BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN

Periodo Primavera 2025

PROGRAMACIÓN AVANZADA

FCCA 020

Profesor: Jaime Alejandro Romero Sierra

ACTIVIDAD

PRÁCTICA UNIDAD 1

Fecha de entrega 12 de febrero 2025

Nombre	Matricula
Ortiz García Luz Marisol	202450809

SISTEMA DE RESERVAS PARA UN CINE

Justificación de la necesidad del programa

En la industria del entretenimiento, la administración eficiente de funciones de cine, reservas y venta de productos en zonas de comida es fundamental para garantizar una experiencia óptima para los clientes y mejorar la gestión operativa del negocio. Sin un sistema automatizado, la administración de reservas, el control de disponibilidad de asientos, la aplicación de promociones y la gestión de ventas pueden volverse procesos lentos, propensos a errores y difíciles de supervisar en tiempo real.

Este programa ha sido diseñado para optimizar la gestión de funciones de cine y mejorar la experiencia del usuario, permitiendo a los clientes realizar y administrar sus reservas de manera eficiente. Además, automatiza el control de disponibilidad de asientos, asegurando que los espacios en las salas se asignen correctamente y evitando problemas de sobreventa o falta de control en las reservas.

Por otro lado, el programa también ofrece herramientas para la administración de promociones y descuentos, lo que incentiva la fidelización de los clientes mediante descuentos personalizados y condiciones especiales para ciertos tipos de funciones o usuarios frecuentes. Asimismo, permite a los empleados, según su rol, gestionar funciones, modificar promociones y supervisar el correcto funcionamiento de la taquilla y las zonas de comida.

La integración de un módulo de venta de productos en la zona de comida permite mejorar la administración de los productos disponibles, proporcionando un sistema ágil para la venta de alimentos y bebidas, aumentando así la eficiencia en el servicio y mejorando la rentabilidad del negocio.

En conclusión, este sistema automatiza y optimiza la operación de un cine, permitiendo una administración más eficiente de reservas, funciones, promociones y ventas en la zona de comida. Su implementación no solo reduce la carga de trabajo del personal, sino que también mejora la experiencia del cliente, ofreciendo un servicio más ágil, preciso y organizado.

Clases a ocupar describiendo atributos y métodos necesarios

1. Clase **Persona** (Clase base)
   - Atributos
     - nombre (str): Almacena el nombre de la persona.
     - edad (str): Guarda la edad de la persona.
     - correo (int): Contiene la dirección de correo electrónico de la persona.
     - lista (list): Lista estática que almacena todas las instancias de Persona registradas.

   - Métodos
     - __init__(self, nombre, edad, correo): Constructor de la clase que inicializa el nombre, la edad y el correo de la persona.
     - registrar(self): Agrega la instancia de la persona a la lista de personas registradas e imprime un mensaje de confirmación.
     - @classmethod personas_resgistradas(cls): Método de clase que muestra una lista de todas las personas registradas.
2. Clase **Usuario** (Hereda de la clase **Persona**)
   - Atributos
     - historial_reservas (list): Lista que almacena las reservas hechas por el usuario. Cada reserva contiene la función, el número de asientos y el precio total.
   - Métodos
     - hacer_reserva(self, función, asientos, promocion = None):
       - ✓ Verifica si hay asientos disponibles en la función.
       - ✓ Aplica un descuento si hay una promoción.
       - ✓ Calcula el precio total de la reserva.
       - ✓ Resta los asientos reservados de la función.
       - ✓ Agrega la reserva al historial del usuario.
       - ✓ Imprime los detalles de la reserva.
     - cancelar_reserva(self, funcion): Cancela una reserva existente
       - ✓ Busca si el usuario tiene una reserva para la función.
       - ✓ Si la encuentra, devuelve los asientos cancelados a la función.
       - ✓ Elimina la reserva del historial.
       - ✓ Muestra un mensaje de confirmación o error.

3. Clase **Empleado** (Hereda de la clase **Persona**)
   - Atributos
     - rol (str): Rol del empleado (Ej. “Taquillero”, “Administrador”, “limpieza”).
   - Métodos
     - agregar_funcion(self, funcion): Agrega una función a la cartelera y muestra un mensaje indicando que una nueva función ha sido agregada.

