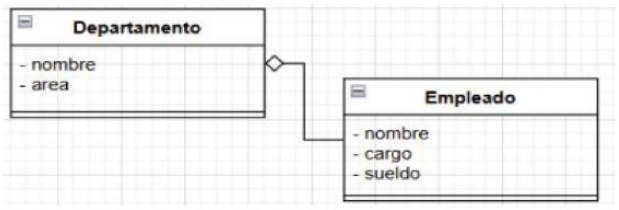
Practica N2

Nombre: Carlos Anegl Morales Cruz

Materia: INF-121

TEMA: COMPOSICION/AGREGACION

Ejercicio 2. Defina las clases:



a) Instanciar 2 departamentos, uno con 5 empleados y el otro sin empleados.
b) Agregar el metodo mostrarEmpleados()
c) Implementar cambioSalario() para todos los empleados de un departamento en específico.
d) Verificar si algun empleado del departamento 1 pertenece al departamento 2.
e) Mover los empleados del departamento 1 al departamento 2. Tras eso mostrar de nuevo los departamentos.

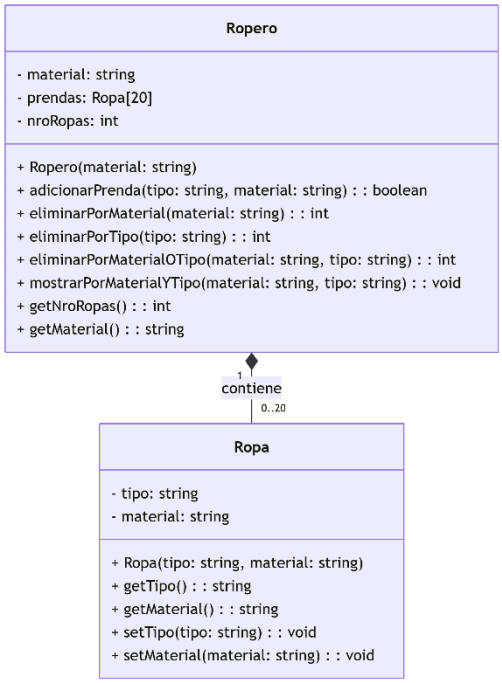
```python
class Departamento:
    def __init__(self, nombre, area):
        self.nombre = nombre
        self.area = area
        self.empleados = []

    def agregar_empleado(self, empleado):
        self.empleados.append(empleado)

    def mostrar_empleados(self):
        print(f"\n--- Empleados del Departamento {self.nombre} ({self.area}) ---")
        if not self.empleados:
            print("No hay empleados en este departamento")
        else:
            for i, empleado in enumerate(self.empleados, 1):
                print(f"{i}. {empleado.nombre} - {empleado.cargo} - ${empleado.sueldo}")

    def cambiar_salario(self, porcentaje):
        for empleado in self.empleados:
            empleado.sueldo *= (1 + porcentaje / 100)
        print(f"Salarios actualizados en {porcentaje}% para el departamento {self.nombre}")

    def mover_empleados_a(self, otro_departamento):
        if not self.empleados:
            print(f"No hay empleados para mover del departamento {self.nombre}")
            return

        print(f"\nMoviendo {len(self.empleados)} empleados de {self.nombre} a {otro_departamento.nombre}")
        otro_departamento.empleados.extend(self.empleados)
        self.empleados.clear()
        print("Movimiento completado exitosamente")


class Empleado:
    def __init__(self, nombre, cargo, sueldo):
        self.nombre = nombre
        self.cargo = cargo
        self.sueldo = sueldo

    def __str__(self):
        return f"{self.nombre} - {self.cargo} - ${self.sueldo}"


departamento1 = Departamento("Ventas", "Comercial")
departamento2 = Departamento("Recursos Humanos", "Administración")

empleados_dep1 = [
    Empleado("Ana García", "Gerente de Ventas", 5000),
```

```python
empleados_dep1 = [
    Empleado("Ana García", "Gerente de Ventas", 5000),
    Empleado("Carlos López", "Ejecutivo de Ventas", 3000),
    Empleado("María Rodríguez", "Asistente de Ventas", 2500),
    Empleado("Pedro Martínez", "Analista Comercial", 3500),
    Empleado("Laura Fernández", "Coordinadora", 2800)
]

for empleado in empleados_dep1:
    departamento1.agregar_empleado(empleado)

print("=== MOSTRAR EMPLEADOS INICIALES ===")
departamento1.mostrar_empleados()
departamento2.mostrar_empleados()

print("\n=== CAMBIO DE SALARIOS ===")
departamento1.cambiar_salario(10)
departamento1.mostrar_empleados()

print("\n=== VERIFICACIÓN DE EMPLEADOS ENTRE DEPARTAMENTOS ===")
empleados_comunes = []
for empleado in departamento1.empleados:
    if empleado in departamento2.empleados:
        empleados_comunes.append(empleado)

if empleados_comunes:
    print(f"Se encontraron {len(empleados_comunes)} empleados en común:")
    for empleado in empleados_comunes:
        print(f"  - {empleado}")
else:
    print("No hay empleados en común entre los departamentos")

print("\n=== MOVIMIENTO DE EMPLEADOS ===")
print("ANTES del movimiento:")
departamento1.mostrar_empleados()
departamento2.mostrar_empleados()

departamento1.mover_empleados_a(departamento2)

print("\nDESPUÉS del movimiento:")
departamento1.mostrar_empleados()
departamento2.mostrar_empleados()
```

```
ANTES del movimiento:

--- Empleados del Departamento Ventas (Comercial) ---
1. Ana García - Gerente de Ventas - $5500.0
2. Carlos López - Ejecutivo de Ventas - $3300.0000000000005
3. María Rodríguez - Asistente de Ventas - $2750.0
4. Pedro Martínez - Analista Comercial - $3850.0000000000005
5. Laura Fernández - Coordinadora - $3080.0000000000005

--- Empleados del Departamento Recursos Humanos (Administración) ---
No hay empleados en este departamento

Moviendo 5 empleados de Ventas a Recursos Humanos
Movimiento completado exitosamente

DESPUÉS del movimiento:

--- Empleados del Departamento Ventas (Comercial) ---
No hay empleados en este departamento

--- Empleados del Departamento Recursos Humanos (Administración) ---
1. Ana García - Gerente de Ventas - $5500.0
2. Carlos López - Ejecutivo de Ventas - $3300.0000000000005
3. María Rodríguez - Asistente de Ventas - $2750.0
4. Pedro Martínez - Analista Comercial - $3850.0000000000005
5. Laura Fernández - Coordinadora - $3080.0000000000005
PS E:\UMSA\third\121\PracticaAuxiliar2>
```

Ejercicio 4. Sea la clase ropero (**material, Ropa[20], nroRopas**) y la clase Ropa(**tipo, material**)

a) Realizar el diagrama de clases
b) Adicionar N prendas al ropero
c) Eliminar las prendas de material x o de tipo y d) mostrar las prendas de materia X y de tipo.




```python
class Ropa:
    def __init__(self, tipo, material):
        self.tipo = tipo
        self.material = material

    def __str__(self):
        return f"{self.tipo} - {self.material}"


class Ropero:
    def __init__(self, material):
        self.material = material
        self.ropa = [None] * 20
        self.nro_ropas = 0

    def adicionar_prenda(self, prenda):
        if self.nro_ropas < 20:
            self.ropa[self.nro_ropas] = prenda
            self.nro_ropas += 1
            print(f"Prenda '{prenda.tipo}' agregada al ropero")
        else:
            print("El ropero está lleno, no se puede agregar más prendas")

    def adicionar_n_prendas(self, prendas):
        for prenda in prendas:
            if self.nro_ropas < 20:
                self.adicionar_prenda(prenda)
            else:
                print("No se pueden agregar más prendas, ropero lleno")
                break

    def eliminar_prendas_material_tipo(self, material=None, tipo=None):
        nuevas_prendas = []
        eliminadas = 0
```

