Directory structure:
```
└── mevaktecno-vane-sistema_bancario/
    ├── README.md
    ├── main.py
    ├── requirements.txt
    ├── src/
    │   ├── __init__.py
    │   ├── app_banco.py
    │   ├── cliente.py
    │   ├── cuenta.py
    │   ├── cuenta_ahorro.py
    │   ├── exportar_datos_a_pdf.py
    │   ├── tarjeta.py
    │   └── transaccion.py
    └── tests/
        ├── __init__.py
        ├── test_cliente.py
        ├── test_cuenta.py
        ├── test_cuenta_ahorro.py
        ├── test_tarjeta.py
        └── test_transaccion.py
```

==
FILE: README.md
==

Sistema Bancario

Proyecto integrador final de **Laboratorio I** y **Control de Versiones**

Integrantes

- **Mara Vanesa San Martín**
- **Daniel Ricardo González**

Descripción del Proyecto

Este proyecto modela un **sistema bancario** utilizando los principios de la **Programación Orientada a Objetos (POO)** con Python, integrando además control de versiones con Git/GitHub, pruebas unitarias, manejo de errores, generación de reportes PDF y una **interfaz gráfica desarrollada con Flet**.

Objetivos del Trabajo

- Aplicar **abstracción, herencia, polimorfismo y encapsulamiento** en la implementación de las clases.
- Desarrollar un sistema funcional que permita:
  - Registrar clientes.
  - Crear cuentas bancarias y tarjetas.
  - Realizar operaciones (depósitos, retiros, compras, pagos).
  - Exportar la información a un archivo PDF.
- Implementar una **interfaz visual amigable**.

- Usar control de versiones (**Git y GitHub**) con trabajo colaborativo.
- Documentar y gestionar tareas mediante **Trello**.

Clases Implementadas

`Cliente`
Representa a un cliente del banco con atributos privados (`nombre`, `apellido`, `dni`).

- Validación de datos (solo letras en nombre y apellido, DNI numérico).
- Métodos `get_` y `mostrar_datos()` para mostrar la información.

`Cuenta`
Maneja las operaciones bancarias principales.

- Métodos: `depositar()`, `retirar()`.
- Control de errores:
  - `SaldoInsuficienteError`
  - Validación de montos y tipos de datos.
- Registra transacciones automáticamente.

`CuentaAhorro`
Hereda de `Cuenta` e implementa polimorfismo aplicando una **tasa de interés**.

`Tarjeta`
Permite registrar compras y pagos de crédito, con límite configurable.

`Transaccion`
Registra la información de cada operación

Interfaz con Flet

Se desarrolló una **interfaz gráfica completa** que permite:
- Registrar nuevos clientes.
- Crear cuentas y tarjetas.
- Realizar operaciones bancarias.
- Visualizar movimientos.
- Exportar toda la información a PDF.

Generación de PDF
Se implementó la función `generar_pdf_reporte()` con la librería **FPDF**.
El archivo PDF exporta datos de:
- Cliente

- Cuenta y saldo actual
- Movimientos y transacciones

Pruebas Unitarias (pytest)
Cada clase principal cuenta con su archivo de prueba:
- `test_cliente.py`
- `test_cuenta.py`
- `test_tarjeta.py`

Ejemplo para ejecutar los tests:
```bash
pytest -v

Manejo de Errores

Se utiliza try-except en toda la app

Validación de datos inválidos al crear clientes o cuentas.

Control de saldos insuficientes y límites de tarjeta.


Requerimientos

Archivo requirements.txt:

flet
fpdf
pytest

Control de Versiones y Trabajo Colaborativo

Proyecto gestionado con Git y GitHub.

Cada integrante trabajó en su rama personal (Vane y Daniel).

Se realizaron commits descriptivos y frecuentes.

Las tareas fueron organizadas en Trello, con estados:

* Pendiente

* En curso

* Finalizada

Documentos adicionales

trello_board.pdf: Export del tablero Trello usando Pretty Print.

docs/gitdiagram.png: Diagrama visual del flujo de Git (ramas, merges, commits).
```

docs/gitingest.md: Explicación del proceso de integración y control de versiones.

Ejecución del Proyecto

Clonar el repositorio:

```
git clone https://github.com/mevaktecno-vane/sistema_bancario.git
```

Entrar al proyecto:

```
cd sistema_bancario
```

Activar entorno virtual:

```
python -m venv venv
venv\Scripts\activate  # En Windows
```

Instalar dependencias:

```
pip install -r requirements.txt
```

Ejecutar la aplicación:

```
python main.py
```

Evaluación Final

El proyecto cumple con:

Principios de POO.

Manejo de errores.

Pruebas unitarias.

Interfaz gráfica funcional.

Control de versiones y documentación.

Trabajo colaborativo documentado.

```
================================================
FILE: main.py
================================================
from src.cliente import Cliente
```

```
from src.cuenta import Cuenta, SaldoInsuficienteError
from src.cuenta_ahorro import CuentaAhorro
from src.tarjeta import Tarjeta, LimiteExcedidoError


# ====== ARREGLOS ======
clientes = []
cuentas = []
tarjetas = []


# ====== FUNCIONES PRINCIPALES ======

def guardar_cliente(nombre, apellido, dni):
    """Agrega un nuevo cliente a la lista."""
    try:
        if not nombre or not apellido or not dni:
            raise ValueError("Todos los campos del cliente son
obligatorios.")
        if any(c.get_dni() == dni for c in clientes):
            raise ValueError("Ya existe un cliente con ese DNI.")
        cliente = Cliente(nombre, apellido, dni)
        clientes.append(cliente)
        print(f"Cliente agregado: {cliente}\n")
        return cliente
    except ValueError as e:
        print(f"Error: {e}\n")


def mostrar_clientes():
    """Muestra todos los clientes registrados."""
    try:
        if len(clientes) == 0:
            raise IndexError("No hay clientes cargados.")
        print("\n=== LISTADO DE CLIENTES ===")
        for cliente in clientes:
            print(cliente)
        print()
    except IndexError as e:
        print(f"Atención: {e}\n")


def guardar_cuenta(nro_cuenta, cliente, saldo=0.0):
    """Agrega una cuenta corriente."""
    try:
        cuenta = Cuenta(nro_cuenta, cliente, saldo)
        cuentas.append(cuenta)
        print(f"Cuenta creada: {nro_cuenta}\n")
        return cuenta
    except ValueError as e:
        print(f"Error: {e}\n")


def mostrar_cuentas():
```

```
    """Muestra todas las cuentas creadas."""
    try:
        if len(cuentas) == 0:
            raise IndexError("No hay cuentas cargadas.")
        print("\n=== LISTADO DE CUENTAS ===")
        for cuenta in cuentas:
            print(
                f"{cuenta.get_nro_cuenta()} -
{cuenta.get_cliente().get_dni()} - ${cuenta.get_saldo()}")
        print()
    except IndexError as e:
        print(f"Atención: {e}\n")


def guardar_tarjeta(numero, cliente, limite):
    """Agrega una nueva tarjeta de crédito."""
    try:
        tarjeta = Tarjeta(numero, cliente, limite)
        tarjetas.append(tarjeta)
        print(f"Tarjeta agregada: {numero}\n")
        return tarjeta
    except ValueError as e:
        print(f"Error: {e}\n")


def mostrar_tarjetas():
    """Muestra todas las tarjetas registradas."""
    try:
        if len(tarjetas) == 0:
            raise IndexError("No hay tarjetas registradas.")
        print("\n=== LISTADO DE TARJETAS ===")
        for tarjeta in tarjetas:
            print(
                f"{tarjeta.get_numero()} -
{tarjeta.get_cliente().get_dni()} - Saldo:
${tarjeta.get_saldo_actual()}")
        print()
    except IndexError as e:
        print(f"Atención: {e}\n")


# ====== BLOQUE PRINCIPAL ======

def main():
    print("=== SISTEMA BANCARIO (Versión evaluable) ===\n")

    # Carga simulada de datos
    cliente1 = guardar_cliente("Juan", "Pérez", "12345678")
    cliente2 = guardar_cliente("Ana", "Gómez", "87654321")

    guardar_cuenta("001", cliente1, 1500)
    guardar_cuenta("002", cliente2, 2000)

    guardar_tarjeta("1111-2222-3333-4444", cliente1, 5000)
```

```
    guardar_tarjeta("9999-8888-7777-6666", cliente2, 3000)

    # Mostrar los datos cargados
    mostrar_clientes()
    mostrar_cuentas()
    mostrar_tarjetas()


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

