codigo prueba

data_clientes.py

```
"""
data_clientes.py
================

Este mÃ³dulo gestiona la lectura de datos de clientes y cuentas bancarias desde archivos CSV.

Funcionalidades:
----------------
- Leer los datos de clientes y cuentas desde rutas especificadas en variables de entorno.
- Manejar errores comunes al leer archivos CSV (archivo no encontrado, archivo vacÃo, errores de parseo).

Clases:
-------
- Read_Cliente:
    - MÃ©todos:
        - read_data(): Devuelve dos DataFrames de pandas con los datos de clientes y cuentas.

Entradas:
---------
- Rutas de archivos CSV definidas en variables de entorno:
    - path_clientes
    - path_cuentas

Dependencias:
-------------
- pandas
- python-dotenv
- os
- pathlib

Autor: Tu Nombre
Fecha: 2024

"""

import pandas as pd
from dotenv import load_dotenv
import os
from pathlib import Path


class Read_Cliente:
    def __init__(self):
        load_dotenv()
        path_clientes = os.getenv('path_clientes')
        path_cuentas = os.getenv('path_cuentas')
        self.file_clientes_path = path_clientes
        self.file_cuentas_path = path_cuentas


    def read_data(self):
        try:
            data_clientes = pd.read_csv(self.file_clientes_path)
            data_cuentas = pd.read_csv(self.file_cuentas_path)

            return data_clientes, data_cuentas
        except FileNotFoundError:
            print(f"Error: The file at {self.file_clientes_path} was not found.")
            print(f"Error: The file at {self.file_cuentas_path} was not found.")
            return None
        except pd.errors.EmptyDataError:
            print("Error: The file is empty.")
            return None
        except pd.errors.ParserError:
            print("Error: There was a parsing error while reading the file.")
            return None
        except Exception as e:
            print(f"An unexpected error occurred: {e}")
            return None
```

cajero.py

clientes.py

```
"""
clientes.py
===========

Este mÃ³dulo implementa la lÃ³gica para la gestiÃ³n de clientes y operaciones bancarias.

Funcionalidades:
----------------
- Crear un nuevo cliente y su cuenta asociada.
```

```
- Realizar consignaciones y retiros en cuentas existentes.
- Consultar el saldo de una cuenta por identificaciÃ³n.

Clases:
-------
- new_client:
    - Permite crear un nuevo cliente y guardar sus datos en el archivo correspondiente.
- manatgement_client:
    - MÃ©todos:
        - consignation(identificacion): Consigna dinero en la cuenta asociada.
        - withdrawal(identificacion): Retira dinero de la cuenta asociada.
        - query_client(identificacion): Consulta el saldo de la cuenta asociada.

Entradas:
---------
- Datos personales del cliente.
- IdentificaciÃ³n del cliente.
- Valor a consignar o retirar.

Dependencias:
-------------
- pandas
- data_clientes.Read_Cliente

"""

from data.data_clientes import Read_Cliente
import pandas as pd
from src.operation import Operacion
class new_client():
    def __init__(self, Nombres, Apellidos, Identificacion, Movil, Correo, tipo_cuenta):
        self.Nombres = Nombres
        self.Apellidos = Apellidos
        self.Identificacion = Identificacion
        self.Movil = Movil
        self.Correo = Correo
        self.tipo_cuenta = tipo_cuenta

    def create_client(self):
        reader = Read_Cliente()
        df_client, df_cuenta = reader.read_data()
        id_client = df_client['ID_cliente'].iloc[-1] + 1
        ID_cuenta = df_client['ID_cuenta'].iloc[-1] + 1


        new_row_client = {
            'Nombres': self.Nombres,
            'Apellidos': self.Apellidos,
            'Identificacion': self.Identificacion,
            'Movil': self.Movil,
            'Correo': self.Correo,
            'Tipo cuenta': self.tipo_cuenta,
            'ID_cliente' : id_client,
            'ID_cuenta': ID_cuenta
        }

        df_client = pd.concat([df_client, pd.DataFrame([new_row_client])], ignore_index=True)
        df_client.to_csv(reader.file_clientes_path, index=False)
        print('Se creo correctamete el cliente con identificacion:', self.Identificacion)
        return 'sl'

class manatgement_client():
    def __init__(self):
        self.read_data = Read_Cliente()
        self.data_clientes, self.data_cuentas = self.read_data.read_data()

    def realizar_operacion(self, identificacion, operacion: "Operacion", value=None):
       return operacion(self.data_clientes, self.data_cuentas, self.read_data.file_cuentas_path).ejecutar(identificacion, value)
```