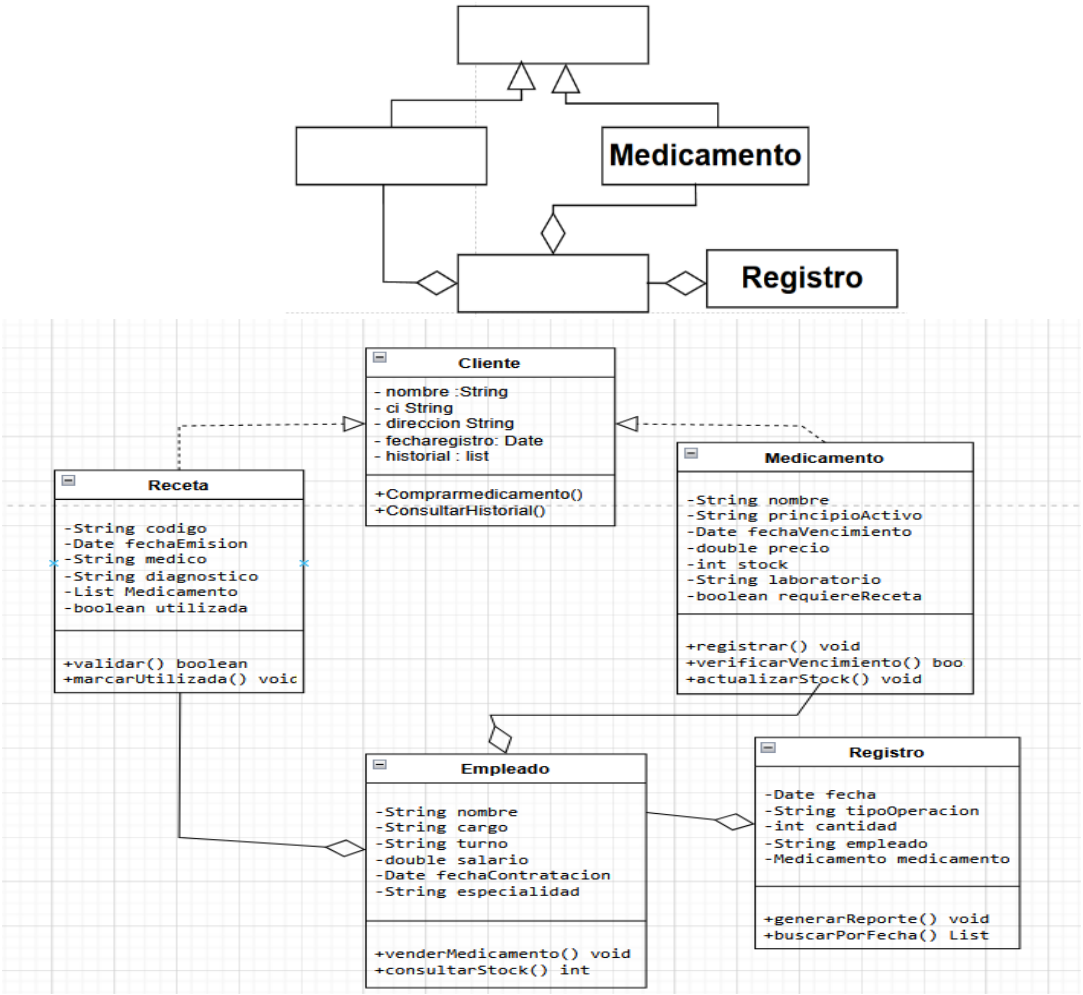
```
        for i in range(self.nro_ropas):
            prenda_actual = self.ropa[i]
            if prenda_actual:
                if (material and prenda_actual.material == material) or (tipo and prenda_actual.tipo == tipo):
                    eliminadas += 1
                else:
                    nuevas_prendas.append(prenda_actual)

        self.ropa = nuevas_prendas + [None] * (20 - len(nuevas_prendas))
        self.nro_ropas = len(nuevas_prendas)

        print(f"Se eliminaron {eliminadas} prendas")

    def mostrar_prendas_material_tipo(self, material=None, tipo=None):
        print(f"\n--- Prendas (Material: {material}, Tipo: {tipo}) ---")
        encontradas = False

        for i in range(self.nro_ropas):
            prenda_actual = self.ropa[i]
            if prenda_actual:
                if (not material or prenda_actual.material == material) and (not tipo or prenda_actual.tipo == tip
                    print(f"- {prenda_actual}")
                    encontradas = True

        if not encontradas:
            print("No se encontraron prendas con esos criterios")


ropero = Ropero("Madera")

prendas = [
    Ropa("Camisa", "Algodón"),
    Ropa("Pantalón", "Jean"),
    Ropa("Vestido", "Seda"),
    Ropa("Camisa", "Lino"),
    Ropa("Chaqueta", "Cuero"),
    Ropa("Pantalón", "Algodón"),
    Ropa("Blusa", "Seda"),
    Ropa("Falda", "Jean")
]

print("=== ADICIONAR PRENDAS AL ROPERO ===")
ropero.adicionar_n_prendas(prendas)
```

```
]

print("=== ADICIONAR PRENDAS AL ROPERO ===")
ropero.adicionar_n_prendas(prendas)

print("\n=== MOSTRAR TODAS LAS PRENDAS ===")
ropero.mostrar_prendas_material_tipo()

print("\n=== ELIMINAR PRENDAS DE MATERIAL 'Jean' ===")
ropero.eliminar_prendas_material_tipo(material="Jean")

print("\n=== MOSTRAR PRENDAS DESPUÉS DE ELIMINAR ===")
ropero.mostrar_prendas_material_tipo()

print("\n=== MOSTRAR PRENDAS DE MATERIAL 'Algodón' ===")
ropero.mostrar_prendas_material_tipo(material="Algodón")

print("\n=== MOSTRAR PRENDAS DE TIPO 'Camisa' ===")
ropero.mostrar_prendas_material_tipo(tipo="Camisa")

print("\n=== ELIMINAR PRENDAS DE TIPO 'Camisa' ===")
ropero.eliminar_prendas_material_tipo(tipo="Camisa")

print("\n=== MOSTRAR TODAS LAS PRENDAS FINALES ===")
ropero.mostrar_prendas_material_tipo()
```

```
- Vestido - Seda
- Camisa - Lino
- Chaqueta - Cuero
- Pantalón - Algodón
- Blusa - Seda

=== MOSTRAR PRENDAS DE MATERIAL 'Algodón' ===

--- Prendas (Material: Algodón, Tipo: None) ---
- Camisa - Algodón
- Pantalón - Algodón

=== MOSTRAR PRENDAS DE TIPO 'Camisa' ===

--- Prendas (Material: None, Tipo: Camisa) ---
- Camisa - Algodón
- Camisa - Lino

=== ELIMINAR PRENDAS DE TIPO 'Camisa' ===
Se eliminaron 2 prendas

=== MOSTRAR TODAS LAS PRENDAS FINALES ===

--- Prendas (Material: None, Tipo: None) ---
- Vestido - Seda
- Chaqueta - Cuero
- Pantalón - Algodón
- Blusa - Seda
PS E:\UMSA\third\121\PracticaAuxiliar2>
```

Ejercicio 6. Completa el diagrama de clases usando las relaciones que se indica, agrega además almenos 2 atributos significativos

(**contexto : farmacia**):



Ejercicio 8. La entrada universitaria requiere un sistema para gestionar las diversas fraternidades, bailarines y facultades que participaran este año. Para ello se pide que pueda se pueda ver a los bailarines y a que fraternidad y facultad pertenecen, además conocer al encargado de cada fraternidad, ver las edades de los participantes y verificar que no estén en 2 o más fraternidades. Cada vez que haya un nuevo integrante debe de ser registrado tomando sus datos personales, la facultad y fraternidad a la que pertenecen respectivamente.

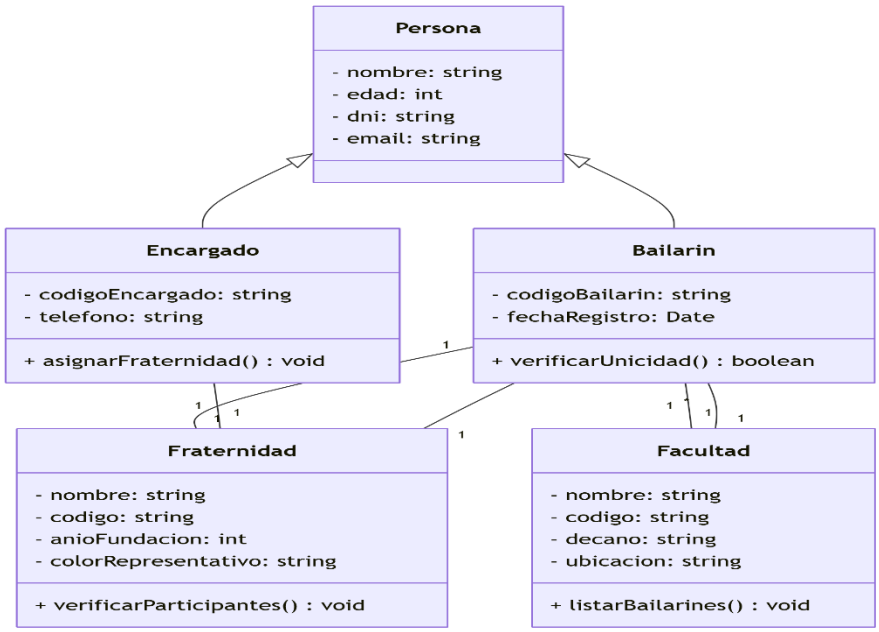
a) Identifica las clases y atributos necesarios para resolver el problema
b) Instancia al menos 5 participantes de los cuales 2 fraternidades con sus respectivos encargados y 2 facultades.
c) Resuelve lo que pide el cliente

```python
class Persona:
    def __init__(self, nombre, edad, dni):
        self.nombre = nombre
        self.edad = edad
        self.dni = dni

class Facultad:
    def __init__(self, nombre, codigo):
        self.nombre = nombre
        self.codigo = codigo

class Fraternidad:
    def __init__(self, nombre, encargado):
        self.nombre = nombre
        self.encargado = encargado
        self.bailarines = []

    def agregar_bailarin(self, bailarin):
        if bailarin.fraternidad:
            print(f"Error: {bailarin.nombre} ya pertenece a {bailarin.fraternidad.nombre}")
            return False
        self.bailarines.append(bailarin)
        bailarin.fraternidad = self
        print(f"{bailarin.nombre} agregado a {self.nombre}")
        return True

class Bailarin(Persona):
    def __init__(self, nombre, edad, dni, facultad):
        super().__init__(nombre, edad, dni)
        self.facultad = facultad
        self.fraternidad = None

class SistemaFraternidades:
    def __init__(self):
        self.fraternidades = []
        self.facultades = []
        self.bailarines = []

    def registrar_facultad(self, facultad):
        self.facultades.append(facultad)

    def registrar_fraternidad(self, fraternidad):
        self.fraternidades.append(fraternidad)

    def registrar_bailarin(self, nombre, edad, dni, facultad_nombre):
        facultad = next((f for f in self.facultades if f.nombre == facultad_nombre), None)
        if not facultad:
            print(f"Error: Facultad {facultad_nombre} no encontrada")
```

```
            print(f"ERROR: Facultad {facultad_nombre} no encontrada")
            return None

        bailarin = Bailarin(nombre, edad, dni, facultad)
        self.bailarines.append(bailarin)
        return bailarin

    def asignar_bailarin_fraternidad(self, bailarin_nombre, fraternidad_nombre):
        bailarin = next((b for b in self.bailarines if b.nombre == bailarin_nombre), None)
        fraternidad = next((f for f in self.fraternidades if f.nombre == fraternidad_nombre), None)

        if not bailarin or not fraternidad:
            print("Bailarín o fraternidad no encontrados")
            return False

        return fraternidad.agregar_bailarin(bailarin)

    def mostrar_bailarines_fraternidad_facultad(self):
        print("\n=== BAILARINES POR FRATERNIDAD Y FACULTAD ===")
        for fraternidad in self.fraternidades:
            print(f"\nFraternidad: {fraternidad.nombre}")
            print(f"Encargado: {fraternidad.encargado.nombre}")
            if fraternidad.bailarines:
                for bailarin in fraternidad.bailarines:
                    print(f"  - {bailarin.nombre} | Facultad: {bailarin.facultad.nombre} | Edad: {bailarin.edad}")
            else:
                print("  No tiene bailarines")

    def mostrar_edades_participantes(self):
        print("\n=== EDADES DE LOS PARTICIPANTES ===")
        for bailarin in self.bailarines:
            print(f"{bailarin.nombre}: {bailarin.edad} años")

    def verificar_multiples_fraternidades(self):
        print("\n=== VERIFICACIÓN DE FRATERNIDADES MÚLTIPLES ===")
        bailarines_sin_fraternidad = []
        bailarines_con_fraternidad = []

        for bailarin in self.bailarines:
            if bailarin.fraternidad:
                bailarines_con_fraternidad.append(bailarin)
            else:
                bailarines_sin_fraternidad.append(bailarin)

        print(f"Bailarines sin fraternidad: {len(bailarines_sin_fraternidad)}")
        for bailarin in bailarines_sin_fraternidad:
            print(f"  - {bailarin.nombre}")
```