```
================================================
FILE: requirements.txt
================================================
flet
fpdf
pytest
```

```
================================================
FILE: src/__init__.py
================================================
[Empty file]
```

```
================================================
FILE: src/app_banco.py
================================================
import flet as ft
import threading
from src.cliente import Cliente
from src.cuenta import Cuenta, SaldoInsuficienteError
from src.tarjeta import Tarjeta
from src.transaccion import Transaccion
from src.cuenta_ahorro import CuentaAhorro
from src.exportar_datos_a_pdf import generar_pdf_reporte

class SaldoInsuficienteError(Exception): pass
class LimiteExcedidoError(Exception): pass

# --- Función Principal (main)---
def main(page: ft.Page):
    page.title = "Registro Financiero POO"
    page.vertical_alignment = ft.MainAxisAlignment.START
    page.padding = 30
    page.scroll = ft.ScrollMode.ADAPTIVE


    estado = {"cliente": None, "cuenta": None, "tarjeta": None}
    clientes_registrados = []
```

```
    lista_clientes_column = ft.Column(scroll=ft.ScrollMode.ALWAYS,
spacing=5, height=200)
    btn_nuevo_cliente = ft.ElevatedButton(text="+ Registrar Nuevo
Cliente", icon=ft.Icons.PERSON_ADD)

    # Botón para exportar datos a PDF
    btn_exportar_pdf = ft.ElevatedButton(
        text="Exportar datos en PDF",
        icon=ft.Icons.PICTURE_AS_PDF,

        bgcolor=ft.Colors.RED_700,
        color=ft.Colors.WHITE
    )
    exportar_container = ft.Container(
        content=btn_exportar_pdf,
        alignment=ft.alignment.center,
        padding=ft.padding.only(top=30)
    )

    # --- Funciones Auxiliares ---
    def mostrar_notificacion(texto, color=ft.Colors.GREEN_700,
duracion=3000):
        page.snack_bar = ft.SnackBar(ft.Text(texto), bgcolor=color,
duration=duracion)
        page.snack_bar.open = True
        page.update()


    def actualizar_historial_transacciones():
        historial_column.controls.clear()
        if estado["cuenta"]:
            transacciones = estado["cuenta"].get_transacciones()
            if not transacciones:
                 historial_column.controls.append(ft.Text("No hay
transacciones registradas.", italic=True))
            else:
                for t in reversed(transacciones):
                    color = ft.Colors.GREEN_700 if t.get_tipo() in
["deposito", "interes"] else ft.Colors.RED_700
                    historial_column.controls.append(ft.Text(t.__str__(),
size=12, color=color, font_family="monospace"))
        page.update()

    def actualizar_historial_tarjeta():

        historial_tarjeta_column.controls.clear()

        if estado["tarjeta"]:
            # Obtener movimientos de la tarjeta
            movimientos = estado["tarjeta"].get_movimientos()
            if not movimientos:
                historial_tarjeta_column.controls.append(ft.Text("No hay
movimientos registrados.", italic=True))
            else:
```

```
                # Mostrar movimientos en orden inverso (más reciente
primero)
                for fecha, monto, tipo in reversed(movimientos):

                    color = ft.Colors.RED_700 if tipo == "Compra" else
ft.Colors.GREEN_700

                    movimiento_str = f"{fecha.strftime('%Y-%m-%d
%H:%M:%S')} | {tipo.upper():<10} | ${monto:.2f}"

                    historial_tarjeta_column.controls.append(
                        ft.Text(movimiento_str, size=12, color=color,
font_family="monospace")
                    )
        page.update()
    def actualizar_saldo_cuenta():
        if estado["cuenta"]:
            lbl_saldo_cuenta.value = f"Saldo Actual:
${estado['cuenta'].get_saldo():.2f}"
            actualizar_historial_transacciones()
            toggle_retiro_button()
        page.update()

    def actualizar_saldo_tarjeta():
        if estado["tarjeta"]:
            lbl_limite_tarjeta.value = f"Límite Total:
${estado['tarjeta'].get_limite():.2f}"
            lbl_saldo_tarjeta.value = f"Deuda Actual:
${estado['tarjeta'].get_saldo_actual():.2f}"
            toggle_compra_button()
        page.update()

    def toggle_retiro_button(e=None):
        """Habilita o deshabilita el botón de retiro basado en el saldo y
el monto ingresado."""
        if estado["cuenta"] is None:
            # Si no hay cuenta, siempre debe estar deshabilitado
            btn_retirar.disabled = True
            return
        # Solo si hay cuenta, intentamos obtener el monto
        try:
            monto_ingresado = float(txt_monto_cuenta.value)
            saldo_actual = estado["cuenta"].get_saldo()

            # La lógica clave:
            if monto_ingresado > 0 and monto_ingresado <= saldo_actual:
                btn_retirar.disabled = False  # Habilitar
            else:
                btn_retirar.disabled = True  # Deshabilitar

        except ValueError:
            # Si el valor no es un número válido (ej. está vacío o es
solo un '.'), deshabilitar
            btn_retirar.disabled = True
```

```
        page.update() # Reflejar el cambio en la interfaz

    def toggle_compra_button(e=None):
        """Habilita o deshabilita el botón de compra basado en el límite
disponible y el monto ingresado."""
        if estado["tarjeta"] is None:
            # Si no hay tarjeta, siempre debe estar deshabilitado
            btn_comprar.disabled = True
            return

        try:
            monto_ingresado = float(txt_monto_tarjeta.value)

            limite_disponible = estado["tarjeta"].get_limite()

