PRACTICA PROGRAMACION

Carlos jesus sotelo pina ICC

1. Sistema de reservas para un cine

Justificación de la necesidad:

Este sistema de reservas para cine está diseñado para gestionar eficientemente las interacciones entre los usuarios, empleados y el cine, asegurando una experiencia fluida tanto para los clientes como para el personal.

Clases:

1. Clase Persona

- **Atributos:**
  - nombre: Nombre de la persona.
  - correo: Correo electrónico de la persona.
- **Métodos:**
  - __init__(self, nombre, correo): Constructor para inicializar el nombre y correo de la persona.
  - registrar(self): Registra la persona en el sistema (añade a la lista de personas registradas).
  - actualizar_datos(self, nombre, correo): Actualiza los datos de la persona.
  - personas_registradas(cls): Método de clase que muestra todas las personas registradas.

2. Clase Usuario (hereda de Persona)

- **Atributos:**
  - historial_reservas: Lista que almacena el historial de reservas del usuario.
- **Métodos:**
  - __init__(self, nombre, correo): Constructor que hereda de Persona y también inicializa el historial de reservas.
  - reservar(self, funcion, asientos): Permite al usuario hacer una reserva de una cantidad de asientos en una función.

- cancelar_reserva(self, funcion): Permite al usuario cancelar una reserva específica.

3. Clase Empleado (hereda de Persona)

- **Atributos:**
  - rol: Rol del empleado (ej. taquillero, administrador, limpieza).
- **Métodos:**
  - __init__(self, nombre, correo, rol): Constructor que hereda de Persona e inicializa el rol del empleado.
  - agregar_funcion(self, pelicula, sala, hora): Permite al empleado agregar una nueva función de una película.
  - agregar_pelicula(self, titulo, genero, duracion): Permite al empleado agregar una nueva película.
  - agregar_promocion(self, descuento, condiciones): Permite al empleado agregar una nueva promoción.
  - modificar_promocion(self, promocion, nuevo_descuento, nuevas_condiciones): Modifica una promoción existente.

4. Clase Espacio (Base para salas y zonas de comida)

- **Atributos:**
  - capacidad: La capacidad máxima del espacio.
  - identificador: El identificador único del espacio (por ejemplo, el nombre de la sala o zona de comida).
- **Métodos:**
  - __init__(self, capacidad, identificador): Constructor que inicializa la capacidad y el identificador.
  - descripcion(self): Muestra una descripción del espacio (capacidad e identificador).

---

5. Clase Sala (hereda de Espacio)

- **Atributos:**
  - tipo: Tipo de sala (2D, 3D, IMAX).

    - disponibilidad: Estado de la sala (si está disponible o no).
- **Métodos:**
    - __init__(self, capacidad, identificador, tipo): Constructor que hereda de Espacio e inicializa los atributos adicionales de tipo y disponibilidad.
    - consultar_disponibilidad(self): Verifica si la sala está disponible para una función.

---

6. Clase ZonaComida (hereda de Espacio)

- **Atributos:**
    - productos: Lista de productos disponibles en la zona de comida.
- **Métodos:**
    - __init__(self, capacidad, identificador): Constructor que hereda de Espacio e inicializa la lista de productos.
    - agregar_producto(self, producto): Agrega un producto a la zona de comida.
    - mostrar_productos(self): Muestra los productos disponibles en la zona de comida.

---

7. Clase Pelicula

- **Atributos:**
    - titulo: Título de la película.
    - genero: Género de la película.
    - duracion: Duración de la película en minutos.
- **Métodos:**
    - __init__(self, titulo, genero, duracion): Constructor para inicializar los atributos de la película.