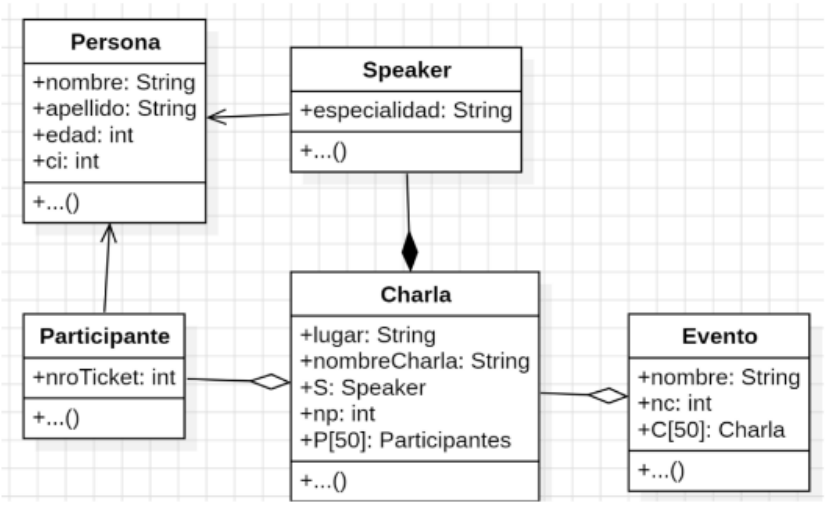
```
        print(f"Bailarines con fraternidad: {len(bailarines_con_fraternidad)}")
        for bailarin in bailarines_con_fraternidad:
            print(f"  - {bailarin.nombre} -> {bailarin.fraternidad.nombre}")

        return len(bailarines_sin_fraternidad) == 0

sistema = SistemaFraternidades()

facultad1 = Facultad("Ingeniería", "FING001")
facultad2 = Facultad("Medicina", "FMED002")

sistema.registrar_facultad(facultad1)
sistema.registrar_facultad(facultad2)

encargado1 = Persona("Carlos Ruiz", 25, "12345678")
encargado2 = Persona("María Lopez", 24, "87654321")

fraternidad1 = Fraternidad("Los Tigres", encargado1)
fraternidad2 = Fraternidad("Los Leones", encargado2)

sistema.registrar_fraternidad(fraternidad1)
sistema.registrar_fraternidad(fraternidad2)

bailarin1 = sistema.registrar_bailarin("Juan Pérez", 20, "11111111", "Ingeniería")
bailarin2 = sistema.registrar_bailarin("Ana García", 21, "22222222", "Ingeniería")
bailarin3 = sistema.registrar_bailarin("Luis Martínez", 22, "33333333", "Medicina")
bailarin4 = sistema.registrar_bailarin("Sofia Rodríguez", 19, "44444444", "Medicina")
bailarin5 = sistema.registrar_bailarin("Pedro Sánchez", 20, "55555555", "Ingeniería")
```

```python
sistema.asignar_bailarin_fraternidad("Juan Pérez", "Los Tigres")
sistema.asignar_bailarin_fraternidad("Ana García", "Los Tigres")
sistema.asignar_bailarin_fraternidad("Luis Martínez", "Los Leones")
sistema.asignar_bailarin_fraternidad("Sofia Rodríguez", "Los Leones")

print("=== SISTEMA DE FRATERNIDADES UNIVERSITARIAS ===")

sistema.mostrar_bailarines_fraternidad_facultad()

sistema.mostrar_edades_participantes()

sistema.verificar_multiples_fraternidades()

print(f"\n=== INTENTO DE ASIGNACIÓN MÚLTIPLE ===")
sistema.asignar_bailarin_fraternidad("Juan Pérez", "Los Leones")

print(f"\n=== ASIGNANDO ÚLTIMO BAILARÍN ===")
sistema.asignar_bailarin_fraternidad("Pedro Sánchez", "Los Tigres")

sistema.mostrar_bailarines_fraternidad_facultad()
sistema.verificar_multiples_fraternidades()
```




```
=== INTENTO DE ASIGNACION MULTIPLE ===
Error: Juan Pérez ya pertenece a Los Tigres

=== ASIGNANDO ÚLTIMO BAILARÍN ===
Pedro Sánchez agregado a Los Tigres

=== BAILARINES POR FRATERNIDAD Y FACULTAD ===

Fraternidad: Los Tigres
Encargado: Carlos Ruiz
  - Juan Pérez | Facultad: Ingeniería | Edad: 20
  - Ana García | Facultad: Ingeniería | Edad: 21
  - Pedro Sánchez | Facultad: Ingeniería | Edad: 20

Fraternidad: Los Leones
Encargado: María Lopez
  - Luis Martínez | Facultad: Medicina | Edad: 22
  - Sofia Rodríguez | Facultad: Medicina | Edad: 19

=== VERIFICACIÓN DE FRATERNIDADES MÚLTIPLES ===
Bailarines sin fraternidad: 0
Bailarines con fraternidad: 5
  - Juan Pérez -> Los Tigres
  - Ana García -> Los Tigres
  - Luis Martínez -> Los Leones
  - Sofia Rodríguez -> Los Leones
  - Pedro Sánchez -> Los Tigres
PS E:\UMSA\third\121\PracticaAuxiliar2>
```

Ejercicio10. Usando el siguiente diagrama de clases se le pide resolver:



a) Cuál es la edad promedio de los participantes en el evento.
b) Verificar si la persona nombre “X” y apellido “Y” se encuentra en alguna de las charlas tantocomo participante o como Speaker.
c) El speaker con el C.I. X no pudo asistir por lo cual se deberá eliminar todas las charlas que iba a dar.
d) Ordenar las charlas por el número de participantes.

Relaciones > Charla.py > ...

```python
class Persona:
    def __init__(self, nombre, apellido, edad, ci):
        self.nombre = nombre
        self.apellido = apellido
        self.edad = edad
        self.ci = ci

class Speaker(Persona):
    def __init__(self, nombre, apellido, edad, ci, especialidad):
        super().__init__(nombre, apellido, edad, ci)
        self.especialidad = especialidad

class Participante(Persona):
    def __init__(self, nombre, apellido, edad, ci, nroTicket):
        super().__init__(nombre, apellido, edad, ci)
        self.nroTicket = nroTicket

class Charla:
    def __init__(self, lugar, nombreCharla, speaker):
        self.lugar = lugar
        self.nombreCharla = nombreCharla
        self.speaker = speaker
        self.np = 0
        self.participantes = [None] * 50

    def agregar_participante(self, participante):
        if self.np < 50:
            self.participantes[self.np] = participante
            self.np += 1
            return True
        return False

    def obtener_participantes(self):
        return [p for p in self.participantes if p is not None]

class Evento:
    def __init__(self, nombre):
        self.nombre = nombre
        self.nc = 0
        self.charlas = [None] * 50

    def agregar_charla(self, charla):
        if self.nc < 50:
            self.charlas[self.nc] = charla
            self.nc += 1
            return True
        return False

    def obtener charlas(self):
```

```
        return [c for c in self.charlas if c is not None]

    def edad_promedio_participantes(self):
        print("\n=== EDAD PROMEDIO DE PARTICIPANTES ===")
        total_edad = 0
        total_participantes = 0
        participantes_unicos = set()

        for charla in self.obtener_charlas():
            for participante in charla.obtener_participantes():
                if participante.ci not in participantes_unicos:
                    participantes_unicos.add(participante.ci)
                    total_edad += participante.edad
                    total_participantes += 1

        if total_participantes > 0:
            promedio = total_edad / total_participantes
            print(f"Edad promedio: {promedio:.2f} años")
            print(f"Total participantes únicos: {total_participantes}")
            return promedio
        else:
            print("No hay participantes en el evento")
            return 0

    def buscar_persona(self, nombre, apellido):
        print(f"\n=== BUSCANDO PERSONA: {nombre} {apellido} ===")
        encontrado = False

        for charla in self.obtener_charlas():
            if charla.speaker and charla.speaker.nombre == nombre and charla.speaker.apellido == apellido:
                print(f"Encontrado como SPEAKER en: '{charla.nombreCharla}'")
                encontrado = True

            for participante in charla.obtener_participantes():
                if participante.nombre == nombre and participante.apellido == apellido:
                    print(f"Encontrado como PARTICIPANTE en: '{charla.nombreCharla}'")
                    encontrado = True

        if not encontrado:
            print("Persona no encontrada en el evento")

        return encontrado

    def eliminar_charlas_por_speaker_ci(self, ci):
        print(f"\n=== ELIMINANDO CHARLAS DEL SPEAKER CON CI: {ci} ===")
        charlas_eliminadas = 0
        nuevas_charlas = []

        for charla in self.obtener_charlas():
            if charla.speaker and charla.speaker.ci == ci:
                print(f"Eliminada charla: '{charla.nombreCharla}'")
                charlas_eliminadas += 1
            else:
                nuevas_charlas.append(charla)

        self.charlas = nuevas_charlas + [None] * (50 - len(nuevas_charlas))
        self.nc = len(nuevas_charlas)
        print(f"Total charlas eliminadas: {charlas_eliminadas}")
```