            # La lógica clave:
            if monto_ingresado > 0 and monto_ingresado <=
limite_disponible:
                btn_comprar.disabled = False  # Habilitar
            else:
                btn_comprar.disabled = True  # Deshabilitar

        except ValueError:
            # Si el valor no es un número válido (ej. está vacío),
deshabilitar
            btn_comprar.disabled = True

        page.update()

    def exportar_datos_a_pdf(e):

        if estado["cliente"] is None:
            mostrar_notificacion("❌ Error: Primero registre y seleccione
un cliente.", ft.Colors.RED_700)
            return
        cliente_actual = estado["cliente"]
        cuenta_actual = estado["cuenta"]
        tarjeta_actual = estado["tarjeta"]

        mostrar_notificacion(f"💾 Generando PDF para
{cliente_actual.get_nombre()}...", ft.Colors.AMBER_700, 5000)

        # Inicia un hilo para la exportación a PDF
        threading.Thread(
            # El hilo ejecuta la función de manejo de resultados,
pasándole la función FPDF
            target=lambda: _manejar_resultado_pdf(
                _generar_y_guardar_pdf(cliente_actual, cuenta_actual,
tarjeta_actual)
            )
        ).start()
    def mostrar_notificacion(texto, color=ft.Colors.GREEN_700,
duracion_ms=3000):
```

```
        page.snack_bar = ft.SnackBar(
            ft.Text(texto),
            bgcolor=color,
            duration=duracion_ms
        )
        page.snack_bar.open = True
        page.update()
    def _manejar_resultado_pdf(resultado):


        exito, mensaje = resultado
        if exito:
            mostrar_notificacion(f"✅ Exportación exitosa. Archivo
'{mensaje}' creado.", ft.Colors.GREEN_700, 7000)
        else:
            mostrar_notificacion(f"❌ Error al guardar PDF: {mensaje}",
ft.Colors.RED_700, 7000)

    def _generar_y_guardar_pdf(cliente_actual, cuenta_actual,
tarjeta_actual):

        # --- Validación inicial ---
        if cliente_actual is None:
             return False, "No hay cliente seleccionado para exportar."

        from src.cuenta_ahorro import CuentaAhorro

        # --- Generación del PDF ---
        return generar_pdf_reporte(
            cliente_actual,
            cuenta_actual,
            tarjeta_actual,
            CuentaAhorro,
            ft.Colors
        )

    # --- Interfaz Cliente ---
    txt_nombre = ft.TextField(
        label="Nombre",
        input_filter=ft.InputFilter(r"[a-zA-Z\s]"),
        width=300
        )

    txt_apellido = ft.TextField(
        input_filter=ft.InputFilter(r"[a-zA-Z\s]"),
        label="Apellido",
        width=300
        )

    txt_dni = ft.TextField(label="DNI",
keyboard_type=ft.KeyboardType.NUMBER , width=300,)
    btn_registrar_cliente = ft.ElevatedButton(text="Registrar Cliente",
icon=ft.Icons.PERSON_ADD)
```

```
# --- Interfaz Cuenta ---
txt_nro_cuenta = ft.TextField(
    label="Número de Cuenta",
    width=300,
    input_filter=ft.InputFilter(r"[0-9]"),
    )

txt_saldo_inicial = ft.TextField(
    label="Saldo Inicial",
    width=300, value="0.0",
    keyboard_type=ft.KeyboardType.NUMBER
    )

txt_interes = ft.TextField(
    label="Tasa de Interés (%)",
    width=300, value="1.0",
    visible=False,
    keyboard_type=ft.KeyboardType.NUMBER,
    input_filter=ft.InputFilter(r"[0-9\.]")
    )

dropdown_tipo_cuenta = ft.Dropdown(
    width=300, label="Tipo de Cuenta",
    options=[ft.dropdown.Option("Cuenta Corriente"),
             ft.dropdown.Option("Ahorro")],
    value="Cuenta Corriente"
    )

btn_crear_cuenta = ft.ElevatedButton(
    text="Crear Cuenta",
    icon=ft.Icons.ACCOUNT_BALANCE
    )

# Operaciones Cuenta
lbl_saldo_cuenta = ft.Text("Saldo Actual: $0.00",
    size=16,
    weight=ft.FontWeight.BOLD
    )

txt_monto_cuenta = ft.TextField(
    label="Monto (Depósito/Retiro)",
    width=300,
    keyboard_type=ft.KeyboardType.NUMBER,

    on_change=toggle_retiro_button,
    )

btn_depositar = ft.ElevatedButton(
    text="Depositar",
    icon=ft.Icons.ARROW_UPWARD,
    disabled=True
    )
```

```
    btn_retirar = ft.ElevatedButton(
        text="Retirar",
        icon=ft.Icons.ARROW_DOWNWARD,
        disabled=True
        )

    btn_aplicar_interes = ft.ElevatedButton(
        text="Aplicar Interés",
        icon=ft.Icons.REPLAY_10,
        disabled=True,
        visible=False
        )

    # --- Interfaz Tarjeta ---

    txt_nro_tarjeta = ft.TextField(
        label="Número de Tarjeta",
        width=300,
        keyboard_type=ft.KeyboardType.NUMBER
        )

    txt_limite_tarjeta = ft.TextField(
        label="Límite de Crédito",
        width=300, value="10000.0",
        input_filter=ft.InputFilter(r"[0-9\.]")
        )

    btn_crear_tarjeta = ft.ElevatedButton(text="Emitir Tarjeta",
icon=ft.Icons.CREDIT_CARD)

    # Operaciones Tarjeta

    lbl_limite_tarjeta = ft.Text("Límite Total: $0.00", size=16,
weight=ft.FontWeight.BOLD)

    lbl_saldo_tarjeta = ft.Text("Deuda Actual: $0.00", size=16,
weight=ft.FontWeight.BOLD)

    txt_monto_tarjeta = ft.TextField(
        label="Monto (Compra/Pago)",
        width=300,
        keyboard_type=ft.KeyboardType.NUMBER,
        on_change=toggle_compra_button
        )

    btn_comprar = ft.ElevatedButton(
        text="Realizar Compra",
        icon=ft.Icons.SHOPPING_CART,
        disabled=True
        )

    btn_pagar = ft.ElevatedButton(
        text="Pagar Tarjeta",
        icon=ft.Icons.PAYMENT,
```

```
        disabled=True
        )

    # --- Historial de Transacciones ---
    historial_column = ft.Column(
        [ft.Text("No hay transacciones registradas.", italic=True)],
                scroll=ft.ScrollMode.ADAPTIVE,
                height=150
                )

    historial_container = ft.Container(
        content=ft.Column(
            [ft.Text("Historial de Transacciones", size=18,
weight=ft.FontWeight.BOLD),
                    historial_column],
                    horizontal_alignment=ft.CrossAxisAlignment.START,
                    spacing=10
                    ),
        padding=10, border_radius=10, border=ft.border.all(1,
ft.Colors.GREY_300), width=350, visible=False
    )