- modificar_promocion(self, promoción, nuevo_descuento, nuevas_condiciones):
    - ✓ Verifica si el empleado tiene los permisos necesarios (solo "Taquillero" o "Administrador").
    - ✓ Modifica el descuento y las condiciones de la promoción.
    - ✓ Imprime un mensaje de éxito o error.

4. Clase **Espacio** (Clase Base)
   - Atributos
     - nombre (str): Nombre del espacio (Ej. “Sala 1”, “Zona de Comida”).
     - capacidad (int): Capacidad del espacio.

5. Clase **Sala** (Hereda de la clase **Espacio**)
   - Atributos
     - tipo_sala (str): Tipo de sala (Ej. “2D”, “3D”, “IMAX”).
     - asientos_disponibles (int): Cantidad de asientos que quedan disponibles en la sala.
   - Métodos
     - mostrar_info(self): Imprime la información de la sala, incluyendo su tipo y capacidad.

6. Clase **ZonaComida** (Hereda de la clase **Espacio**)
   - Atributos
     - Productos (dict): Diccionario con los productos disponibles y sus precios (Ejemplo: {"Palomitas": 50, "Refresco": 30}).
   - Métodos
     - mostrar_menu(self): Muestra todos los productos disponibles con sus precios.
     - Vender_producto(self, producto, cantidad): Procesa la venta de un producto.
       - ✓ Verifica si el producto está disponible.
       - ✓ Calcula el total a pagar.
       - ✓ Muestra un mensaje de venta exitosa o error.

7. Clase **Pelicula**
   - Atributos
     - titulo (str): Titulo de la película.
     - genero (str): Genero de la película (Ej. "Acción", "Drama", “Terror”)
     - duración (int): Duración de la película en minutos.
     - clasificación (str): Clasificación de la película (Ej. "PG-13", "R").

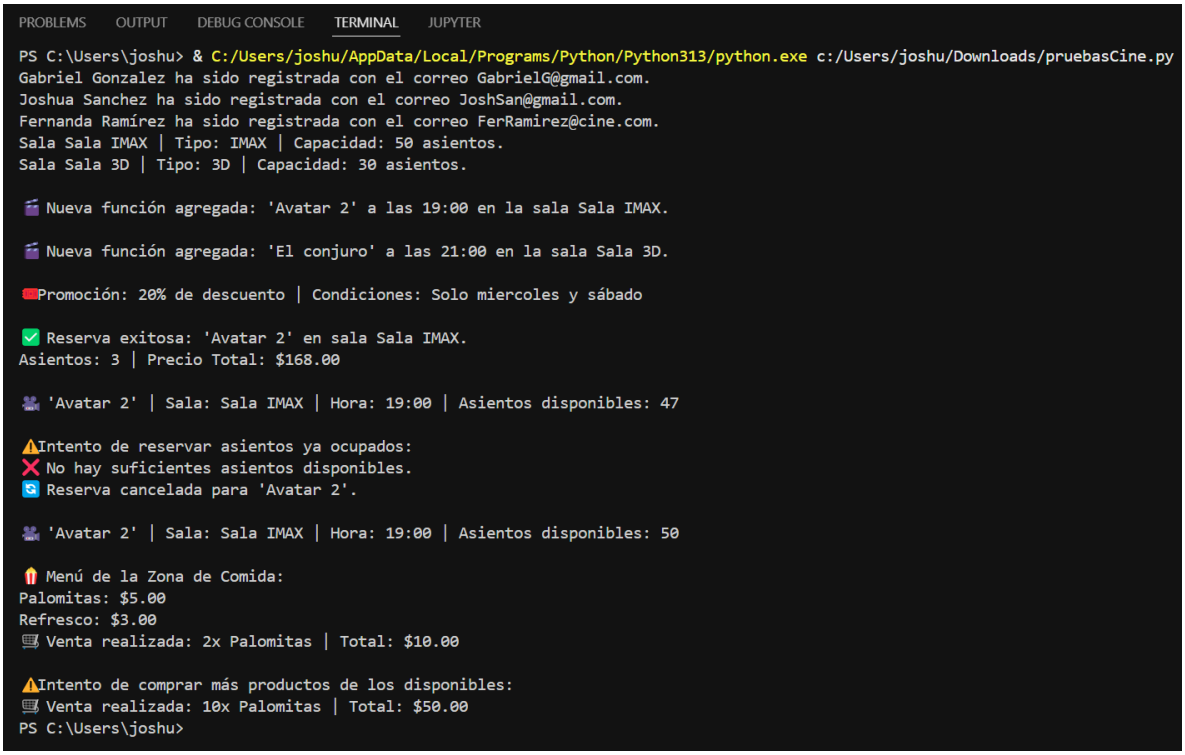
8. Clase **Funcion**
   - Atributos

- `película` (Película): Película que se va a proyectar.
- `sala` (Sala): Sala donde se proyectará la película.
- `hora` (str): hola de la función.
- `asientos_disponibles` (int): Número de asientos disponibles.