---

8. Clase Funcion

- **Atributos:**

    - pelicula: Película asociada a la función.
    - sala: Sala donde se proyectará la película.
    - hora: Hora en que la película se proyectará.
    - asientos_disponibles: Número de asientos disponibles para la función.
    - asientos_reservados: Lista de asientos que han sido reservados.
- **Métodos:**
    - __init__(self, pelicula, sala, hora): Constructor para inicializar los atributos de la función.
    - verificar_disponibilidad(self, asientos): Verifica si hay suficientes asientos disponibles para una reserva.
    - reservar_asientos(self, asientos): Reserva una cantidad de asientos en la función.
    - liberar_asientos(self, asientos): Libera una cantidad de asientos previamente reservados.
    - mostrar_asientos_reservados(self): Muestra los asientos que han sido reservados para la función.

---

9. Clase Promocion

- **Atributos:**
    - descuento: Porcentaje de descuento de la promoción.
    - condiciones: Condiciones de la promoción (por ejemplo, "válido de lunes a jueves").
- **Métodos:**
    - __init__(self, descuento, condiciones): Constructor para inicializar el descuento y las condiciones de la promoción.
    - mostrar(self): Muestra la información de la promoción.

---

10. Clase Reserva

- **Atributos:**

    - usuario: Usuario que realiza la reserva.
    - funcion: Función a la que se realiza la reserva.
    - asientos: Número de asientos reservados.
    - promocion: Promoción aplicada, si la hay.
- **Métodos:**
    - __init__(self, usuario, funcion, asientos, promocion=None): Constructor para inicializar los detalles de la reserva.
    - aplicar_descuento(self): Aplica el descuento de la promoción (si existe) al total de la reserva.

---

11. Clase Inventario

- **Atributos:**
    - ingredientes: Diccionario con los ingredientes disponibles y su cantidad (por ejemplo, {'leche': 10, 'azúcar': 5}).
- **Métodos:**
    - __init__(self): Constructor que inicializa el inventario con los ingredientes disponibles.
    - actualizar_inventario(self, ingrediente, cantidad): Actualiza la cantidad de un ingrediente específico.
    - verificar_stock(self, ingrediente, cantidad): Verifica si hay suficiente stock de un ingrediente para preparar el pedido.

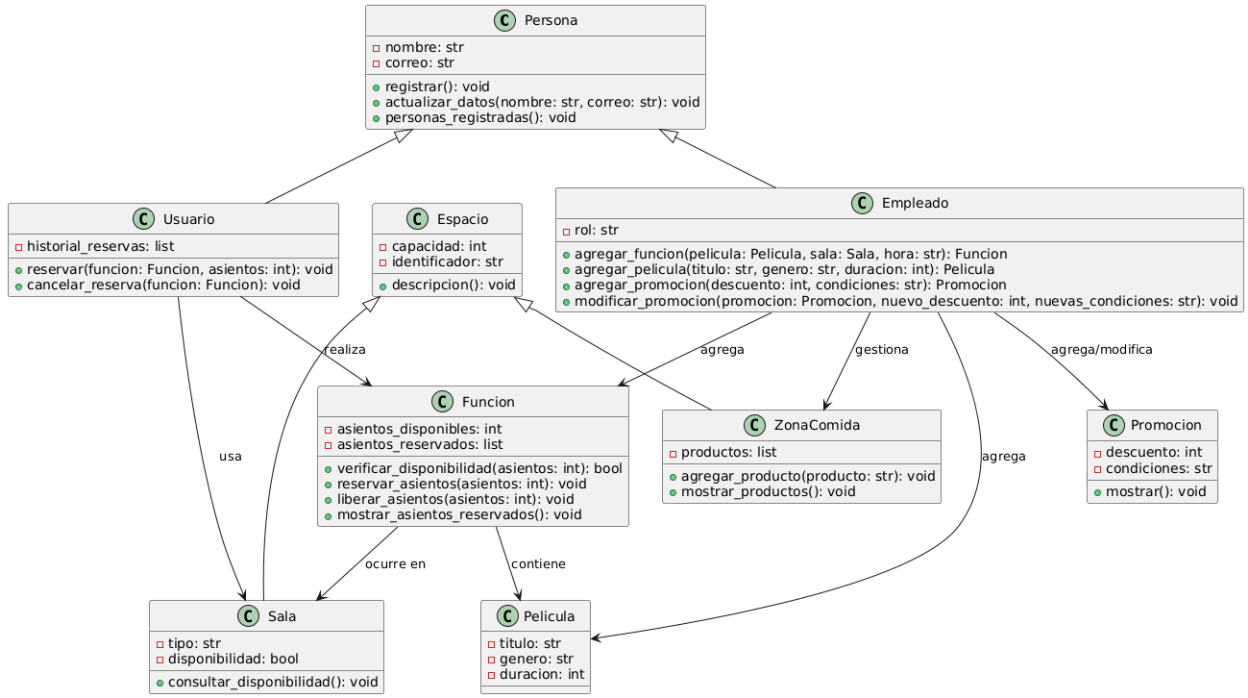
```
La persona Ana Pérez ha sido registrada con el correo ana.perez@email.com
La persona Luis Martínez ha sido registrada con el correo luis.martinez@email.com

--- Sistema de Cine ---
1. Menú de Usuario
2. Menú de Empleado
3. Salir
```

```
--- Menú de Usuario ---
1. Reservar asientos
2. Cancelar reserva
3. Ver historial de reservas
4. Salir
```