    # --- Historial de Movimientos de Tarjeta ---
    historial_tarjeta_column = ft.Column(
        [ft.Text("No hay movimientos registrados.", italic=True)],
                scroll=ft.ScrollMode.ADAPTIVE,
                height=150
                )

    historial_tarjeta_container = ft.Container(
        content=ft.Column(
            [ft.Text("Historial de Tarjeta", size=18,
weight=ft.FontWeight.BOLD),
                    historial_tarjeta_column],
                    horizontal_alignment=ft.CrossAxisAlignment.START,
                    spacing=10
                    ),
        padding=10, border_radius=10, border=ft.border.all(1,
ft.Colors.GREY_300), width=350, visible=False
    )

    # Contenedores de Formulario y Operaciones

    cuenta_form_container = ft.Column(
        [ft.Text("2. Crear Cuenta", size=20, weight=ft.FontWeight.BOLD),
                dropdown_tipo_cuenta,
                txt_nro_cuenta,
                txt_saldo_inicial,
                txt_interes,
                btn_crear_cuenta
                ],
        horizontal_alignment=ft.CrossAxisAlignment.CENTER, spacing=10,
disabled=True,
    )
```

```
    tarjeta_form_container = ft.Column(
        [ft.Text("3. Emitir Tarjeta", size=20,
weight=ft.FontWeight.BOLD),
                txt_nro_tarjeta,
                txt_limite_tarjeta,
                btn_crear_tarjeta
                ],
        horizontal_alignment=ft.CrossAxisAlignment.CENTER, spacing=10,
disabled=True,
    )

    gestion_clientes_container = ft.Container(
        content=ft.Column(
            [
                ft.Text("4. Clientes Registrados", size=20,
weight=ft.FontWeight.BOLD),
                btn_nuevo_cliente,
                ft.Divider(),
                lista_clientes_column,
            ],
            horizontal_alignment=ft.CrossAxisAlignment.CENTER,
            spacing=10
        ),
        padding=20,
        border_radius=10,
        border=ft.border.all(1, ft.Colors.PURPLE_200),
        width=350,
        height=380
    )
    operaciones_cuenta_contenido = ft.Container(
        content=ft.Column(
            [lbl_saldo_cuenta,
             txt_monto_cuenta,
             ft.Row([btn_depositar, btn_retirar],
                    alignment=ft.MainAxisAlignment.CENTER),
                    btn_aplicar_interes],
                    horizontal_alignment=ft.CrossAxisAlignment.CENTER,
                    spacing=10
                    ),
                    visible=False, width=350, padding=10
    )
    operaciones_tarjeta_contenido = ft.Container(
        content=ft.Column(
            [
                lbl_limite_tarjeta,
                lbl_saldo_tarjeta,
                txt_monto_tarjeta,
                ft.Row([btn_comprar, btn_pagar],
alignment=ft.MainAxisAlignment.CENTER),
                historial_tarjeta_container
            ],
            horizontal_alignment=ft.CrossAxisAlignment.CENTER,
            spacing=10
```

```
        ),
        visible=False, width=350, padding=10
    )

    contenedor_operaciones_cuenta_final = ft.Container(
        content=ft.Column(
            [
                ft.Text("Operaciones Cuenta", size=18,
weight=ft.FontWeight.BOLD),

                operaciones_cuenta_contenido,
                historial_container
            ],
            horizontal_alignment=ft.CrossAxisAlignment.CENTER
        ),
        padding=20, border_radius=10, border=ft.border.all(1,
ft.Colors.INDIGO_200), width=370, visible=False,
    )

    contenedor_operaciones_tarjeta_final = ft.Container(
        content=ft.Column(
            [
                ft.Text("Operaciones Tarjeta", size=18,
weight=ft.FontWeight.BOLD),

                operaciones_tarjeta_contenido
            ],
            horizontal_alignment=ft.CrossAxisAlignment.CENTER
        ),
        padding=20, border_radius=10, border=ft.border.all(1,
ft.Colors.ORANGE_200), width=370, visible=False,
    )

    # ----------------- Lógica del Negocio de la Aplicación -------------
----
    def actualizar_tarjeta_clientes():
        lista_clientes_column.controls.clear()

        if not clientes_registrados:
            lista_clientes_column.controls.append(ft.Text("No hay
clientes registrados.", italic=True))
        else:
            for cliente in clientes_registrados:
                cliente_row = ft.Row(
                    [
                        ft.Text(f"{cliente.get_nombre()}
{cliente.get_apellido()} (DNI: {cliente._Cliente__dni})", size=14),

                        ft.ElevatedButton(
                            "Seleccionar",
                            icon=ft.Icons.CHECK_CIRCLE_OUTLINE,

                            on_click=lambda e, c=cliente:
seleccionar_cliente(c),
```

```
                        ),
                    ],
                    alignment=ft.MainAxisAlignment.SPACE_BETWEEN
                )
                lista_clientes_column.controls.append(cliente_row)
        page.update()

    def seleccionar_cliente(cliente_seleccionado):

        # Restablece la cuenta y tarjeta antiguas
        estado["cuenta"] = None
        estado["tarjeta"] = None
        estado["cliente"] = cliente_seleccionado # 1. Establece el
cliente seleccionado

        # Restablece visualmente el formulario de registro
        txt_nombre.value = cliente_seleccionado.get_nombre()
        txt_apellido.value = cliente_seleccionado.get_apellido()

        txt_dni.value = cliente_seleccionado._Cliente__dni

        # Habilita las secciones de productos (Cuenta/Tarjeta) y
deshabilita el registro
        btn_registrar_cliente.disabled = True
        cuenta_form_container.disabled = False
        tarjeta_form_container.disabled = False

        # Oculta las operaciones hasta que se creen productos para este
cliente
        operaciones_cuenta_contenido.visible = False
        operaciones_tarjeta_contenido.visible = False
        historial_container.visible = False

        mostrar_notificacion(f"👤 Cliente
'{cliente_seleccionado.get_nombre()}' seleccionado. Cree su Cuenta y
Tarjeta.", ft.Colors.AMBER_700)
        page.update()

    # Resetea el formulario para crear un nuevo cliente
    def reset_formulario_cliente(e):

        estado["cliente"] = None
        estado["cuenta"] = None
        estado["tarjeta"] = None

        txt_nombre.value = ""; txt_apellido.value = ""; txt_dni.value =
""
        btn_registrar_cliente.disabled = False # Habilita el registro

        # Deshabilita productos/operaciones
        cuenta_form_container.disabled = True
        tarjeta_form_container.disabled = True
        operaciones_cuenta_contenido.visible = False
```

```
        operaciones_tarjeta_contenido.visible = False

        mostrar_notificacion("Formulario listo. Ingrese los datos del
nuevo cliente.", ft.Colors.CYAN_700)
        page.update()

    btn_nuevo_cliente.on_click = reset_formulario_cliente
    def registrar_cliente(e):
        nombre, apellido, dni = txt_nombre.value.strip(),
txt_apellido.value.strip(), txt_dni.value.strip()
        try:
            nuevo_cliente = Cliente(nombre, apellido, dni)
            clientes_registrados.append(nuevo_cliente)
            estado["cliente"] = nuevo_cliente
            mostrar_notificacion(f"Cliente registrado:
{nuevo_cliente.__str__()}", ft.Colors.BLUE_700)
            btn_registrar_cliente.disabled = True
            cuenta_form_container.disabled = False
            tarjeta_form_container.disabled = False