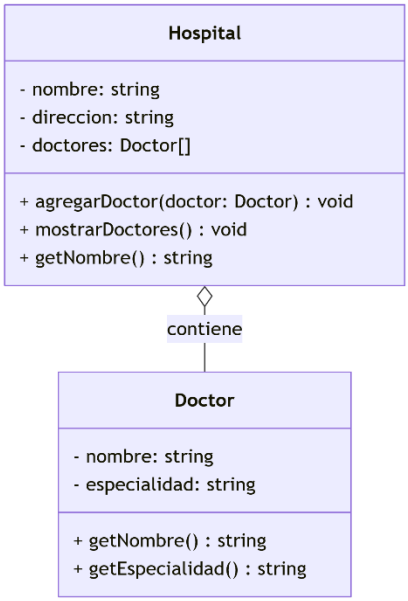
```
        def eliminar_charlas_por_speaker_ci(self, ci):
            print(f"Total charlas eliminadas: {charlas_eliminadas}")
            return charlas_eliminadas

        def ordenar_charlas_por_participantes(self):
            print("\n=== CHARLAS ORDENADAS POR NÚMERO DE PARTICIPANTES ===")
            charlas_ordenadas = sorted(self.obtener_charlas(),
                                       key=lambda charla: charla.np,
                                       reverse=True)

            for i, charla in enumerate(charlas_ordenadas, 1):
                print(f"{i}. '{charla.nombreCharla}' - {charla.np} participantes - Speaker: {charla.speaker.no

            return charlas_ordenadas

    evento = Evento("Conferencia de Tecnología 2024")

    speaker1 = Speaker("Ana", "García", 35, "1234567", "Inteligencia Artificial")
    speaker2 = Speaker("Carlos", "López", 40, "7654321", "Blockchain")
    speaker3 = Speaker("María", "Rodríguez", 32, "1111111", "Ciberseguridad")

    participante1 = Participante("Juan", "Pérez", 25, "2222222", 1001)
    participante2 = Participante("Laura", "Martínez", 28, "3333333", 1002)
    participante3 = Participante("Pedro", "Sánchez", 30, "4444444", 1003)
    participante4 = Participante("Sofia", "Díaz", 22, "5555555", 1004)
    participante5 = Participante("Miguel", "Torres", 26, "6666666", 1005)
    participante6 = Participante("Elena", "Castro", 29, "7777777", 1006)

    charla1 = Charla("Auditorio A", "Introducción a IA", speaker1)
    charla2 = Charla("Auditorio B", "Blockchain en Finanzas", speaker2)
    charla3 = Charla("Sala C", "Seguridad Informática", speaker3)
    charla4 = Charla("Auditorio A", "IA Avanzada", speaker1)

    charla1.agregar_participante(participante1)
    charla1.agregar_participante(participante2)
    charla1.agregar_participante(participante3)

    charla2.agregar_participante(participante2)
    charla2.agregar_participante(participante4)

    charla3.agregar_participante(participante1)
    charla3.agregar_participante(participante5)
    charla3.agregar_participante(participante6)

    charla4.agregar_participante(participante3)
    charla4.agregar_participante(participante4)
    charla4.agregar_participante(participante5)
    charla4.agregar_participante(participante6)

    evento.agregar_charla(charla1)
    evento.agregar_charla(charla2)
    evento.agregar_charla(charla3)
    evento.agregar_charla(charla4)

    print("=== SISTEMA DE EVENTOS Y CHARLAS ===")

    evento.edad_promedio_participantes()

    evento.buscar_persona("Ana", "García")
```

```python
evento.buscar_persona("Ana", "García")
evento.buscar_persona("Juan", "Pérez")
evento.buscar_persona("Luis", "Fernández")

evento.ordenar_charlas_por_participantes()

evento.eliminar_charlas_por_speaker_ci("1234567")

print("\n=== CHARLAS DESPUÉS DE ELIMINACIÓN ===")
for charla in evento.obtener_charlas():
    print(f"'{charla.nombreCharla}' - {charla.np} participantes")

evento.ordenar_charlas_por_participantes()
```

```
Encontrado como SPEAKER en: 'Introducción a IA'
Encontrado como SPEAKER en: 'IA Avanzada'

=== BUSCANDO PERSONA: Juan Pérez ===
Encontrado como PARTICIPANTE en: 'Introducción a IA'
Encontrado como PARTICIPANTE en: 'Seguridad Informática'

=== BUSCANDO PERSONA: Luis Fernández ===
Persona no encontrada en el evento

=== CHARLAS ORDENADAS POR NÚMERO DE PARTICIPANTES ===
1. 'IA Avanzada' - 4 participantes - Speaker: Ana
2. 'Introducción a IA' - 3 participantes - Speaker: Ana
3. 'Seguridad Informática' - 3 participantes - Speaker: María
4. 'Blockchain en Finanzas' - 2 participantes - Speaker: Carlos

=== ELIMINANDO CHARLAS DEL SPEAKER CON CI: 1234567 ===
Eliminada charla: 'Introducción a IA'
Eliminada charla: 'IA Avanzada'
Total charlas eliminadas: 2

=== CHARLAS DESPUÉS DE ELIMINACIÓN ===
'Blockchain en Finanzas' - 2 participantes
'Seguridad Informática' - 3 participantes

=== CHARLAS ORDENADAS POR NÚMERO DE PARTICIPANTES ===
1. 'Seguridad Informática' - 3 participantes - Speaker: María
2. 'Blockchain en Finanzas' - 2 participantes - Speaker: Carlos
PS E:\UMSA\third\121\PracticaAuxiliar2>
```

Ejercicio 12. Crea una relación de agregación entre **Hospital** y **Doctor**. Los doctores pueden trabajar en distintos hospitales.

a) Crea la clase Doctor con atributos nombre, especialidad.
b) Crea la clase Hospital que tiene una lista de doctores.
c) Permite asignar doctores a hospitales y mostrar los doctores de cada hospital




```java
public class Doctor {
    private String nombre;
    private String especialidad;

    public Doctor(String nombre, String especialidad) {
        this.nombre = nombre;
        this.especialidad = especialidad;
    }

    public String getNombre() {
        return nombre;
    }

    public String getEspecialidad() {
        return especialidad;
    }

    @Override
    public String toString() {
        return nombre + " - " + especialidad;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        Doctor doctor = (Doctor) obj;
        return nombre.equals(doctor.nombre) && especialidad.equals(doctor.especialidad);
    }
}
```

```java
public class SistemaHospital{

    public static int contarHospitales(Doctor doctor, Hospital... hospitales) {
        int count = 0;
        for (Hospital hospital : hospitales) {
            if (hospital.tieneDoctor(doctor)) {
                count++;
            }
        }
        return count;
    }

    Run | Debug
    public static void main(String[] args) {
        System.out.println(x: "=== SISTEMA HOSPITAL-DOCTORES ===\n");


        Doctor doctor1 = new Doctor(nombre: "Dr. Carlos Ruiz", especialidad: "Cardiología");
        Doctor doctor2 = new Doctor(nombre: "Dra. Ana García", especialidad: "Pediatría");
        Doctor doctor3 = new Doctor(nombre: "Dr. Luis Martínez", especialidad: "Cirugía");
        Doctor doctor4 = new Doctor(nombre: "Dra. María López", especialidad: "Neurología");
        Doctor doctor5 = new Doctor(nombre: "Dr. Pedro Sánchez", especialidad: "Traumatología");

        Hospital hospital1 = new Hospital(nombre: "Hospital Central");
        Hospital hospital2 = new Hospital(nombre: "Hospital del Norte");
        Hospital hospital3 = new Hospital(nombre: "Clínica Sur");

        System.out.println(x: "=== ASIGNANDO DOCTORES A HOSPITALES ===\n");

        hospital1.asignarDoctor(doctor1);
        hospital1.asignarDoctor(doctor2);
        hospital1.asignarDoctor(doctor3);

        hospital2.asignarDoctor(doctor2);
        hospital2.asignarDoctor(doctor4);
        hospital2.asignarDoctor(doctor5);


        hospital3.asignarDoctor(doctor1);
        hospital3.asignarDoctor(doctor3);
        hospital3.asignarDoctor(doctor5);


        System.out.println(x: "\n=== INTENTO DE ASIGNACIÓN DUPLICADA ===");
        hospital1.asignarDoctor(doctor1);

        System.out.println(x: "\n=== MOSTRANDO DOCTORES POR HOSPITAL ===");

        hospital1.mostrarDoctores();
        hospital2.mostrarDoctores();
        hospital3.mostrarDoctores();

        System.out.println(x: "\n=== RESUMEN DE ASIGNACIONES ===");
        System.out.println("Doctor " + doctor1.getNombre() + " trabaja en " +
                           contarHospitales(doctor1, hospital1, hospital2, hospital3) + " hospital(es)");
        System.out.println("Doctor " + doctor2.getNombre() + " trabaja en " +
                           contarHospitales(doctor2, hospital1, hospital2, hospital3) + " hospital(es)");
        System.out.println("Doctor " + doctor5.getNombre() + " trabaja en " +
                           contarHospitales(doctor5, hospital1, hospital2, hospital3) + " hospital(es)");
    }
```