Ingrese el título de la película: (Press 'Enter' to confirm or 'Escape' to cancel)

Ingrese el número de asientos a reservar: (Press 'Enter' to confirm or 'Escape' to cancel)



2. Gestión de Pedidos en una Cafetería

El programa de gestión de pedidos en una cafetería es fundamental para optimizar y automatizar diversas operaciones dentro de un negocio de este tipo. Aquí se justifican las principales características y necesidades del sistema basadas en las especificaciones del proyecto:

Clases:

1. Clase Persona (base)

- **Atributos:**
    - nombre: Nombre de la persona.
    - id: Identificador único de la persona (puede ser un código o número).
    - telefono: Número de teléfono de la persona.
- **Métodos:**
    - No tiene métodos específicos en la clase base, ya que es solo una clase común para Cliente y Empleado.

---

2. Clase Cliente (hereda de Persona)

- **Atributos:**
    - historial_pedidos: Lista de objetos Pedido que contienen el historial de pedidos del cliente.
    - puntos_fidelidad: Número de puntos que el cliente ha acumulado para beneficios o promociones.
- **Métodos:**
    - realizar_pedido(productos: List[ProductoPersonalizado], cafeteria: Cafeteria): Permite al cliente realizar un pedido. Verifica el inventario antes de crear el pedido.
    - consultar_historial(): Muestra el historial de pedidos del cliente con detalles como estado, productos y total.

---

3. Clase Empleado (hereda de Persona)

- **Atributos:**
    - rol: El rol del empleado (puede ser MESERO, BARISTA o GERENTE).
- **Métodos:**
    - actualizar_inventario(inventario: Inventario, ingrediente: str, cantidad: int): Permite a los empleados con el rol adecuado (Barista o Gerente) actualizar el inventario.
    - actualizar_estado_pedido(pedido: Pedido, nuevo_estado: EstadoPedido): Permite a los empleados actualizar el estado de un pedido (pendiente, en preparación, entregado).

---

4. Clase ProductoBase (base para productos)

- **Atributos:**
    - nombre: Nombre del producto (por ejemplo, "Café Americano").
    - precio_base: Precio base del producto.
    - descripcion: Descripción del producto.
- **Métodos:**
    - No tiene métodos específicos. Es una clase base para productos como Bebida y Postre.