            #  ACTUALIZAR LA TARJETA DE CLIENTES
            actualizar_tarjeta_clientes()
            page.update()

        except ValueError as ex:
            mostrar_notificacion(f"Error Cliente: {ex}",
ft.Colors.RED_700)

    def crear_cuenta(e):
        nro_cuenta, tipo_cuenta = txt_nro_cuenta.value.strip(),
dropdown_tipo_cuenta.value
        if not estado["cliente"]:
            mostrar_notificacion("✗ Primero debe registrar un cliente.",
ft.Colors.RED_700)
            return

        try:
            saldo_inicial = float(txt_saldo_inicial.value)
            if tipo_cuenta == "Ahorro":
                interes = float(txt_interes.value)
                nueva_cuenta = CuentaAhorro(nro_cuenta,
estado["cliente"], saldo_inicial, interes)
                btn_aplicar_interes.visible = True

            else:
                nueva_cuenta = Cuenta(nro_cuenta, estado["cliente"],
saldo_inicial)
                btn_aplicar_interes.visible = False

            estado["cuenta"] = nueva_cuenta
            mostrar_notificacion(f"✓ Cuenta creada con éxito.",
ft.Colors.GREEN_700)

            cuenta_form_container.disabled = True
```

```
            operaciones_cuenta_contenido.visible = True
            historial_container.visible = True
            btn_depositar.disabled = btn_retirar.disabled = False
            actualizar_saldo_cuenta()
        except ValueError as ex:
            mostrar_notificacion(f"⚠ Error en cuenta: {ex}",
ft.Colors.RED_700)
        except TypeError as ex:
            mostrar_notificacion(f"⚠ Error en tipo de dato: {ex}",
ft.Colors.RED_700)

    def crear_tarjeta(e):
        nro_tarjeta = txt_nro_tarjeta.value
        if estado["cliente"] is None: mostrar_notificacion(" Error:
Primero debe registrar un cliente.", ft.Colors.RED_700); return
        try:
            limite = float(txt_limite_tarjeta.value)
            nueva_tarjeta = Tarjeta(numero=nro_tarjeta,
cliente=estado["cliente"], limite=limite)
            estado["tarjeta"] = nueva_tarjeta
            historial_tarjeta_container.visible = True

            mostrar_notificacion(f" Tarjeta emitida: Nro
{nueva_tarjeta.get_numero()}", ft.Colors.INDIGO_700)
            tarjeta_form_container.disabled = True
            operaciones_tarjeta_contenido.visible = True
            btn_comprar.disabled = btn_pagar.disabled = False
            actualizar_saldo_tarjeta()

        except ValueError as ex: mostrar_notificacion(f" Error Tarjeta:
{ex}", ft.Colors.RED_700)

    def depositar(e):
        try:
            monto = float(txt_monto_cuenta.value)
            estado["cuenta"].depositar(monto)
            mostrar_notificacion(f"💰 Depósito de ${monto:.2f}
realizado.", ft.Colors.GREEN_700)
            actualizar_saldo_cuenta()
        except (ValueError, TypeError) as ex:
            mostrar_notificacion(f"❌ Error depósito: {ex}",
ft.Colors.RED_700)


    def retirar(e):
        try:
            monto = float(txt_monto_cuenta.value)
            estado["cuenta"].retirar(monto)
            mostrar_notificacion(f"💸 Retiro de ${monto:.2f} realizado.",
ft.Colors.AMBER_700)
            actualizar_saldo_cuenta()
        except (SaldoInsuficienteError, ValueError, TypeError) as ex:
```

```
            mostrar_notificacion(f"⚠ Error retiro: {ex}",
ft.Colors.RED_700)


    def aplicar_interes(e):
        if isinstance(estado["cuenta"], CuentaAhorro):
            saldo_anterior = estado["cuenta"].get_saldo()
            estado["cuenta"].aplicar_interes()
            interes_aplicado = estado["cuenta"].get_saldo() -
saldo_anterior
            mostrar_notificacion(f" Interés aplicado
(${interes_aplicado:.2f}).", ft.Colors.YELLOW_700)
            actualizar_saldo_cuenta()
        else: mostrar_notificacion(" Esta cuenta no admite la aplicación
de intereses.", ft.Colors.RED_700)

    def realizar_compra(e):
        try:
            monto = float(txt_monto_tarjeta.value)
            estado["tarjeta"].realizar_compra(monto)
            mostrar_notificacion(f"Compra de ${monto:.2f} realizada.",
ft.Colors.ORANGE_700)
            actualizar_saldo_tarjeta()
            actualizar_historial_tarjeta()
        except LimiteExcedidoError as ex:
            mostrar_notificacion(f" Error Compra: {ex}",
ft.Colors.RED_700)
        except ValueError as ex:
            mostrar_notificacion(f" Error Compra: {ex}",
ft.Colors.RED_700)

    def pagar_tarjeta(e):
        try:
            monto = float(txt_monto_tarjeta.value)
            estado["tarjeta"].pagar_tarjeta(monto)
            mostrar_notificacion(f" Pago de ${monto:.2f} realizado.",
ft.Colors.GREEN_700)
            actualizar_saldo_tarjeta()
            actualizar_historial_tarjeta()
        except ValueError as ex: mostrar_notificacion(f" Error Pago:
{ex}", ft.Colors.RED_700)

    # Asignar eventos
    dropdown_tipo_cuenta.on_change = lambda e:
txt_interes.set_value(txt_interes.value) if e.control.value == "Ahorro"
and txt_interes.visible == True else setattr(txt_interes, 'visible',
e.control.value == "Ahorro")
    btn_depositar.on_click = depositar
    btn_retirar.on_click = retirar
    btn_aplicar_interes.on_click = aplicar_interes
    btn_comprar.on_click = realizar_compra
    btn_pagar.on_click = pagar_tarjeta
    btn_exportar_pdf.on_click = exportar_datos_a_pdf
    def update_operaciones_row_visibility(e=None):
```

```
        operaciones_row = page.controls[-1].content
        if estado["cuenta"]:
            contenedor_operaciones_cuenta_final.visible = True
        if estado["tarjeta"]:
            contenedor_operaciones_tarjeta_final.visible = True
        page.update()

    btn_crear_cuenta.on_click = lambda e: (crear_cuenta(e),
update_operaciones_row_visibility(e))
    btn_crear_tarjeta.on_click = lambda e: (crear_tarjeta(e),
update_operaciones_row_visibility(e))
    btn_registrar_cliente.on_click = registrar_cliente

    # ----------------- Estructura de la Interfaz -----------------

    cliente_container = ft.Container(
        content=ft.Column([ft.Text("1. Datos del Cliente", size=20,
weight=ft.FontWeight.BOLD), txt_nombre, txt_apellido, txt_dni,
btn_registrar_cliente],
horizontal_alignment=ft.CrossAxisAlignment.CENTER, spacing=10),
        padding=20, border_radius=10, border=ft.border.all(1,
ft.Colors.BLUE_200), width=350,
    )

    page.add(
        ft.Row(
            [

                # COLUMNA 1: Datos del Cliente (Formulario de Registro)
                ft.Column([cliente_container], spacing=30,
alignment=ft.MainAxisAlignment.START),
                ft.VerticalDivider(),