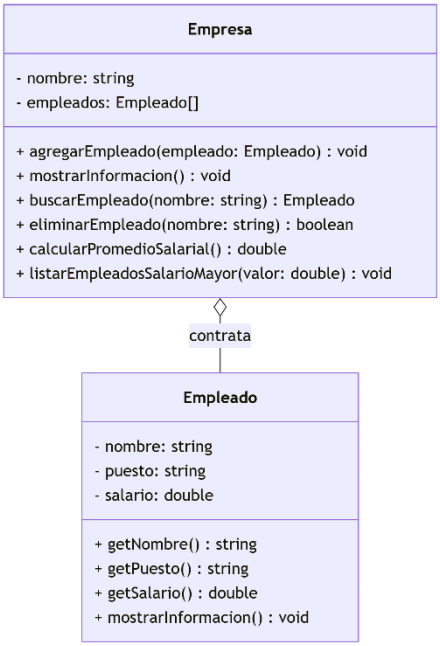
```java
import java.util.ArrayList;
import java.util.List;

public class Hospital {
    private String nombre;
    private List<Doctor> doctores;

    public Hospital(String nombre) {
        this.nombre = nombre;
        this.doctores = new ArrayList<>();
    }

    public String getNombre() {
        return nombre;
    }

    public void asignarDoctor(Doctor doctor) {
        if (!doctores.contains(doctor)) {
            doctores.add(doctor);
            System.out.println("Doctor " + doctor.getNombre() + " asignado al hospital " + nombre);
        } else {
            System.out.println("El doctor " + doctor.getNombre() + " ya está asignado a este hospital");
        }
    }

    public void mostrarDoctores() {
        System.out.println("\n=== Doctores del Hospital " + nombre + " ===");
        if (doctores.isEmpty()) {
            System.out.println(x: "No hay doctores asignados a este hospital");
        } else {
            for (int i = 0; i < doctores.size(); i++) {
                System.out.println((i + 1) + ". " + doctores.get(i));
            }
        }
    }

    public List<Doctor> getDoctores() {
        return new ArrayList<>(doctores);
    }

    public boolean tieneDoctor(Doctor doctor) {
        return doctores.contains(doctor);
    }
}
```

```
=== ASIGNANDO DOCTORES A HOSPITALES ===

Doctor Dr. Carlos Ruiz asignado al hospital Hospital Central
Doctor Dra. Ana García asignado al hospital Hospital Central
Doctor Dr. Luis Martínez asignado al hospital Hospital Central
Doctor Dra. Ana García asignado al hospital Hospital del Norte
Doctor Dra. María López asignado al hospital Hospital del Norte
Doctor Dr. Pedro Sánchez asignado al hospital Hospital del Norte
Doctor Dr. Carlos Ruiz asignado al hospital Clínica Sur
Doctor Dr. Luis Martínez asignado al hospital Clínica Sur
Doctor Dr. Pedro Sánchez asignado al hospital Clínica Sur

=== INTENTO DE ASIGNACIÓN DUPLICADA ===
El doctor Dr. Carlos Ruiz ya está asignado a este hospital

=== MOSTRANDO DOCTORES POR HOSPITAL ===

=== Doctores del Hospital Hospital Central ===
1. Dr. Carlos Ruiz - Cardiología
2. Dra. Ana García - Pediatría
3. Dr. Luis Martínez - Cirugía

=== Doctores del Hospital Hospital del Norte ===
1. Dra. Ana García - Pediatría
2. Dra. María López - Neurología
3. Dr. Pedro Sánchez - Traumatología

=== Doctores del Hospital Clínica Sur ===
1. Dr. Carlos Ruiz - Cardiología
2. Dr. Luis Martínez - Cirugía
3. Dr. Pedro Sánchez - Traumatología

=== RESUMEN DE ASIGNACIONES ===
Doctor Dr. Carlos Ruiz trabaja en 2 hospital(es)
Doctor Dra. Ana García trabaja en 2 hospital(es)
Doctor Dr. Pedro Sánchez trabaja en 2 hospital(es)
PS E:\UMSA\third\121\PracticaAuxiliar2>
```

Ejercicio 14. Agregación: Empresa y empleados

- Clase Empleado con los atributos: nombre, puesto, salario
- Clase Empresa con los atributos: nombre, empleados (vector de objetos Empleado) Se pide:

a) Crear una empresa y agregar varios empleados
b) Mostrar la información de la empresa y sus empleados.
c) Implementar un método en Empresa para buscar un empleado por nombre y retornar su información.
d) Crear un método para eliminar un empleado de la empresa por nombre.
e) Agregar un método que calcule y muestre el promedio salarial de todos los empleados. Implementar un método para listar todos los empleados que tengan un salario mayor a un valor dado.




```java
public class Empleado {
    private String nombre;
    private String puesto;
    private double salario;

    public Empleado(String nombre, String puesto, double salario) {
        this.nombre = nombre;
        this.puesto = puesto;
        this.salario = salario;
    }

    public String getNombre() {
        return nombre;
    }

    public String getPuesto() {
        return puesto;
    }
                double Empleado.getSalario()
    public double getSalario() {
        return salario;
    }

    public void setSalario(double salario) {
        this.salario = salario;
    }

    @Override
    public String toString() {
        return String.format(format: "%-15s %-20s bs%,.2f", nombre, puesto, salario);
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        Empleado empleado = (Empleado) obj;
        return nombre.equalsIgnoreCase(empleado.nombre);
    }
```

```java
import java.util.ArrayList;
import java.util.List;

public class Empresa {
    private String nombre;
    private List<Empleado> empleados;

    public Empresa(String nombre) {
        this.nombre = nombre;
        this.empleados = new ArrayList<>();
    }

    public String getNombre() {
        return nombre;
    }

    public void agregarEmpleado(Empleado empleado) {
        if (!empleados.contains(empleado)) {
            empleados.add(empleado);
            System.out.println("Empleado " + empleado.getNombre() + " agregado a la empresa " + nombre);
        } else {
            System.out.println("El empleado " + empleado.getNombre() + " ya existe en la empresa");
        }
    }

    public void mostrarInformacion() {
        System.out.println("\n=== INFORMACIÓN DE LA EMPRESA: " + nombre + " ===");
        System.out.println("Total de empleados: " + empleados.size());
        System.out.println(x: "\n--- LISTA DE EMPLEADOS ---");
        System.out.printf(format: "%-15s %-20s %s\n", ...args: "NOMBRE", "PUESTO", "SALARIO");
        System.out.println(x: "-------------------------------------------------");

        if (empleados.isEmpty()) {
            System.out.println(x: "No hay empleados en la empresa");
        } else {
            for (int i = 0; i < empleados.size(); i++) {
                System.out.println((i + 1) + ". " + empleados.get(i));
            }
        }
    }

    public Empleado buscarEmpleado(String nombre) {
        for (Empleado empleado : empleados) {
            if (empleado.getNombre().equalsIgnoreCase(nombre)) {
                return empleado;
            }
        }
        return null;
    }
```

```java
    public boolean eliminarEmpleado(String nombre) {
        Empleado empleadoAEliminar = null;
        for (Empleado empleado : empleados) {
            if (empleado.getNombre().equalsIgnoreCase(nombre)) {
                empleadoAEliminar = empleado;
                break;
            }
        }

        if (empleadoAEliminar != null) {
            empleados.remove(empleadoAEliminar);
            System.out.println("Empleado " + nombre + " eliminado de la empresa");
            return true;
        } else {
            System.out.println("Empleado " + nombre + " no encontrado en la empresa");
            return false;
        }
    }

    public double calcularPromedioSalarial() {
        if (empleados.isEmpty()) {
            return 0.0;
        }

        double totalSalarios = 0;
        for (Empleado empleado : empleados) {
            totalSalarios += empleado.getSalario();
        }

        return totalSalarios / empleados.size();
    }
```