---

5. Clase Bebida (hereda de ProductoBase)

- **Atributos:**
    - tipo: Tipo de bebida (caliente o fría).
    - ingredientes_base: Diccionario que contiene los ingredientes base (por ejemplo, café, leche, azúcar) y su cantidad.
- **Métodos:**
    - agregar_ingrediente_base(ingrediente: str, cantidad: int): Agrega ingredientes base a la bebida (por ejemplo, cantidad de café o leche).

---

6. Clase Postre (hereda de ProductoBase)

- **Atributos:**
    - es_vegano: Indica si el postre es vegano (True o False).
    - sin_gluten: Indica si el postre es sin gluten (True o False).
    - ingredientes_base: Diccionario con los ingredientes base del postre (por ejemplo, harina, huevos).
- **Métodos:**
    - No tiene métodos adicionales a los definidos en ProductoBase.

---

7. Clase ProductoPersonalizado

- **Atributos:**
    - producto_base: Objeto de tipo ProductoBase (ya sea una bebida o un postre).
    - modificaciones: Diccionario que almacena las modificaciones personalizadas (por ejemplo, "extra leche" o "sin azúcar").
    - precio_final: El precio final del producto después de aplicar las modificaciones.
- **Métodos:**
    - agregar_modificacion(ingrediente: str, cantidad: int): Permite agregar una modificación al producto (por ejemplo, extra leche o sin azúcar).
    - nombre: Propiedad que devuelve el nombre del producto con las modificaciones (si las hay).

---

8. Clase Inventario

- **Atributos:**
    - stock: Diccionario que contiene los ingredientes disponibles en el inventario y su cantidad.
- **Métodos:**

- agregar_ingrediente(ingrediente: str, cantidad: int): Agrega ingredientes al inventario.
- actualizar_cantidad(ingrediente: str, cantidad: int): Actualiza la cantidad de un ingrediente en el inventario (agregar o reducir).
- hay_suficiente_stock(ingrediente: str, cantidad_requerida: int): Verifica si hay suficiente stock de un ingrediente específico para satisfacer un pedido.
- mostrar_stock(): Muestra los ingredientes disponibles y sus cantidades en el inventario.

---

9. Clase Pedido

- **Atributos:**
  - cliente: Objeto Cliente que realiza el pedido.
  - productos: Lista de objetos ProductoPersonalizado que representan los productos seleccionados por el cliente.
  - estado: El estado del pedido (pendiente, en preparación, entregado).
  - fecha_creacion: Fecha y hora en que se crea el pedido.
  - promociones_aplicadas: Lista de objetos Promocion aplicadas al pedido.
  - total: El total calculado del pedido, incluyendo las modificaciones y las promociones.
- **Métodos:**
  - _calcular_total(): Calcula el total del pedido sumando los precios de los productos y aplicando las promociones.
  - actualizar_estado(nuevo_estado: EstadoPedido): Actualiza el estado del pedido.
  - aplicar_promocion(promocion: Promocion): Aplica una promoción al pedido si es aplicable.

---

10. Clase Promocion

- **Atributos:**

- nombre: Nombre de la promoción.
- descuento_porcentaje: Porcentaje de descuento aplicado en la promoción.
- puntos_requeridos: Número de puntos de fidelidad necesarios para aplicar la promoción.
- productos_minimos: Número mínimo de productos que se deben pedir para aplicar la promoción.

- **Métodos:**
  - es_aplicable(pedido: Pedido): Verifica si la promoción es aplicable al pedido según los puntos de fidelidad y los productos mínimos.
  - aplicar_descuento(total: float): Aplica el descuento sobre el total del pedido.