                # COLUMNA 2: Creación de Productos (Cuenta y Tarjeta)
                ft.Column([cuenta_form_container,
tarjeta_form_container], spacing=30,
alignment=ft.MainAxisAlignment.START),
                ft.VerticalDivider(),

                # COLUMNA 3: Gestión de Clientes (La nueva tarjeta de
selección)
                gestion_clientes_container

            ],
            vertical_alignment=ft.CrossAxisAlignment.START, spacing=30
        ),
        ft.Container(
            content=ft.Row(
                [
                    contenedor_operaciones_cuenta_final,
                    ft.VerticalDivider(visible=False),
                    contenedor_operaciones_tarjeta_final,
                ],
                spacing=30
```

```
            ),
            margin=ft.margin.only(top=30)
        ),
            exportar_container
    )

if __name__ == "__main__":
    ft.app(target=main)


================================================
FILE: src/cliente.py
================================================
class Cliente:
    def __init__(self, nombre, apellido, dni):
        # Validaciones básicas
        if not nombre or not apellido or not dni:
            raise ValueError("Todos los campos del cliente son
obligatorios.")

        # Validación: solo letras en nombre y apellido
        if not nombre.replace(" ", "").isalpha() or not
apellido.replace(" ", "").isalpha():
            raise ValueError(
                "El nombre y apellido deben contener solo letras.")

        # Validación: solo números en DNI
        if not dni.isdigit():
            raise ValueError("El DNI debe contener solo números.")

        self.__nombre = nombre.strip().title()
        self.__apellido = apellido.strip().title()
        self.__dni = dni.strip()

    # Métodos getters
    def get_nombre(self):
        return self.__nombre

    def get_apellido(self):
        return self.__apellido

    def get_dni(self):
        return self.__dni

    def mostrar_datos(self):
        return f"{self.__nombre} {self.__apellido} (DNI: {self.__dni})"

    def __str__(self):
        return self.mostrar_datos()



================================================
FILE: src/cuenta.py
```

```
==============================================
from src.transaccion import Transaccion


class SaldoInsuficienteError(Exception):
    """Excepción personalizada para saldo insuficiente."""
    pass


class Cuenta:
    def __init__(self, nro_cuenta: str, cliente, saldo: float = 0.0):
        if not nro_cuenta or not isinstance(nro_cuenta, str):
            raise ValueError(
                "El número de cuenta debe ser una cadena no vacía.")
        if saldo < 0:
            raise ValueError("El saldo inicial no puede ser negativo.")
        if cliente is None:
            raise ValueError("Debe asignarse un cliente válido a la
cuenta.")

        self.__nro_cuenta = nro_cuenta
        self.__cliente = cliente
        self.__saldo = saldo
        self.__transacciones = []

    # Métodos getters
    def get_nro_cuenta(self):
        return self.__nro_cuenta

    def get_saldo(self):
        return self.__saldo

    def get_cliente(self):
        return self.__cliente

    def get_transacciones(self):
        return self.__transacciones

    # Operaciones
    def depositar(self, monto: float):
        """Agrega dinero a la cuenta."""
        if not isinstance(monto, (int, float)):
            raise TypeError("El monto del depósito debe ser un número.")
        if monto <= 0:
            raise ValueError("El monto del depósito debe ser mayor a
cero.")
        self.__saldo += monto
        self.__transacciones.append(Transaccion("deposito", monto))

    def retirar(self, monto: float):
        """Retira dinero de la cuenta si hay saldo suficiente."""
        if not isinstance(monto, (int, float)):
            raise TypeError("El monto del retiro debe ser un número.")
        if monto <= 0:
```

```
            raise ValueError("El monto del retiro debe ser mayor a
cero.")
        if monto > self.__saldo:
            raise SaldoInsuficienteError(
                "Saldo insuficiente para realizar el retiro.")
        self.__saldo -= monto
        self.__transacciones.append(Transaccion("retiro", monto))

    def mostrar_transacciones(self):
        """Devuelve una lista legible de las transacciones realizadas."""
        if not self.__transacciones:
            return ["No hay transacciones registradas."]
        return [str(t) for t in self.__transacciones]



================================================
FILE: src/cuenta_ahorro.py
================================================
from src.cuenta import Cuenta


class CuentaAhorro(Cuenta):
    def __init__(self, nro_cuenta: str, cliente, saldo: float = 0.0,
interes: float = 1.0):
        super().__init__(nro_cuenta, cliente, saldo)
        if interes < 0:
            raise ValueError("La tasa de interés no puede ser negativa.")
        self.__interes = interes

    def get_interes(self):
        return self.__interes

    def aplicar_interes(self):
        """Aplica el interés al saldo actual."""
        saldo_actual = self.get_saldo()
        nuevo_saldo = saldo_actual + (saldo_actual * self.__interes /
100)
        # Actualizamos el saldo internamente
        self._Cuenta__saldo = nuevo_saldo  # usamos el atributo protegido
de la clase base
        return nuevo_saldo



================================================
FILE: src/exportar_datos_a_pdf.py
================================================
# Contenido COMPLETO y CORREGIDO de src/exportar_datos_a_pdf.py

from fpdf import FPDF
from datetime import datetime
from typing import Tuple, Any
```

```
def generar_pdf_reporte(cliente_actual, cuenta_actual, tarjeta_actual,
EsCuentaAhorro, ft_colors) -> Tuple[bool, str]:


    if cliente_actual is None:
        return False, "No hay cliente seleccionado para exportar."

    try:
        # ------------------------------------------------------
        # 1. INICIALIZAR PDF
        # ------------------------------------------------------

        pdf = FPDF(orientation='P', unit='mm', format='A4')
        pdf.set_auto_page_break(auto=True, margin=15)
        pdf.add_page()

        pdf.image("../sistema_bancario/img/logo_DevOps.jpg", x=160, y=2,
w=30)
        pdf.image("../sistema_bancario/img/its.png", x=20, y=8, w=30)
        pdf.ln(25)

        # ------------------------------------------------------
        # 2. DATOS DEL CLIENTE
        # ------------------------------------------------------

        pdf.set_fill_color(220, 220, 220)
        pdf.set_font('Arial', 'B', 12)
        pdf.cell(0, 8, 'DATOS DEL CLIENTE', 1, 1, 'L', True)

        pdf.set_font('Arial', '', 10)
        pdf.cell(40, 6, 'Nombre:', 0, 0)
        pdf.cell(0, 6, f'{cliente_actual.get_nombre()}
{cliente_actual.get_apellido()}', 0, 1)

        pdf.cell(40, 6, 'DNI:', 0, 0)
        pdf.cell(0, 6, f'{cliente_actual._Cliente__dni}', 0, 1)
        pdf.ln(2)