- Métodos
  - `mostrar_funcion(self)`: Muestra los detalles de la función, incluyendo la película, sala, hora y asientos disponibles.

9. Clase **`Promocion`**
   - Atributos
     - `descuento` (int): Porcentaje de descuento aplicado (Ejemplo: 20 significa 20% de descuento).
     - `condiciones` (str): Descripción de las condiciones para aplicar el descuento.
   - Métodos
     - `mostrar` (self): Muestra el porcentaje de descuento y las condiciones de la promoción.

Capturas de pantalla del código ejecutado



GESTÓN DE PEDIDOS EN UNA CAFETERIA

Justificación de la necesidad del programa

En la actualidad, la eficiencia y la automatización son factores clave en la administración de negocios del sector alimenticio, especialmente en cafeterías y restaurantes. La gestión manual de pedidos, inventario y promociones puede ser propensa a errores, generar demoras en la atención al cliente y dificultar el control de los recursos.

Este sistema ha sido desarrollado con el objetivo de optimizar y agilizar la gestión de pedidos en una cafetería, permitiendo a los clientes realizar sus pedidos de manera eficiente y recibir un seguimiento automatizado del proceso de preparación y entrega. Al mismo tiempo, el sistema facilita a los empleados la actualización del inventario en tiempo real, asegurando que siempre haya disponibilidad de productos y reduciendo desperdicios por falta de control.

Además, la implementación de un historial de pedidos y un sistema de fidelización mejora la experiencia del cliente, permitiendo la aplicación automática de descuentos a consumidores frecuentes. También se incorporan promociones y personalizaciones de productos, ofreciendo un servicio más personalizado y atractivo.

Por otro lado, la funcionalidad de gestión de inventario permite garantizar el abastecimiento de ingredientes, evitando la venta de productos sin stock y mejorando la planificación de compras. Esto no solo optimiza los costos operativos, sino que también garantiza un servicio más confiable para los clientes.

Este programa automatiza y mejora la eficiencia operativa de una cafetería, reduciendo errores humanos, agilizando la atención al cliente y optimizando el control de pedidos, inventario y promociones. Su implementación representa un paso hacia la digitalización del negocio, asegurando una administración más organizada y efectiva.

Clases a ocupar describiendo atributos y métodos necesarios

1. Clase **Persona** (Clase Base)
   Representa a una persona dentro del sistema (puede ser un cliente o un empleado).
   - Atributos
     - `nombre` (str): Nombre de la persona.
     - `identificacion` (str): Identificación única de la persona.
   - Métodos
     - `__init__(self, nombre, identificacion)`: Constructor que inicializa los atributos nombre e identificación.

2. Clase **Cliente** (Hereda de la Clase **Persona**)
   Representa a un cliente que puede realizar pedidos y consultar su historial.

- Atributos
  - `historial_pedidos` (list): Lista que almacena los pedidos entregados del cliente.
  - `pedidos_realizados` (int): Contador del número de pedidos realizados por el cliente.
- Métodos
  - `realizar_pedido(self, pedido)`:
    - ✓ Simula el proceso de cambio de estado del pedido hasta que sea "ENTREGADO".
    - ✓ Agrega el pedido al historial.
    - ✓ Incrementa el contador de pedidos.
  - `consultar_historial(self)`:
    - ✓ Muestra los pedidos entregados del cliente.
    - ✓ Si no tiene pedidos, muestra un mensaje de alerta.
  - `aplicar_descuento_fidelidad(self, pedido)`: Si el cliente ha realizado al menos un pedido, aplica un 10% de descuento.

3. Clase **Empleado** (Hereda de la Clase **Persona**)
   Representa a un empleado del sistema, que puede actualizar el inventario.
   - Atributos
     - `rol` (str): Indica el rol del empleado (Ejemplo: "Barista", "Cajero").
   - Métodos
     - `actualizar_inventario(self, inventario, ingrediente, cantidad)`: Permite modificar la cantidad de un ingrediente en el inventario.