---

11. Clase Cafeteria

- **Atributos:**
  - inventario: Objeto Inventario que maneja los ingredientes.
  - pedidos: Lista de objetos Pedido con todos los pedidos realizados en la cafetería.
  - menu: Diccionario de productos disponibles en el menú (bebidas y postres).
  - promociones: Lista de objetos Promocion disponibles en la cafetería.
- **Métodos:**
  - agregar_al_menu(producto: ProductoBase): Agrega un producto al menú de la cafetería.
  - mostrar_menu(): Muestra los productos disponibles en el menú de la cafetería.
  - agregar_pedido(pedido: Pedido): Agrega un pedido a la lista de pedidos de la cafetería.
  - _actualizar_inventario_por_pedido(pedido: Pedido): Actualiza el inventario después de un pedido.
  - validar_inventario(pedido: Pedido): Verifica si el inventario tiene suficiente stock para completar un pedido.

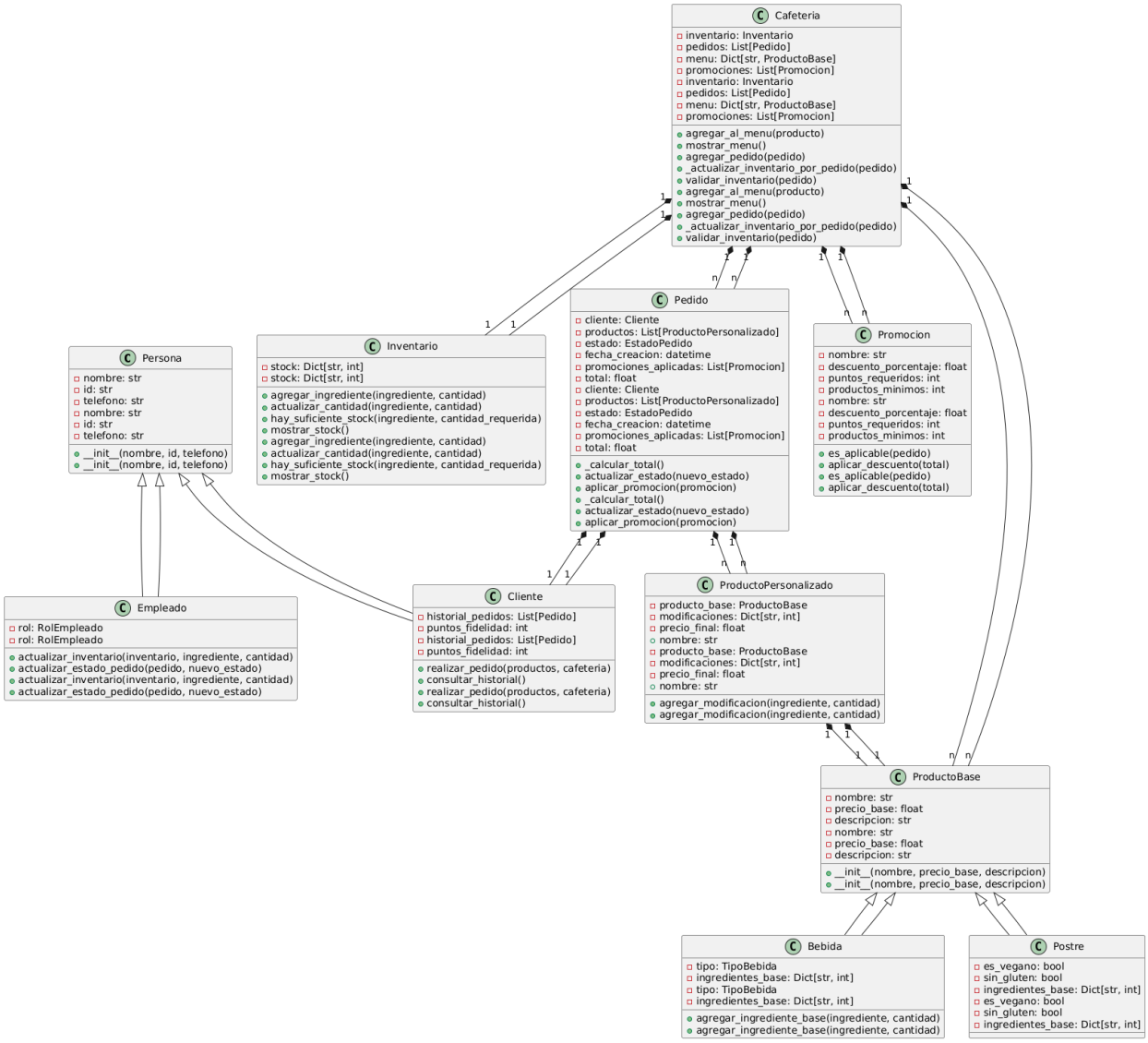
```
=== SISTEMA DE CAFETERÍA ===
1. Realizar pedido
2. Consultar historial de cliente
3. Actualizar estado de pedido (empleados)
4. Ver inventario (empleados)
5. Ver menú
6. Salir
```

```
Menú de la Cafetería:

BEBIDAS:
Café Americano: $2.50
  Café negro tradicional
Café con Leche: $3.00
  Café con leche cremosa
...
  (Vegano)
Brownie Sin Gluten: $3.50
  Brownie de chocolate sin gluten
  (Sin Gluten)
```

```
¿Desea agregar una modificación? (s/n): (Press 'Enter' to confirm or 'Escape' to cancel)
```

```
¡Pedido realizado con éxito!
Total a pagar: $2.50
```