        # ------------------------------------------------------
        # 3. DATOS DE LA CUENTA BANCARIA
        # ------------------------------------------------------

        if cuenta_actual:
            pdf.set_fill_color(220, 255, 220)
            pdf.set_font('Arial', 'B', 12)
            pdf.cell(0, 8, 'CUENTA BANCARIA', 1, 1, 'L', True)

            pdf.set_font('Arial', '', 10)
            pdf.cell(40, 6, 'Nro Cuenta:', 0, 0)
            pdf.cell(0, 6, f'{cuenta_actual._Cuenta__nro_cuenta}', 0, 1)

            pdf.cell(40, 6, 'Tipo:', 0, 0)
            pdf.set_font('Arial', 'B', 10)
```

```
            pdf.cell(0, 6, f'{"Ahorro" if isinstance(cuenta_actual,
EsCuentaAhorro) else "Base"}', 0, 1)

            pdf.cell(40, 6, 'SALDO ACTUAL:', 0, 0)
            pdf.set_font('Arial', 'B', 10)
            pdf.cell(0, 6, f'${cuenta_actual.get_saldo():.2f}', 0, 1)
            pdf.ln(2)

            # 3.1. Historial de Transacciones
            pdf.set_font('Arial', 'U', 11)
            pdf.cell(0, 6, 'Historial de Transacciones:', 0, 1)

            pdf.set_font('Arial', 'B', 8)
            pdf.cell(40, 5, 'FECHA/HORA', 1, 0, 'C')
            pdf.cell(30, 5, 'TIPO', 1, 0, 'C')
            pdf.cell(20, 5, 'MONTO', 1, 1, 'C')

            pdf.set_font('Arial', '', 8)
            transacciones = cuenta_actual.get_transacciones()
            if transacciones:
                for t in transacciones:
                    fecha = t.get_fecha().strftime('%Y-%m-%d %H:%M:%S')
                    monto_str = f'${t.get_monto():.2f}'
                    pdf.cell(40, 5, fecha, 1, 0)
                    pdf.cell(30, 5, t.get_tipo().upper(), 1, 0)
                    pdf.cell(20, 5, monto_str, 1, 1, 'R')
            else:
                pdf.cell(90, 5, 'No se registraron transacciones.', 1, 1,
'C')
            pdf.ln(5)

        # ----------------------------------------------------
        # 4. DATOS DE LA TARJETA DE CRÉDITO
        # ----------------------------------------------------

        if tarjeta_actual:
            pdf.set_fill_color(255, 220, 220)
            pdf.set_font('Arial', 'B', 12)
            pdf.cell(0, 8, 'TARJETA DE CRÉDITO', 1, 1, 'L', True)

            pdf.set_font('Arial', '', 10)
            pdf.cell(40, 6, 'Nro Tarjeta:', 0, 0)
            pdf.cell(0, 6, f'{tarjeta_actual._Tarjeta__numero}', 0, 1)

            pdf.cell(40, 6, 'Límite Total:', 0, 0)
            pdf.cell(0, 6, f'${tarjeta_actual.get_limite():.2f}', 0, 1)

            pdf.cell(40, 6, 'DEUDA ACTUAL:', 0, 0)
            pdf.set_font('Arial', 'B', 10)
            pdf.cell(0, 6, f'${tarjeta_actual.get_saldo_actual():.2f}',
0, 1)
            pdf.ln(2)

            # 4.1. Historial de Movimientos de la Tarjeta
```

```
            pdf.set_font('Arial', 'U', 11)
            pdf.cell(0, 6, 'Historial de Movimientos:', 0, 1)

            pdf.set_font('Arial', 'B', 8)
            pdf.cell(40, 5, 'FECHA/HORA', 1, 0, 'C')
            pdf.cell(30, 5, 'TIPO', 1, 0, 'C')
            pdf.cell(20, 5, 'MONTO', 1, 1, 'C')

            pdf.set_font('Arial', '', 8)
            movimientos = tarjeta_actual._Tarjeta__movimientos
            if movimientos:
                for fecha, monto, tipo in movimientos:
                    fecha_str = fecha.strftime('%Y-%m-%d %H:%M:%S')
                    monto_str = f'${monto:.2f}'
                    pdf.cell(40, 5, fecha_str, 1, 0)
                    pdf.cell(30, 5, tipo.upper(), 1, 0)
                    pdf.cell(20, 5, monto_str, 1, 1, 'R')
            else:
                pdf.cell(90, 5, 'No se registraron movimientos.', 1, 1,
'C')
            pdf.ln(5)

        # ------------------------------------------------
        # 5. SALIDA DEL ARCHIVO
        # ------------------------------------------------

        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        nombre_cliente =
f"{cliente_actual.get_nombre()}_{cliente_actual.get_apellido()}".replace(
" ", "_")
        nombre_archivo = f"Reporte_{nombre_cliente}_{timestamp}.pdf"

        pdf.output(nombre_archivo)


        return True, nombre_archivo

    except Exception as exc:


        return False, str(exc)


================================================
FILE: src/tarjeta.py
================================================
from datetime import datetime


class LimiteExcedidoError(Exception):
    """Excepción personalizada para límite de crédito excedido."""
    pass
```

```
class Tarjeta:
    def __init__(self, numero: str, cliente, limite: float = 10000.0):
        if not numero:
            raise ValueError("El número de tarjeta no puede estar
vacío.")
        if limite <= 0:
            raise ValueError("El límite debe ser mayor a cero.")

        self.__numero = numero
        self.__cliente = cliente
        self.__limite = limite
        self.__saldo_actual = 0.0
        self.__movimientos = []

    # Métodos getters
    def get_numero(self):
        return self.__numero

    def get_cliente(self):
        return self.__cliente   # ✓ agregado

    def get_limite(self):
        return self.__limite

    def get_saldo_actual(self):
        return self.__saldo_actual

    def get_movimientos(self):
        return self.__movimientos

    # Operaciones
    def realizar_compra(self, monto: float):
        if monto <= 0:
            raise ValueError("El monto de la compra debe ser mayor a
cero.")
        if self.__saldo_actual + monto > self.__limite:
            raise LimiteExcedidoError("Se excede el límite de crédito.")
        self.__saldo_actual += monto
        self.__movimientos.append((datetime.now(), monto, "Compra"))

    def pagar_tarjeta(self, monto: float):
        if monto <= 0:
            raise ValueError("El monto del pago debe ser mayor a cero.")
        self.__saldo_actual -= monto
        self.__movimientos.append((datetime.now(), monto, "Pago"))

    def __str__(self):
        return f"Tarjeta {self.__numero} - Cliente:
{self.__cliente.get_nombre()} {self.__cliente.get_apellido()}"



==============================================
```

```
FILE: src/transaccion.py
================================================
from datetime import datetime


class Transaccion:
    def __init__(self, tipo: str, monto: float):
        if tipo not in ["deposito", "retiro"]:
            raise ValueError("Tipo de transacción inválido")
        if not isinstance(monto, (int, float)):
            raise TypeError("El monto debe ser un número.")
        if monto <= 0:
            raise ValueError("El monto debe ser mayor a cero.")

        self.__tipo = tipo
        self.__monto = monto
        self.__fecha = datetime.now()

