenta un menú para interactuar con el sistema bancario, permitiendo a los usuarios registrarse, iniciar sesión y realizar operaciones bancarias.

Ejemplo de Código

```
import os

# Clase base para abstracción de datos y encapsulamiento
class Usuario:
    def __init__(self, nombre, username, password):
        self.__nombre = nombre
        self.__username = username
        self.__password = password

    def verificar_password(self, password):
        return self.__password == password

    def get_nombre(self):
        return self.__nombre

    def get_username(self):
        return self.__username

    def get_password(self):
        return self.__password

    def to_string(self):
        return f"{self.__nombre},{self.__username},{self.__password}"

# Clase base para cuentas bancarias
class Cuenta(Usuario):
    def __init__(self, nombre, username, password, saldo=0.0):
        super().__init__(nombre, username, password)
        self.__saldo = saldo

    def depositar(self, monto):
        self.__saldo += monto

    def retirar(self, monto):
        if monto <= self.__saldo:
            self.__saldo -= monto
        else:
            raise ValueError("Saldo insuficiente")
```

```
    def mostrar_saldo(self):
        return self.__saldo

    def get_saldo(self):
        return self.__saldo

    def to_string(self):
        return f"{self.get_nombre()},{self.get_username()},
{self.get_password()},{self.__saldo}"

# Subclase de Cuenta para cuentas de ahorro
class CuentaAhorros(Cuenta):
    def __init__(self, nombre, username, password, saldo=0.0):
        super().__init__(nombre, username, password, saldo)

    def retirar(self, monto):
        if monto > self.mostrar_saldo() * 0.8:
            raise ValueError("No puede retirar más del 80% de su saldo en una
cuenta de ahorros")
        super().retirar(monto)

# Subclase de Cuenta para cuentas corrientes
class CuentaCorriente(Cuenta):
    def __init__(self, nombre, username, password, saldo=0.0, sobregiro=500.0):
        super().__init__(nombre, username, password, saldo)
        self.__sobregiro = sobregiro

    def retirar(self, monto):
        if monto > self.mostrar_saldo() + self.__sobregiro:
            raise ValueError("Saldo insuficiente incluso con sobregiro")
        self._Cuenta__saldo -= monto  # Acceder al saldo privado de la clase
base

    def to_string(self):
        return f"{self.get_nombre()},{self.get_username()},
{self.get_password()},{self.mostrar_saldo()},{self.__sobregiro}"

# Clase que maneja la gestión del banco
class Banco:
    def __init__(self, nombre_archivo):
        self.nombre_archivo = nombre_archivo
        self.usuarios = self._cargar_datos()

    def _cargar_datos(self):
        usuarios = {}
        if os.path.exists(self.nombre_archivo):
            with open(self.nombre_archivo, 'r') as file:
                for linea in file:
                    datos = linea.strip().split(',')
                    if len(datos) == 4:
                        nombre, username, password, saldo = datos
                        usuarios[username] = Cuenta(nombre, username, password,
float(saldo))
```

```
                    elif len(datos) == 5:
                        nombre, username, password, saldo, sobregiro = datos
                        usuarios[username] = CuentaCorriente(nombre, username,
password, float(saldo), float(sobregiro))
        return usuarios

    def _guardar_datos(self):
        with open(self.nombre_archivo, 'w') as file:
            for cuenta in self.usuarios.values():
                file.write(cuenta.to_string() + '\n')

    def registrar_usuario(self, tipo_cuenta, nombre, username, password):
        if username in self.usuarios:
            raise ValueError("El usuario ya existe")
        if tipo_cuenta == "ahorros":
            cuenta = CuentaAhorros(nombre, username, password)
        elif tipo_cuenta == "corriente":
            cuenta = CuentaCorriente(nombre, username, password)
        else:
            cuenta = Cuenta(nombre, username, password)
        self.usuarios[username] = cuenta
        self._guardar_datos()

    def iniciar_sesion(self, username, password):
        if username in self.usuarios:
            cuenta = self.usuarios[username]
            if cuenta.verificar_password(password):
                return cuenta
        raise ValueError("Usuario o contraseña incorrecta")

    def transferir_dinero(self, cuenta_origen, username_destino, monto):
        if username_destino in self.usuarios:
            cuenta_destino = self.usuarios[username_destino]
            if monto > cuenta_origen.mostrar_saldo():
                raise ValueError("Saldo insuficiente para la transferencia")
            cuenta_origen.retirar(monto)
            cuenta_destino.depositar(monto)
            self._guardar_datos()
        else:
            raise ValueError("El usuario destino no existe")

def menu_usuario(banco, cuenta):
    while True:
        print("\n--- Menú ---")
        print("1. Mostrar saldo")
        print("2. Depositar dinero")
        print("3. Retirar dinero")
        print("4. Transferir dinero")
        print("5. Salir")
        opcion = input("Seleccione una opción: ")

        if opcion == "1":
            print(f"Su saldo actual es: {cuenta.mostrar_saldo()}")
```

```
        elif opcion == "2":
            monto = float(input("Ingrese el monto a depositar: "))
            cuenta.depositar(monto)
            banco._guardar_datos()
            print("Depósito exitoso")
        elif opcion == "3":
            monto = float(input("Ingrese el monto a retirar: "))
            try:
                cuenta.retirar(monto)
                banco._guardar_datos()
                print("Retiro exitoso")
            except ValueError as e:
                print(e)
        elif opcion == "4":
            username_destino = input("Ingrese el nombre de usuario destino: ")
            monto = float(input("Ingrese el monto a transferir: "))
            try:
                banco.transferir_dinero(cuenta, username_destino, monto)
                print("Transferencia exitosa")
            except ValueError as e:
                print(e)
        elif opcion == "5":
            break
        else:
            print("Opción no válida")

# Función principal para interactuar con el sistema
def main():
    banco = Banco("usuarios.txt")

    while True:
        print("\n--- Banco ---")
        print("1. Registrar usuario")
        print("2. Iniciar sesión")
        print("3. Salir")
        opcion = input("Seleccione una opción: ")

        if opcion == "1":
            tipo_cuenta = input("Tipo de cuenta (ahorros/corriente): ").lower()
            nombre = input("Ingrese su nombre: ")
            username = input("Ingrese un nombre de usuario: ")
            password = input("Ingrese una contraseña: ")
            try:
                banco.registrar_usuario(tipo_cuenta, nombre, username,
password)
                print("Usuario registrado exitosamente")
            except ValueError as e:
                print(e)
        elif opcion == "2":
            username = input("Ingrese su nombre de usuario: ")
            password = input("Ingrese su contraseña: ")
            try:
                cuenta = banco.iniciar_sesion(username, password)
```

```
                menu_usuario(banco, cuenta)
            except ValueError as e:
                print(e)
        elif opcion == "3":
            break
        else:
            print("Opción no válida")

if __name__ == "__main__":
    main()
```

Archivo de Datos

Los datos de los usuarios se almacenan en un archivo `usuarios.txt`, donde cada línea representa un usuario con su información separada por comas.