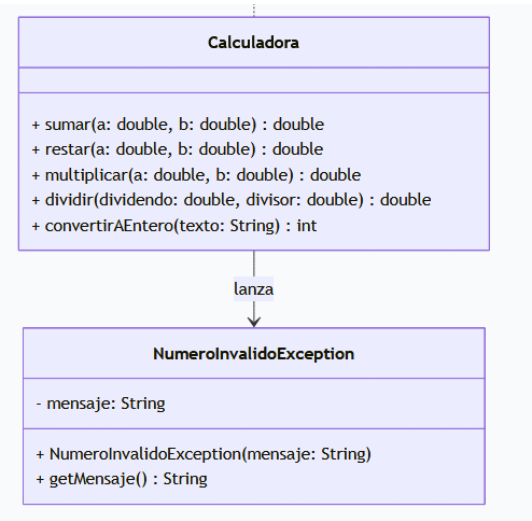
```java

    public void mostrarEmpleadosSalarioMayorA(double salarioMinimo) {
        System.out.println("\n=== EMPLEADOS CON SALARIO MAYOR A bs" + salarioMinimo + " ===");
        System.out.printf(format: "%-15s %-20s %s\n", ...args: "NOMBRE", "PUESTO", "SALARIO");
        System.out.println(x: "-------------------------------------------------");

        boolean encontrados = false;
        for (Empleado empleado : empleados) {
            if (empleado.getSalario() > salarioMinimo) {
                System.out.println(empleado);
                encontrados = true;
            }
        }

        if (!encontrados) {
            System.out.println("No se encontraron empleados con salario mayor a bs" + salarioMinimo);
        }
    }

    public int getCantidadEmpleados() {
        return empleados.size();
    }
}
```

```
=== ASIGNANDO DOCTORES A HOSPITALES ===

Doctor Dr. Carlos Ruiz asignado al hospital Hospital Central
Doctor Dra. Ana García asignado al hospital Hospital Central
Doctor Dr. Luis Martínez asignado al hospital Hospital Central
Doctor Dra. Ana García asignado al hospital Hospital del Norte
Doctor Dra. María López asignado al hospital Hospital del Norte
Doctor Dr. Pedro Sánchez asignado al hospital Hospital del Norte
Doctor Dr. Carlos Ruiz asignado al hospital Clínica Sur
Doctor Dr. Luis Martínez asignado al hospital Clínica Sur
Doctor Dr. Pedro Sánchez asignado al hospital Clínica Sur

=== INTENTO DE ASIGNACIÓN DUPLICADA ===
El doctor Dr. Carlos Ruiz ya está asignado a este hospital

=== MOSTRANDO DOCTORES POR HOSPITAL ===

=== Doctores del Hospital Hospital Central ===
1. Dr. Carlos Ruiz - Cardiología
2. Dra. Ana García - Pediatría
3. Dr. Luis Martínez - Cirugía

=== Doctores del Hospital Hospital del Norte ===
1. Dra. Ana García - Pediatría
2. Dra. María López - Neurología
3. Dr. Pedro Sánchez - Traumatología

=== Doctores del Hospital Clínica Sur ===
1. Dr. Carlos Ruiz - Cardiología
2. Dr. Luis Martínez - Cirugía
3. Dr. Pedro Sánchez - Traumatología

=== RESUMEN DE ASIGNACIONES ===
Doctor Dr. Carlos Ruiz trabaja en 2 hospital(es)
Doctor Dra. Ana García trabaja en 2 hospital(es)
Doctor Dr. Pedro Sánchez trabaja en 2 hospital(es)
PS E:\UMSA\third\121\PracticaAuxiliar2>
```

TEMA: MANEJO DE EXCEPCIONES

Ejercicio 2. Se desea crear una calculadora que realice operaciones básicas, manejando errores comunes:

a) Crear la clase Calculadora con métodos estáticos para sumar, restar, multiplicar y dividir.
b) El método dividir debe lanzar una excepción **ArithmeticException** si el divisor es cero.
c) Implementar un método que convierta un String a número entero, lanzando una excepción personalizada **NumeroInvalidoException** si el valor no es numérico.
d) Crear un programa principal que pruebe todos los métodos, incluyendo casos con error.



ManejoDeExpreciones > Calculadora.py > probar_conversion_numeros

```python
class NumeroInvalidoException(Exception):
    def __init__(self, mensaje):
        super().__init__(mensaje)

class Calculadora:
    @staticmethod
    def sumar(a, b):
        return a + b

    @staticmethod
    def restar(a, b):
        return a - b

    @staticmethod
    def multiplicar(a, b):
        return a * b

    @staticmethod
    def dividir(dividendo, divisor):
        if divisor == 0:
            raise ArithmeticError("ERROR: No se puede dividir por cero")
        return dividendo / divisor

    @staticmethod
    def convertir_a_entero(numero_str):
        try:
            return int(numero_str)
        except ValueError:
            raise NumeroInvalidoException(f"ERROR: '{numero_str}' no es un número entero válido")

    @staticmethod
    def convertir_a_float(numero_str):
        try:
            return float(numero_str)
        except ValueError:
            raise NumeroInvalidoException(f"ERROR: '{numero_str}' no es un número válido")
```

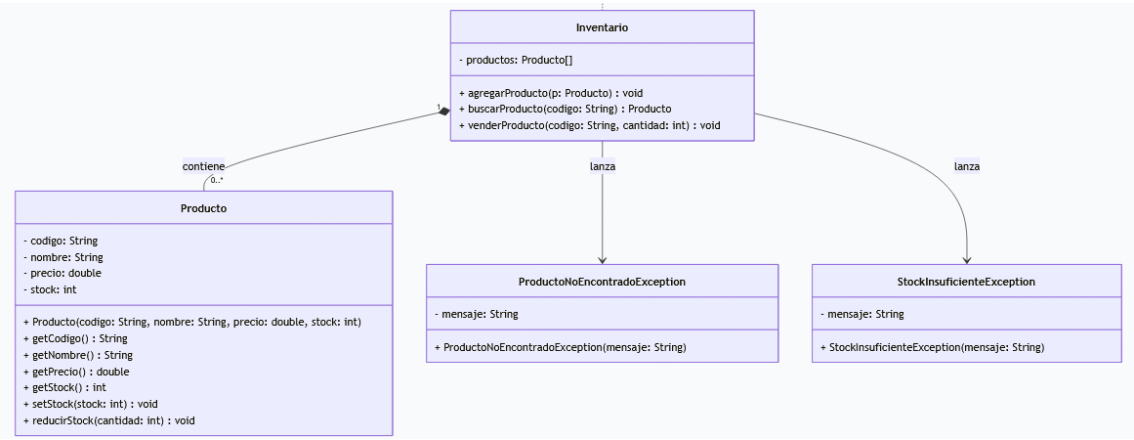
```python
def probar_operaciones_basicas():
    print("=== PRUEBA DE OPERACIONES BÁSICAS ===")
    a, b = 10, 5
    print(f"{a} + {b} = {Calculadora.sumar(a, b)}")
    print(f"{a} - {b} = {Calculadora.restar(a, b)}")
    print(f"{a} * {b} = {Calculadora.multiplicar(a, b)}")
    try:
        print(f"{a} / {b} = {Calculadora.dividir(a, b)}")
    except ArithmeticError as e:
        print(e)

def probar_division_por_cero():
    print("\n=== PRUEBA DE DIVISIÓN POR CERO ===")
    try:
        resultado = Calculadora.dividir(10, 0)
        print(f"Resultado: {resultado}")
    except ArithmeticError as e:
        print(f"Excepción capturada: {e}")

def probar_conversion_numeros():
    print("\n=== PRUEBA DE CONVERSIÓN DE NÚMEROS ===")
    numeros_prueba = ["123", "45.67", "abc", "-100", "12a3", "3.14"]
    for numero_str in numeros_prueba:
        try:
            numero_entero = Calculadora.convertir_a_entero(numero_str)
            print(f"'{numero_str}' convertido a entero: {numero_entero}")
        except NumeroInvalidoException as e:
            print(f"No se pudo convertir a entero: {e}")
            try:
                numero_float = Calculadora.convertir_a_float(numero_str)
                print(f"'{numero_str}' convertido a float: {numero_float}")
            except NumeroInvalidoException as e2:
                print(f"Tampoco se pudo convertir a float: {e2}")

def probar_calculadora_completa():
    print("\n=== PRUEBA COMPLETA DE CALCULADORA ===")
    operandos1 = ["15", "8", "abc", "20", "10.5"]
    operandos2 = ["3", "0", "4", "def", "2"]
    for i in range(len(operandos1)):
        print(f"\n--- Operación {i + 1} ---")
        print(f"Operando 1: {operandos1[i]}, Operando 2: {operandos2[i]}")
        try:
            try:
                num1 = Calculadora.convertir_a_entero(operandos1[i])
            except NumeroInvalidoException:
                num1 = Calculadora.convertir_a_float(operandos1[i])
            try:
                num2 = Calculadora.convertir_a_entero(operandos2[i])
            except NumeroInvalidoException:
                num2 = Calculadora.convertir_a_float(operandos2[i])
            print(f"Suma: {Calculadora.sumar(num1, num2)}")
            print(f"Resta: {Calculadora.restar(num1, num2)}")
            print(f"Multiplicación: {Calculadora.multiplicar(num1, num2)}")
            try:
                print(f"División: {Calculadora.dividir(num1, num2)}")
            except ArithmeticError as e:
                print(f"Error en división: {e}")
```

```
            except ArithmeticError as e:
                print(f"Error en división: {e}")
        except NumeroInvalidoException as e:
            print(f"Error en conversión: {e}")

def menu_interactivo():
    print("\n" + "="*50)
    print("=== CALCULADORA INTERACTIVA ===")
    print("="*50)
    while True:
        print("\nOpciones:")
        print("1. Realizar operación básica")
        print("2. Probar casos de error")
        print("3. Salir")
        opcion = input("\nSeleccione una opción: ")
        if opcion == "1":
            try:
                num1_str = input("Ingrese el primer número: ")
                num2_str = input("Ingrese el segundo número: ")
                try:
                    num1 = Calculadora.convertir_a_entero(num1_str)
                except NumeroInvalidoException:
                    num1 = Calculadora.convertir_a_float(num1_str)
                try:
                    num2 = Calculadora.convertir_a_entero(num2_str)
                except NumeroInvalidoException:
                    num2 = Calculadora.convertir_a_float(num2_str)
                print(f"\nResultados con {num1} y {num2}:")
                print(f"Suma: {Calculadora.sumar(num1, num2)}")
                print(f"Resta: {Calculadora.restar(num1, num2)}")
                print(f"Multiplicación: {Calculadora.multiplicar(num1, num2)}")
                try:
                    print(f"División: {Calculadora.dividir(num1, num2)}")
                except ArithmeticError as e:
                    print(f"División: {e}")
            except NumeroInvalidoException as e:
                print(f"Error: {e}")
        elif opcion == "2":
            print("\nEjecutando pruebas automáticas de errores...")
            probar_division_por_cero()
            probar_conversion_numeros()
        elif opcion == "3":
            print("¡Hasta luego!")
            break
        else:
            print("Opción no válida. Intente nuevamente.")

if __name__ == "__main__":
    print("=== CALCULADORA CON MANEJO DE EXCEPCIONES ===\n")
    probar_operaciones_basicas()
    probar_division_por_cero()
    probar_conversion_numeros()
    probar_calculadora_completa()
    menu_interactivo()
    print("\n=== PROGRAMA FINALIZADO ===")
```

```
=== PRUEBA DE DIVISIÓN POR CERO ===
Excepción capturada: ERROR: No se puede dividir por cero

=== PRUEBA DE CONVERSIÓN DE NÚMEROS ===
'123' convertido a entero: 123
No se pudo convertir a entero: ERROR: '45.67' no es un número entero válido
'45.67' convertido a float: 45.67
No se pudo convertir a entero: ERROR: 'abc' no es un número entero válido
Tampoco se pudo convertir a float: ERROR: 'abc' no es un número válido
'-100' convertido a entero: -100
No se pudo convertir a entero: ERROR: '12a3' no es un número entero válido
Tampoco se pudo convertir a float: ERROR: '12a3' no es un número válido
No se pudo convertir a entero: ERROR: '3.14' no es un número entero válido
'3.14' convertido a float: 3.14

=== PRUEBA COMPLETA DE CALCULADORA ===

--- Operación 1 ---
Operando 1: 15, Operando 2: 3
Suma: 18
Resta: 12
Multiplicación: 45
División: 5.0

--- Operación 2 ---
Operando 1: 8, Operando 2: 0
Suma: 8
Resta: 8
Multiplicación: 0
Error en división: ERROR: No se puede dividir por cero

--- Operación 3 ---
Operando 1: abc, Operando 2: 4
Error en conversión: ERROR: 'abc' no es un número válido

--- Operación 4 ---
Operando 1: 20, Operando 2: def
Error en conversión: ERROR: 'def' no es un número válido

--- Operación 5 ---
Operando 1: 10.5, Operando 2: 2
Suma: 12.5
Resta: 8.5
Multiplicación: 21.0
División: 5.25

================================================
=== CALCULADORA INTERACTIVA ===
================================================

Opciones:
1. Realizar operación básica
2. Probar casos de error
```