    # Getters
    def get_tipo(self):
        return self.__tipo

    def get_monto(self):
        return self.__monto

    def get_fecha(self):
        return self.__fecha

    def __str__(self):
        return f"{self.__fecha.strftime('%Y-%m-%d %H:%M:%S')} -
{self.__tipo}: ${self.__monto}"



================================================
FILE: tests/__init__.py
================================================
[Empty file]


================================================
FILE: tests/test_cliente.py
================================================
import pytest
from src.cliente import Cliente


def test_crear_cliente_valido():
    cliente = Cliente("Ana", "López", "12345678")
    assert cliente.get_nombre() == "Ana"
    assert cliente.get_apellido() == "López"
    assert cliente.get_dni() == "12345678"
```

```
def test_cliente_nombre_vacio():
    with pytest.raises(ValueError, match="Todos los campos del cliente
son obligatorios."):
        Cliente("", "López", "12345678")


def test_cliente_dni_vacio():
    with pytest.raises(ValueError, match="Todos los campos del cliente
son obligatorios."):
        Cliente("Ana", "López", "")


def test_cliente_nombre_con_numeros():
    # No debe permitir números en el nombre
    with pytest.raises(ValueError, match="solo letras"):
        Cliente("Ana123", "López", "12345678")


def test_cliente_apellido_con_simbolos():
    # No debe permitir símbolos en el apellido
    with pytest.raises(ValueError, match="solo letras"):
        Cliente("Ana", "López@", "12345678")


def test_cliente_dni_con_letras():
    # No debe permitir letras en el DNI
    with pytest.raises(ValueError, match="solo números"):
        Cliente("Ana", "López", "12A45B78")



================================================
FILE: tests/test_cuenta.py
================================================
import pytest
from src.cliente import Cliente
from src.cuenta import Cuenta, SaldoInsuficienteError


def test_crear_cuenta_valida():
    cliente = Cliente("Juan", "Perez", "11111111")
    cuenta = Cuenta("001", cliente, saldo=500.0)
    assert cuenta.get_saldo() == 500.0
    assert cuenta.get_cliente().get_nombre() == "Juan"


def test_no_permitir_saldo_negativo_inicial():
    cliente = Cliente("Juan", "Perez", "11111111")
    with pytest.raises(ValueError):
        Cuenta("002", cliente, saldo=-100)


def test_deposito_valido():
    cliente = Cliente("Ana", "Lopez", "22222222")
```

```
    cuenta = Cuenta("002", cliente)
    cuenta.depositar(100)
    assert cuenta.get_saldo() == 100.0


def test_retiro_valido():
    cliente = Cliente("Carlos", "Diaz", "33333333")
    cuenta = Cuenta("003", cliente, saldo=200)
    cuenta.retirar(50)
    assert cuenta.get_saldo() == 150.0


def test_saldo_insuficiente():
    cliente = Cliente("Pedro", "Gomez", "44444444")
    cuenta = Cuenta("004", cliente, saldo=10)
    with pytest.raises(SaldoInsuficienteError):
        cuenta.retirar(50)


def test_monto_no_numerico_en_deposito():
    cliente = Cliente("Laura", "Mendez", "55555555")
    cuenta = Cuenta("005", cliente)
    with pytest.raises(TypeError):
        cuenta.depositar("cien")


def test_retiro_monto_no_numerico():
    cliente = Cliente("Mario", "Suarez", "66666666")
    cuenta = Cuenta("006", cliente, saldo=100)
    with pytest.raises(TypeError):
        cuenta.retirar("veinte")


def test_retiro_monto_negativo():
    cliente = Cliente("Sofia", "Ramos", "77777777")
    cuenta = Cuenta("007", cliente, saldo=100)
    with pytest.raises(ValueError):
        cuenta.retirar(-50)



================================================
FILE: tests/test_cuenta_ahorro.py
================================================
from src.cuenta_ahorro import CuentaAhorro
from src.cliente import Cliente


def test_crear_cuenta_ahorro():
    cliente = Cliente("Lucía", "Martinez", "44556677")
    cuenta_ahorro = CuentaAhorro("005", cliente, saldo=1000, interes=2.0)
    assert cuenta_ahorro.get_interes() == 2.0
    assert cuenta_ahorro.get_saldo() == 1000
```

```
def test_aplicar_interes():
    cliente = Cliente("Mario", "Diaz", "99887766")
    cuenta_ahorro = CuentaAhorro("006", cliente, saldo=1000, interes=5.0)
    nuevo_saldo = cuenta_ahorro.aplicar_interes()
    assert round(nuevo_saldo, 2) == 1050.00



================================================
FILE: tests/test_tarjeta.py
================================================
import pytest
from src.cliente import Cliente
from src.tarjeta import Tarjeta, LimiteExcedidoError


def test_crear_tarjeta_valida():
    cliente = Cliente("Juan", "Perez", "12345678")
    tarjeta = Tarjeta("1111-2222-3333-4444", cliente, limite=5000)
    assert tarjeta.get_numero() == "1111-2222-3333-4444"
    assert tarjeta.get_limite() == 5000
    assert tarjeta.get_saldo_actual() == 0


def test_realizar_compra_valida():
    cliente = Cliente("Ana", "Lopez", "87654321")
    tarjeta = Tarjeta("5555-6666-7777-8888", cliente, limite=2000)
    tarjeta.realizar_compra(500)
    assert tarjeta.get_saldo_actual() == 500


def test_exceder_limite():
    cliente = Cliente("Carlos", "Gomez", "98765432")
    tarjeta = Tarjeta("9999-0000-1111-2222", cliente, limite=1000)
    with pytest.raises(LimiteExcedidoError):
        tarjeta.realizar_compra(2000)


def test_pagar_tarjeta():
    cliente = Cliente("Lucia", "Diaz", "55555555")
    tarjeta = Tarjeta("3333-4444-5555-6666", cliente, limite=3000)
    tarjeta.realizar_compra(1000)
    tarjeta.pagar_tarjeta(500)
    assert tarjeta.get_saldo_actual() == 500


def test_monto_no_numerico():
    cliente = Cliente("Sofia", "Martinez", "99999999")
    tarjeta = Tarjeta("1234-5678-9012-3456", cliente)
    with pytest.raises(TypeError):
        tarjeta.realizar_compra("mil")
```

```
================================================
FILE: tests/test_transaccion.py
================================================
import pytest
from src.transaccion import Transaccion


def test_crear_transaccion_deposito():
    t = Transaccion("deposito", 100.0)
    assert t.get_tipo() == "deposito"
    assert t.get_monto() == 100.0
    assert t.get_fecha() is not None
    assert "deposito" in str(t)


def test_crear_transaccion_retiro():
    t = Transaccion("retiro", 50.0)
    assert t.get_tipo() == "retiro"
    assert t.get_monto() == 50.0
    assert t.get_fecha() is not None
    assert "retiro" in str(t)


def test_tipo_invalido():
    with pytest.raises(ValueError):
        Transaccion("transferencia", 100.0)


def test_monto_invalido():
    with pytest.raises(ValueError):
        Transaccion("deposito", -50.0)


def test_monto_no_numerico():
    with pytest.raises(TypeError):
        Transaccion("deposito", "cien")
```