4. Clase **ProductoBase** (Clase Base)
   Representar cualquier producto (bebida o postre).
   - Atributos
     - `nombre` (str): Nombre del producto.
     - `Precio` (float): Precio del producto.
   - Métodos
     - `__init__(self, nombre, precio)`: Constructor que inicializa los atributos.

5. Clase **Bebida** (Hereda de **ProductoBase**)
   Representa una bebida dentro del sistema, con la posibilidad de personalización.
   - Atributos
     - `tamaño` (str): Tamaño de la bebida (Ejemplo: "Grande", "Mediano").
     - `ingredientes_requeridos` (dict): Ingredientes y cantidades necesarias para preparar la bebida.
     - `personalizables` (list): Lista de personalizaciones aplicadas a la bebida.
     - `PERSONALIZACIONES_VALIDAS` (set): Conjunto de personalizaciones permitidas.

- Métodos
    - `agregar_personalizaciones(self, personalizacion)`: Agrega una personalización válida a la bebida.

6. Clase **`Postre`** (Hereda de **`ProductoBase`**)
   Representa un postre con información sobre restricciones alimenticias.
   - Atributos
     - `vegano` (bool): Indica si el postre es apto para veganos.
     - `sin_gluten` (bool): Indica si el postre no contiene gluten.

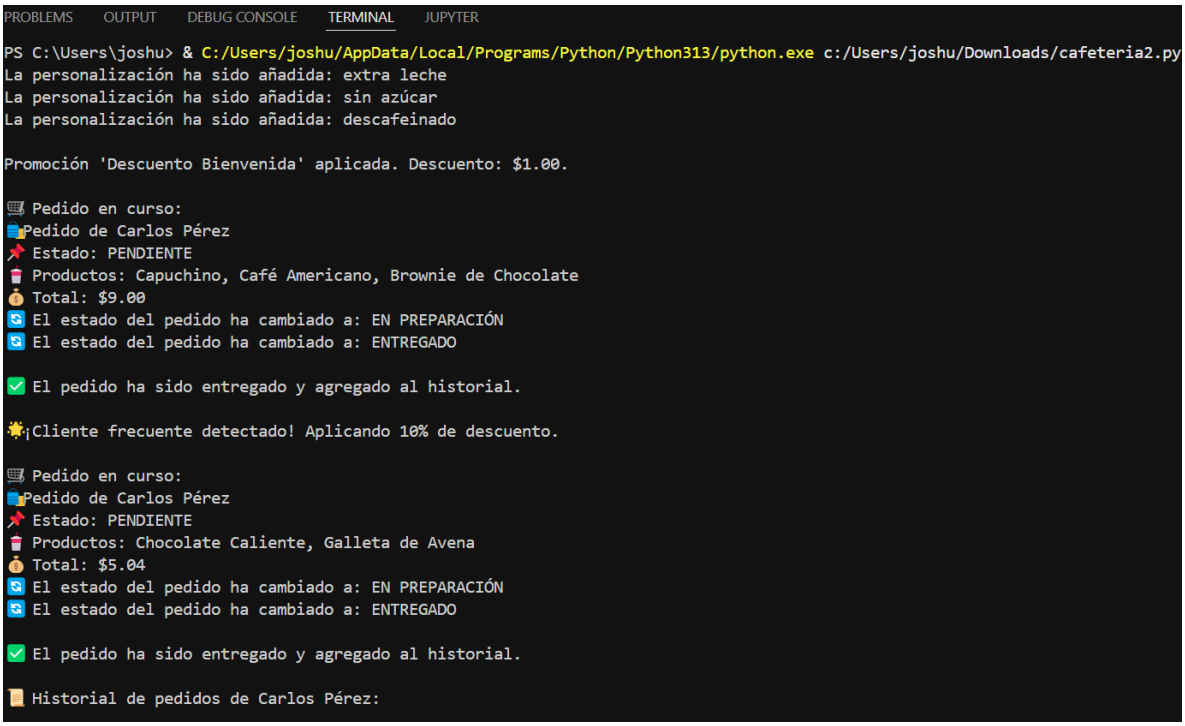
7. Clase **`Inventario`**
   La clase encargada de manejar los ingredientes disponibles para preparar bebidas.
   - Atributos
     - `stock` (dict): Diccionario que almacena los ingredientes y sus cantidades.
   - Métodos
     - `agregar_ingrediente(self, ingrediente, cantidad)`: Añade un ingrediente al inventario.
     - `verificar_stock(self, ingrediente, cantidad)`: Comprueba si hay suficiente cantidad de un ingrediente en el inventario.
     - `actualizar_stock(self, ingrediente, cantidad)`: Modifica la cantidad de un ingrediente en el inventario.
     - `consumir_ingredientes(self, ingredientes_requeridos)`: Reduce los ingredientes del inventario según los requeridos por una bebida.