3. Biblioteca Digital

El programa que se presenta cubre una serie de necesidades fundamentales para la operación eficiente de una biblioteca moderna, tomando en cuenta tanto los materiales físicos como digitales, la administración de penalizaciones y la gestión de múltiples sucursales. A continuación se justifica la necesidad de este sistema de acuerdo con las especificaciones proporcionadas.

1. Clase Material (Clase Base)

Atributos:

- **titulo**: El título del material (libro, revista, material digital, etc.).

- **estado**: El estado del material, que puede ser "disponible" o "prestado".

Métodos:

- **__str__**: Retorna una representación en forma de texto del material, mostrando su título y estado.

---

2. Clase Libro (Hereda de Material)

Atributos:

- **autor**: El autor del libro.
- **genero**: El género del libro (por ejemplo, novela, ciencia ficción, historia).

Métodos:

- **__str__**: Retorna una representación en forma de texto del libro, que incluye el autor, el género y el estado.

---

3. Clase Revista (Hereda de Material)

Atributos:

- **edicion**: La edición de la revista (por ejemplo, la número 10).
- **periodicidad**: La periodicidad de la revista (mensual, trimestral, etc.).

Métodos:

- **__str__**: Retorna una representación en texto de la revista, que incluye la edición, la periodicidad y el estado.

---

4. Clase MaterialDigital (Hereda de Material)

Atributos:

- **tipo_archivo**: El tipo de archivo digital del material (por ejemplo, PDF, EPUB).
- **enlace_descarga**: El enlace de descarga del material digital.

Métodos:

- **__str__**: Retorna una representación en texto del material digital, indicando el tipo de archivo y su estado.

---

5. Clase Persona (Clase Base)

Atributos:

- **nombre**: El nombre de la persona.
- **identificacion**: Un número o cadena única que identifica a la persona.

---

6. Clase Usuario (Hereda de Persona)

Atributos:

- **materiales_prestados**: Una lista de los materiales que el usuario tiene prestados.
- **penalizaciones**: Una lista de las penalizaciones acumuladas por el usuario debido a retrasos u otros problemas.

Métodos:

- **devolver_material**: Permite al usuario devolver un material prestado, actualizando su estado y devolviéndolo a la sucursal correspondiente.

---

7. Clase Bibliotecario (Hereda de Persona)

Métodos:

- **agregar_material**: Permite al bibliotecario agregar nuevos materiales al catálogo de una sucursal.
- **gestionar_prestamo**: Gestiona el préstamo de un material a un usuario, actualizando el estado del material y registrando el préstamo.
- **transferir_material**: Permite al bibliotecario transferir un material entre diferentes sucursales de la biblioteca.