Ejercicio 4. Un sistema de inventario debe controlar errores en la gestión de productos:

a) Crear una clase Producto con atributos (codigo, nombre, precio y stock).
b) Crear la clase Inventario que contenga un vector de productos.
c) Implementar un método agregarProducto(Producto p) que **lance una excepción si el código ya existe o si el precio/stock son negativos.**
d) Implementar un método buscarProducto(String codigo) que lance una excepción personalizada **ProductoNoEncontradoException** si no existe.
e) Implementar un método venderProducto(String codigo, int cantidad) que reduzca el stock si hay suficiente, de lo contrario lance una excepción **StockInsuficienteException**.




```python
class FondosInsuficientesException(Exception):
    def __init__(self, mensaje):
        super().__init__(mensaje)

class CuentaBancaria:
    def __init__(self, numero_cuenta, titular, saldo_inicial=0):
        self.numero_cuenta = numero_cuenta
        self.titular = titular
        self.saldo = saldo_inicial

    def depositar(self, monto):
        if monto <= 0:
            raise ValueError("Error: El monto a depositar debe ser positivo")
        self.saldo += monto
        print(f"Depósito exitoso: +${monto:.2f}")

    def retirar(self, monto):
        if monto <= 0:
            raise ValueError("Error: El monto a retirar debe ser positivo")

        if monto > self.saldo:
            raise FondosInsuficientesException(
                f"Fondos insuficientes. Saldo: ${self.saldo:.2f}, Retiro: ${monto:.2f}"
            )

        self.saldo -= monto
        print(f"Retiro exitoso: -${monto:.2f}")

    def mostrar_info(self):
        print(f"Cuenta: {self.numero_cuenta}")
        print(f"Titular: {self.titular}")
        print(f"Saldo: ${self.saldo:.2f}")

def probar_depositos():
    print("=== PRUEBA DE DEPÓSITOS ===")
    cuenta = CuentaBancaria("12345", "Juan Pérez", 1000)
    cuenta.mostrar_info()

    try:
        print("\nDepósito válido:")
        cuenta.depositar(500)
        cuenta.mostrar_info()
    except ValueError as e:
        print(f"Error: {e}")
```

```
    try:
        print("\nDepósito con monto negativo:")
        cuenta.depositar(-100)
    except ValueError as e:
        print(f"Error esperado: {e}")

    try:
        print("\nDepósito con monto cero:")
        cuenta.depositar(0)
    except ValueError as e:
        print(f"Error esperado: {e}")

def probar_retiros():
    print("\n=== PRUEBA DE RETIROS ===")
    cuenta = CuentaBancaria("12345", "Juan Pérez", 1000)
    cuenta.mostrar_info()

    try:
        print("\nRetiro válido:")
        cuenta.retirar(300)
        cuenta.mostrar_info()
    except (ValueError, FondosInsuficientesException) as e:
        print(f"Error: {e}")

    try:
        print("\nRetiro que supera el saldo:")
        cuenta.retirar(1500)
    except FondosInsuficientesException as e:
        print(f"Error esperado: {e}")

    try:
        print("\nRetiro con monto negativo:")
        cuenta.retirar(-50)
    except ValueError as e:
        print(f"Error esperado: {e}")

def probar_operaciones_completas():
    print("\n=== PRUEBA DE OPERACIONES COMPLETAS ===")
    cuenta = CuentaBancaria("12345", "Juan Pérez", 1000)

    print("Estado inicial:")
    cuenta.mostrar_info()

    operaciones = [
        ("depositar", 200),
        ("retirar", 150),
        ("depositar", -50),
        ("retirar", 1200),
        ("retirar", 0),
        ("depositar", 100),
        ("retirar", 800)
    ]

    for operacion, monto in operaciones:
        print(f"\n--- {operacion.upper()} ${monto:.2f} ---")
        try:
            if operacion == "depositar":
```

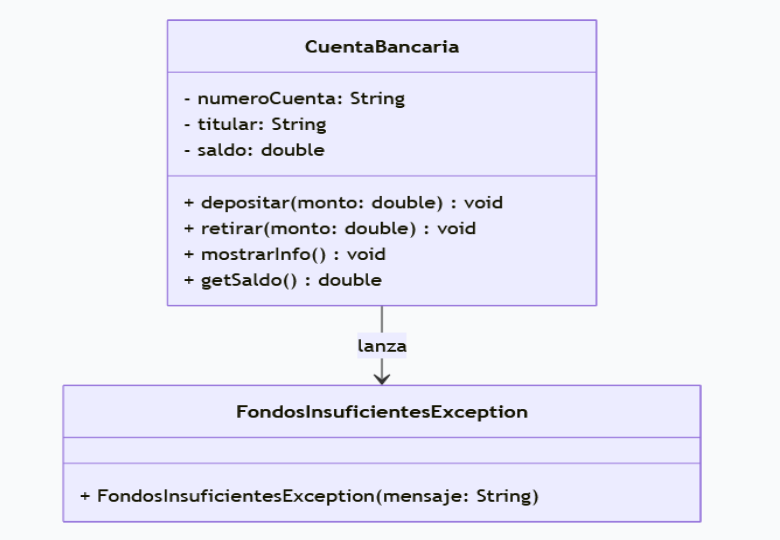
```
            try:
                if operacion == "depositar":
                    cuenta.depositar(monto)
                elif operacion == "retirar":
                    cuenta.retirar(monto)
                cuenta.mostrar_info()
            except (ValueError, FondosInsuficientesException) as e:
                print(f"Error: {e}")

def menu_interactivo():
    print("\n" + "="*50)
    print("=== SISTEMA BANCARIO INTERACTIVO ===")
    print("="*50)

    cuenta = CuentaBancaria("12345", "Juan Pérez", 1000)

    while True:
        print(f"\nSaldo actual: ${cuenta.saldo:.2f}")
        print("\nOpciones:")
        print("1. Depositar")
        print("2. Retirar")
        print("3. Ver información")
        print("4. Salir")

        opcion = input("Seleccione una opción: ")

        if opcion == "1":
            try:
                monto = float(input("Monto a depositar: $"))
                cuenta.depositar(monto)
            except ValueError as e:
                print(f"Error: {e}")

        elif opcion == "2":
            try:
                monto = float(input("Monto a retirar: $"))
                cuenta.retirar(monto)
            except (ValueError, FondosInsuficientesException) as e:
                print(f"Error: {e}")

        elif opcion == "3":
            cuenta.mostrar_info()

        elif opcion == "4":
            print("Saliendo del sistema bancario...")
            break

        else:
            print("Opción no válida")

if __name__ == "__main__":
    print("=== SISTEMA BANCARIO CON MANEJO DE EXCEPCIONES ===\n")

    probar_depositos()
    probar_retiros()
    probar_operaciones_completas()

    menu_interactivo()
```

\

```
PS E:\UMSA\third\121\PracticaAuxiliar2> & C:/Users/heicar_23/AppData/Local/Progr
=== SISTEMA BANCARIO CON MANEJO DE EXCEPCIONES ===

=== PRUEBA DE DEPÓSITOS ===
Cuenta: 12345
Titular: Juan Pérez
Saldo: $1000.00

Depósito válido:
Depósito exitoso: +$500.00
Cuenta: 12345
Titular: Juan Pérez
Saldo: $1500.00

Depósito con monto negativo:
Error esperado: Error: El monto a depositar debe ser positivo

Depósito con monto cero:
Error esperado: Error: El monto a depositar debe ser positivo

=== PRUEBA DE RETIROS ===
Cuenta: 12345
Titular: Juan Pérez
Saldo: $1000.00

Retiro válido:
Retiro exitoso: -$300.00
Cuenta: 12345
Titular: Juan Pérez
Saldo: $700.00

Retiro que supera el saldo:
Error esperado: Fondos insuficientes. Saldo: $700.00, Retiro: $1500.00

Retiro con monto negativo:
Error esperado: Error: El monto a retirar debe ser positivo

=== PRUEBA DE OPERACIONES COMPLETAS ===
Estado inicial:
Cuenta: 12345
Titular: Juan Pérez
Saldo: $1000.00

--- DEPOSITAR $200.00 ---
Depósito exitoso: +$200.00
Cuenta: 12345
Titular: Juan Pérez
Saldo: $1200.00

--- RETIRAR $150.00 ---
Retiro exitoso: -$150.00
Cuenta: 12345
Titular: Juan Pérez
Saldo: $1050.00

--- DEPOSITAR $-50.00 ---
Error: Error: El monto a depositar debe ser positivo

--- RETIRAR $1200.00 ---
Error: Fondos insuficientes. Saldo: $1050.00, Retiro: $1200.00

--- RETIRAR $0.00 ---
```