main.py

```
"""
main.py
========

Este mÃ³dulo implementa la interfaz principal para el sistema de gestiÃ³n de clientes de un cajero bancario.

Funcionalidades:
----------------
- Crear un nuevo cliente y su cuenta asociada.
- Consultar el saldo de una cuenta por identificaciÃ³n.
- Consignar dinero en una cuenta existente.
- Retirar dinero de una cuenta existente.

Uso:
----
Al ejecutar el script, se muestra un menÃº interactivo en consola donde el usuario puede seleccionar la operaciÃ³n deseada.

Clases utilizadas:
------------------
- new_client: Permite crear un nuevo cliente.
- manatgement_client: Permite consultar saldo, consignar y retirar dinero.
```

```
Entradas:
---------
- IdentificaciÃ³n del cliente (CC)
- Datos personales para la creaciÃ³n de cliente
- Valor a consignar o retirar



"""

from src.clientes import new_client, manatgement_client
import os
from src.operation import ConsultaSaldo, Consignacion, Retiro

while True:
   # os.system('cls')  # Limpia la consola en Windows
    print("Bienvenido al sistema de gestiÃ³n de clientes.")
    Identificacion = int(input("CC:\n"))
    print("Indique que operacion desea realizar:")
    print("1. Crear cliente")
    print("2. Consultar saldo")
    print("3. Consignar dinero")
    print("4. Retirar dinero")

    opcion = input("Seleccione una opciÃ³n:\n")
    try:
        opcion = int(opcion)
    except ValueError:
        print("Por favor ingrese un nÃºmero vÃ¡lido.")
        continue
    client = manatgement_client()
    if opcion == 1:
        print("Crear cliente")

        nombre = input("Nombres:\n")
        apellido = input("Apellidos:\n")
        movil = int(input("Movil:\n"))
        correo = input("Correo:\n")
        tipo_cuenta = input("Tipo de cuenta:\n")

        client = new_client(
            Nombres=nombre,
            Apellidos=apellido,
            Identificacion=Identificacion,
            Movil=movil,
            Correo=correo,
            tipo_cuenta=tipo_cuenta
        )
        client.create_client()
    elif opcion == 2:
        client.realizar_operacion(Identificacion, ConsultaSaldo)

    elif opcion == 3:
        client.realizar_operacion(Identificacion, Consignacion)

    elif opcion == 4:
        client.realizar_operacion(Identificacion, Retiro)
```

operation.py

```
# operaciones.py
from abc import ABC, abstractmethod
import pandas as pd

class Operacion(ABC):
    """Clase abstracta que define la interfaz comÃºn para todas las operaciones"""

    def __init__(self, data_clientes, data_cuentas, file_cuentas_path):
        self.data_clientes = data_clientes
        self.data_cuentas = data_cuentas
        self.file_cuentas_path = file_cuentas_path

    @abstractmethod
    def ejecutar(self, identificacion):
        pass


class Consignacion(Operacion):
    def ejecutar(self, identificacion, value):
        row_client = self.data_clientes[self.data_clientes['Identificacion'] == identificacion]
        if len(row_client) == 0:
            print(f"No se encontrÃ³ cuenta asociada a la identificaciÃ³n {identificacion}.")
            return

        id_cuenta = row_client['ID_cuenta'].values[0]
        row_cuenta = self.data_cuentas[self.data_cuentas['ID_Cuenta'] == id_cuenta]
        #value = int(input("Valor a consignar:\n"))
        last_row = row_cuenta.sort_values('Fecha_operacion', ascending=False)

        if last_row.empty:
            nuevo_saldo = value
        else:
            nuevo_saldo = last_row['Nuevo_saldo'].values[0] + value
```

```
        new_operation = {
            'ID_Cuenta': id_cuenta,
            'Saldo' : nuevo_saldo,
            'Tipo_movimiento': 1,
            'valor_movimiento': value,
            'Nuevo_saldo': nuevo_saldo,
            'Fecha_operacion' : pd.Timestamp.now()
        }

        df_cuenta = pd.concat([self.data_cuentas, pd.DataFrame([new_operation])], ignore_index=True)
        df_cuenta.to_csv(self.file_cuentas_path, index=False)
        print(f'Se ha consignado {value} a la cuenta {id_cuenta}.')
        return nuevo_saldo

class Retiro(Operacion):
    def ejecutar(self, identificacion):
        row_client = self.data_clientes[self.data_clientes['Identificacion'] == identificacion]
        if len(row_client) == 0:
            print(f"No se encontrÃ³ cuenta asociada a la identificaciÃ³n {identificacion}.")
            return

        id_cuenta = row_client['ID_cuenta'].values[0]
        row_cuenta = self.data_cuentas[self.data_cuentas['ID_Cuenta'] == id_cuenta]
        value = int(input("Valor a retirar:\n"))
        last_row = row_cuenta.sort_values('Fecha_operacion', ascending=False)

        if last_row.empty:
            print("No hay movimientos previos, no se puede retirar.")
            return

        nuevo_saldo = last_row['Nuevo_saldo'].values[0] - value
        if nuevo_saldo < 0:
            print("Saldo insuficiente.")
            return

        new_operation = {
            'ID_Cuenta': id_cuenta,
            'Saldo' : nuevo_saldo,
            'Tipo_movimiento': 2,
            'valor_movimiento': value,
            'Nuevo_saldo': nuevo_saldo,
            'Fecha_operacion' : pd.Timestamp.now()
        }

        df_cuenta = pd.concat([self.data_cuentas, pd.DataFrame([new_operation])], ignore_index=True)
        df_cuenta.to_csv(self.file_cuentas_path, index=False)
        print(f'Se ha retirado {value} de la cuenta {id_cuenta}.')


class ConsultaSaldo(Operacion):
    def ejecutar(self, identificacion):
        row_client = self.data_clientes[self.data_clientes['Identificacion'] == identificacion]
        if len(row_client) == 0:
            return "a1"

        id_cuenta = row_client['ID_cuenta'].values[0]
        row_cuenta = self.data_cuentas[self.data_cuentas['ID_Cuenta'] == id_cuenta]
        if row_cuenta.empty:
            print("No movimientos")
            return "a2"
        else:
            saldo = row_cuenta.sort_values("Fecha_operacion", ascending=True)['Saldo'].iloc[-1]
            print(f"{saldo}")
            return saldo
```