---

8. Clase Sucursal

Atributos:

- **nombre**: El nombre de la sucursal.
- **catalogo**: Una lista de los materiales disponibles en la sucursal.

Métodos:

- **agregar_material**: Agrega un nuevo material al catálogo de la sucursal.
- **remover_material**: Elimina un material del catálogo de la sucursal.
- **buscar_disponibles**: Devuelve una lista de los materiales disponibles para préstamo en la sucursal.

---

9. Clase Prestamo

Atributos:

- **usuario**: El usuario que realiza el préstamo.
- **material**: El material que está siendo prestado.
- **fecha_prestamo**: La fecha en que se realizó el préstamo.
- **fecha_devolucion**: La fecha en que se debe devolver el material.

Métodos:

- **calcular_penalizacion**: Calcula la penalización que se debe aplicar si el material no se devuelve en la fecha estipulada.

---

10. Clase Penalizacion

Atributos:

- **usuario**: El usuario al que se le aplica la penalización.
- **monto**: El monto de la penalización por devolución tardía.

Métodos:

- **aplicar_penalizacion**: Aplica la penalización al usuario, añadiéndola a su historial de penalizaciones.

---

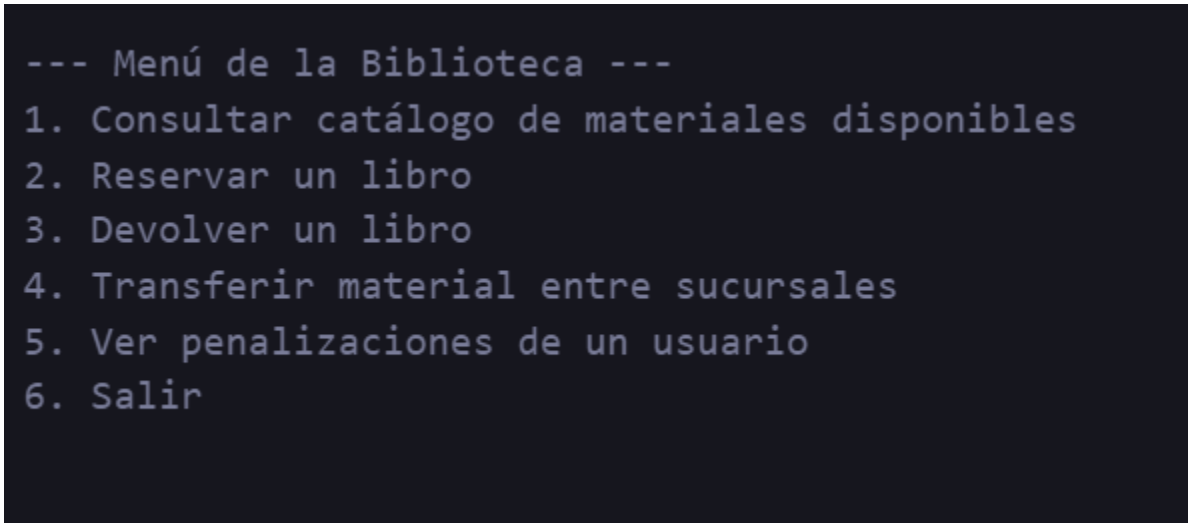
11. Clase Catalogo

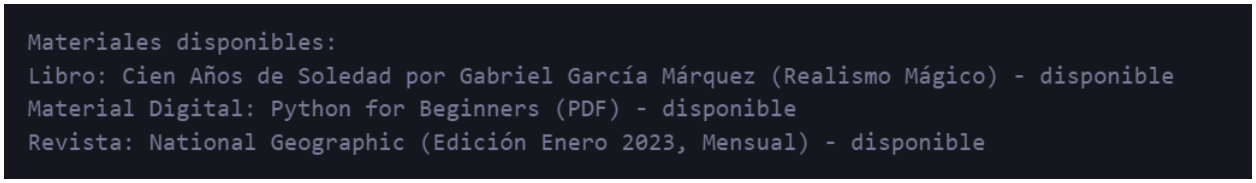
Atributos:

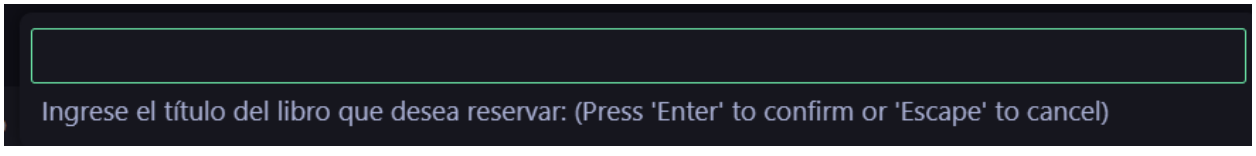
- **sucursales**: Una lista de las sucursales donde se encuentran los materiales de la biblioteca.

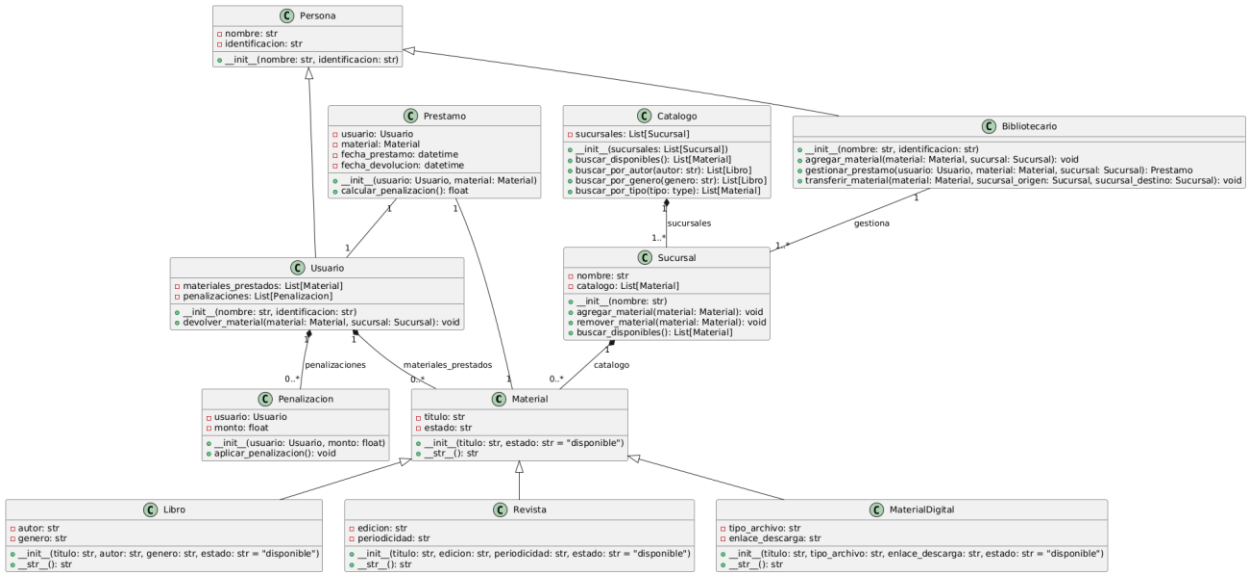
Métodos:

- **buscar_disponibles**: Devuelve todos los materiales disponibles en todas las sucursales.
- **buscar_por_autor**: Devuelve todos los libros de un autor específico.
- **buscar_por_genero**: Devuelve todos los libros de un género específico.
- **buscar_por_tipo**: Devuelve todos los materiales de un tipo específico (libros, revistas, materiales digitales).









Link de github:
https://github.com/lakecg2/practica_progavan