Ejercicio 6. Sea la clase CuentaBancaria con los siguientes atributos: numeroCuenta (String), titular (String), saldo (double)

a) Cree la excepción personalizada **FondosInsuficientesException**, que se lance cuando se intente retirar un monto mayor al saldo disponible.
b) Implemente los métodos:
   - depositar(double monto) → aumenta el saldo si el monto es positivo, si no, lanza una excepción IllegalArgumentException.
   - retirar(double monto) → descuenta el saldo si hay fondos suficientes, de lo contrario lanza FondosInsuficientesException.
   - mostrarInfo() → imprime en pantalla número de cuenta, titular y saldo.
c) Cree un programa principal que:
   - Genere una cuenta con número "12345", titular "Juan Pérez", y saldo inicial de 1000.
   - Realice un depósito válido y otro con monto negativo para disparar la excepción.
   - Realice un retiro válido y otro que supere el saldo para disparar la excepción personalizada.
   - Use bloques try-catch para capturar y mostrar los errores.




```python
class ProductoNoEncontradoException(Exception):
    def __init__(self, mensaje):
        super().__init__(mensaje)

class StockInsuficienteException(Exception):
    def __init__(self, mensaje):
        super().__init__(mensaje)

class Producto:
    def __init__(self, codigo, nombre, precio, stock):
        self.codigo = codigo
        self.nombre = nombre
        self.precio = precio
        self.stock = stock

    def __str__(self):
        return f"{self.codigo}: {self.nombre} - ${self.precio} - Stock: {self.stock}"
```

```python

class Inventario:
    def __init__(self):
        self.productos = []

    def agregar_producto(self, producto):
        for p in self.productos:
            if p.codigo == producto.codigo:
                raise ValueError(f"Error: El código {producto.codigo} ya existe")

        if producto.precio < 0:
            raise ValueError("Error: El precio no puede ser negativo")

        if producto.stock < 0:
            raise ValueError("Error: El stock no puede ser negativo")

        self.productos.append(producto)
        print(f"Producto {producto.nombre} agregado al inventario")

    def buscar_producto(self, codigo):
        for producto in self.productos:
            if producto.codigo == codigo:
                return producto
        raise ProductoNoEncontradoException(f"Producto con código {codigo} no encontrado")

    def vender_producto(self, codigo, cantidad):
        producto = self.buscar_producto(codigo)

        if producto.stock < cantidad:
            raise StockInsuficienteException(
                f"Stock insuficiente. Disponible: {producto.stock}, Solicitado: {cantidad}"
            )

        producto.stock -= cantidad
        total = producto.precio * cantidad
        print(f"Venta realizada: {cantidad} x {producto.nombre} = ${total}")
        return total

    def mostrar_inventario(self):
        print("\n=== INVENTARIO ACTUAL ===")
        if not self.productos:
            print("No hay productos en el inventario")
        else:
            for producto in self.productos:
                print(producto)

def probar_agregar_productos():
    print("=== PRUEBA AGREGAR PRODUCTOS ===")
    inventario = Inventario()

    try:
        p1 = Producto("P001", "Laptop", 1200.50, 10)
        p2 = Producto("P002", "Mouse", 25.99, 50)
        p3 = Producto("P003", "Teclado", 45.75, 30)

        inventario.agregar_producto(p1)
        inventario.agregar_producto(p2)
```

```
        inventario.agregar_producto(p3)

        inventario.mostrar_inventario()

    except ValueError as e:
        print(f"Error al agregar producto: {e}")

def probar_errores_agregar():
    print("\n=== PRUEBA ERRORES AL AGREGAR ===")
    inventario = Inventario()

    try:
        p1 = Producto("P001", "Producto Normal", 100, 10)
        inventario.agregar_producto(p1)

        p2 = Producto("P001", "Producto Duplicado", 200, 5)
        inventario.agregar_producto(p2)
    except ValueError as e:
        print(f"Error esperado: {e}")

    try:
        p3 = Producto("P003", "Precio Negativo", -50, 10)
        inventario.agregar_producto(p3)
    except ValueError as e:
        print(f"Error esperado: {e}")

    try:
        p4 = Producto("P004", "Stock Negativo", 50, -5)
        inventario.agregar_producto(p4)
    except ValueError as e:
        print(f"Error esperado: {e}")

def probar_buscar_producto():
    print("\n=== PRUEBA BUSCAR PRODUCTO ===")
    inventario = Inventario()

    p1 = Producto("P001", "Monitor", 300, 15)
    inventario.agregar_producto(p1)

    try:
        producto = inventario.buscar_producto("P001")
        print(f"Producto encontrado: {producto}")
    except ProductoNoEncontradoException as e:
        print(f"Error: {e}")

    try:
        inventario.buscar_producto("P999")
    except ProductoNoEncontradoException as e:
        print(f"Error esperado: {e}")

def probar_ventas():
    print("\n=== PRUEBA VENTAS ===")
    inventario = Inventario()

    p1 = Producto("P001", "Tablet", 250, 5)
    inventario.agregar_producto(p1)
```

```python
    try:
        inventario.vender_producto("P001", 3)
        inventario.mostrar_inventario()

        inventario.vender_producto("P001", 5)
    except (ProductoNoEncontradoException, StockInsuficienteException) as e:
        print(f"Error en venta: {e}")

    try:
        inventario.vender_producto("P999", 1)
    except ProductoNoEncontradoException as e:
        print(f"Error esperado: {e}")

def menu_interactivo():
    print("\n" + "="*50)
    print("=== SISTEMA DE INVENTARIO INTERACTIVO ===")
    print("="*50)

    inventario = Inventario()

    while True:
        print("\nOpciones:")
        print("1. Agregar producto")
        print("2. Buscar producto")
        print("3. Vender producto")
        print("4. Mostrar inventario")
        print("5. Salir")

        opcion = input("Seleccione una opción: ")

        if opcion == "1":
            try:
                codigo = input("Código: ")
                nombre = input("Nombre: ")
                precio = float(input("Precio: "))
                stock = int(input("Stock: "))

                producto = Producto(codigo, nombre, precio, stock)
                inventario.agregar_producto(producto)

            except ValueError as e:
                print(f"Error: {e}")

        elif opcion == "2":
            try:
                codigo = input("Código a buscar: ")
                producto = inventario.buscar_producto(codigo)
                print(f"Producto encontrado: {producto}")
            except ProductoNoEncontradoException as e:
                print(f"Error: {e}")

        elif opcion == "3":
            try:
                codigo = input("Código del producto: ")
                cantidad = int(input("Cantidad a vender: "))
                total = inventario.vender_producto(codigo, cantidad)
                print(f"Venta exitosa. Total: ${total}")
```

```
                print(f"Venta exitosa. Total: ${total}")
            except (ProductoNoEncontradoException, StockInsuficienteException, ValueError) as e:
                print(f"Error en venta: {e}")

        elif opcion == "4":
            inventario.mostrar_inventario()

        elif opcion == "5":
            print("Saliendo del sistema...")
            break

        else:
            print("Opción no válida")

if __name__ == "__main__":
    print("=== SISTEMA DE INVENTARIO CON MANEJO DE EXCEPCIONES ===\n")

    probar_agregar_productos()
    probar_errores_agregar()
    probar_buscar_producto()
    probar_ventas()

    menu_interactivo()

    print("\n=== PROGRAMA FINALIZADO ===")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS E:\UMSA\third\121\PracticaAuxiliar2> & C:/Users/heicar_23/AppData/Lo
=== SISTEMA DE INVENTARIO CON MANEJO DE EXCEPCIONES ===

=== PRUEBA AGREGAR PRODUCTOS ===
Producto Laptop agregado al inventario
Producto Mouse agregado al inventario
Producto Teclado agregado al inventario

=== INVENTARIO ACTUAL ===
P001: Laptop - $1200.5 - Stock: 10
P002: Mouse - $25.99 - Stock: 50
P003: Teclado - $45.75 - Stock: 30

=== PRUEBA ERRORES AL AGREGAR ===
Producto Producto Normal agregado al inventario
Error esperado: Error: El código P001 ya existe
Error esperado: Error: El precio no puede ser negativo
Error esperado: Error: El stock no puede ser negativo

=== PRUEBA BUSCAR PRODUCTO ===
Producto Monitor agregado al inventario
Producto encontrado: P001: Monitor - $300 - Stock: 15
Error esperado: Producto con código P999 no encontrado

=== PRUEBA VENTAS ===
Producto Tablet agregado al inventario
Venta realizada: 3 x Tablet = $750

=== INVENTARIO ACTUAL ===
P001: Tablet - $250 - Stock: 2
Error en venta: Stock insuficiente. Disponible: 2, Solicitado: 5
Error esperado: Producto con código P999 no encontrado

==============================================
=== SISTEMA DE INVENTARIO INTERACTIVO ===
==============================================

Opciones:
1. Agregar producto
2. Buscar producto
3. Vender producto
4. Mostrar inventario
5. Salir
Seleccione una opción: 2
Código a buscar: p001
Error: Producto con código p001 no encontrado

Opciones:
1. Agregar producto
2. Buscar producto
3. Vender producto
4. Mostrar inventario
5. Salir
Seleccione una opción:
```