8. Clase **`Pedido`**
   Representa un pedido que puede contener múltiples productos.
   - Atributos
     - `cliente` (Cliente): Cliente que realiza el pedido.
     - `inventario` (Inventario): Referencia al inventario del sistema.
     - `productos` (list): Lista de productos en el pedido.
     - `subtotal` (float): Precio total antes de aplicar descuentos.
     - `total` (float): Precio final después de aplicar descuentos.
     - `descuento_aplicado` (float): Descuento total aplicado.
     - `estado` (str): Estado actual del pedido (Ejemplo: "PENDIENTE", "EN PREPARACIÓN", "ENTREGADO").
     - `ESTADOS` (list): Lista con los estados posibles del pedido.
   - Métodos
     - `__str__(self)`: Devuelve un resumen del pedido.
     - `agregar_producto_con_inventario(self, producto)`: Agrega un producto al pedido si hay suficientes ingredientes en el inventario.

- calcular_total(self): Calcula el total a pagar restando los descuentos.
- cambiar_estado(self): Cambia el estado del pedido al siguiente en la lista ESTADOS.

9. Clase **Promocion**
   Representa una promoción que aplica descuentos a los pedidos.
   - Atributos
     - nombre (str): Nombre de la promoción.
     - descuento (int): Porcentaje de descuento.
   - Métodos
     - aplicar_descuento(self, pedido): Aplica el descuento al pedido y recalcula el total.

Capturas de pantalla del código ejecutado



```
📦 Pedido 1:
    🏷️Estado: ✅ ENTREGADO
    🥤 Productos:
    - 🧉 Capuchino (Mediano) - $4.0
      ✨ Personalización: extra leche, sin azúcar
    - 🧉 Café Americano (Grande) - $3.5
      ✨ Personalización: descafeinado
    - 🍰 Brownie de Chocolate - $2.5

    💵 Subtotal: $10.00
    🎁 Descuento aplicado: $1.00
    Total final: $9.00

📦 Pedido 2:
    🏷️Estado: ✅ ENTREGADO
    🥤 Productos:
    - 🧉 Chocolate Caliente (Grande) - $3.8
      ✨ Personalización: Sin personalización
    - 🍰 Galleta de Avena - $1.8

    💵 Subtotal: $5.60
    🎁 Descuento aplicado: $0.56
    Total final: $5.04
Inventario actualizado por María López (Barista). Café: 5
Inventario actualizado por María López (Barista). Leche: 5

Intentando agregar Capuchino al pedido...

Intentando agregar Latte al pedido...

❌ Pedido rechazado: No hay suficientes ingredientes para Latte.

🛒 Pedido en curso:
🟦Pedido de Luz Marisol
📌 Estado: PENDIENTE
🥤 Productos: Capuchino
💰 Total: $4.00
🔄 El estado del pedido ha cambiado a: EN PREPARACIÓN
🔄 El estado del pedido ha cambiado a: ENTREGADO

✅ El pedido ha sido entregado y agregado al historial.

📜 Historial de pedidos de Luz Marisol:

📦 Pedido 1:
    🏷️Estado: ✅ ENTREGADO
    🥤 Productos:
    - 🧉 Capuchino (Mediano) - $4.0
      ✨ Personalización: Sin personalización

    💵 Subtotal: $4.00
    🎁 Descuento aplicado: $0.00
    Total final: $4.00
PS C:\Users\joshu>
```

BIBLIOTECA DIGITAL

Justificación de la necesidad del programa

Las bibliotecas han sido fundamentales para la educación y el acceso a la información. Sin embargo, la gestión manual de préstamos, devoluciones y catalogación de materiales suele generar problemas como:

- **Pérdida de registros**: En bibliotecas que dependen de registros físicos, es común que se extravíen datos de préstamos, devoluciones o disponibilidad de material.
- **Dificultad en la consulta de materiales**: Sin un sistema eficiente, los usuarios deben buscar manualmente en listas impresas o consultar con un bibliotecario, lo que lo hace tedioso y conlleva mucho tiempo.
- **Retrasos en devoluciones**: Sin un control adecuado, los usuarios pueden olvidar devolver materiales, generando acumulación de préstamos y provocando la falta de disponibilidad del material para el uso de los demás.
- **Falta de control sobre penalizaciones**: Sin un sistema automatizado, es complicado aplicar sanciones a usuarios que devuelven libros fuera del plazo.
- **Gestión ineficiente del inventario**: La actualización del catálogo y la transferencia de materiales entre sucursales pueden ser procesos tediosos y propensos a errores.

El programa desarrollado busca automatizar y optimizar la gestión bibliotecaria, permitiendo un control más eficiente sobre los materiales y los préstamos. Su implementación facilita una administración más organizada, accesible y eficaz para todos los involucrados.

A través de la digitalización, se optimizan los procesos de préstamo, consulta y devolución de materiales, lo que garantiza un mejor servicio a la comunidad. Además, este sistema contribuye significativamente a la modernización de las bibliotecas, mejorando la eficiencia operativa y ofreciendo una experiencia más ágil tanto para los bibliotecarios como para los usuarios.

La automatización del control de préstamos y penalizaciones fomenta el uso responsable de los materiales y ayuda a reducir la pérdida de libros y revistas, asegurando así una gestión más sostenible y efectiva.

Las principales ventajas son:

- Automatización de préstamos y devoluciones
  - Registro automático del estado de los materiales (DISPONIBLE, PRESTADO).

- Notificación y control sobre la fecha de devolución esperada.
- Aplicación automática de penalizaciones por retrasos en la devolución.

- Facilidad en la consulta de catálogo
  - Los usuarios pueden acceder al catálogo y verificar la disponibilidad de materiales en tiempo real.
  - Búsqueda por título o género en múltiples sucursales.

- Gestión eficiente del inventario
  - Agregar, eliminar y transferir materiales entre sucursales de manera estructurada.
  - Control sobre el estado de los materiales para evitar pérdidas o desorganización.

- Mejor experiencia para usuarios y bibliotecarios
  - Los usuarios pueden encontrar y solicitar materiales de forma rápida y sencilla.
  - Los bibliotecarios pueden optimizar su trabajo y enfocarse en brindar un mejor servicio.

Clases a ocupar describiendo atributos y métodos necesarios

1. Clase **Material** (Clase Base)
   Representa cualquier material dentro de la biblioteca.
   - Atributos
     - `titulo` (str): Nombre del material.
     - `estado` (str): Estado del material, por defecto es "DISPONIBLE".

2. Clase **Libro** (Hereda de la Clase **Material**)
   Representa un libro en la biblioteca.
   - Atributos
     - `titulo` (str): Nombre del linbro.
     - `autor` (str): Autor del libro.
     - `genero` (str): Género literario del libro.
     - `estado` (str): Estado del libro (heredado de Material).

3. Clase **Revista** (Hereda de la Clase **Material**)
   Representa una revista dentro de la biblioteca.
   - Atributos
     - `titulo` (str): Nombre de la revista.
     - `edicion` (str): Número de edición de la revista.

    - `periodidad` (str): Periodicidad de la publicación (mensual, semanal, etc.).
    - `estado` (str): Estado de la revista (heredado de Material).
4. Clase **MaterialDigital** (Hereda de la Clase **Material**)
   Representa un material digital en la biblioteca.
   - Atributos
     - `titulo` (str): Nombre del material digital.
     - `tipo_archivo` (str): Tipo de archivo (PDF, EPUB, MP3, etc.).
     - `enlace_descarga` (str): URL o enlace de descarga del material.
     - `estado` (str): Estado del material (heredado de Material, siempre "DISPONIBLE").

5. Clase **Persona** (Clase Base)
   Representa a cualquier persona en el sistema.
   - Atributos
     - `nombre` (str): Nombre de la persona.

6. Clase **Usuario** (Hereda de la Clase **Persona**)
   Representa a un usuario de la biblioteca que puede pedir prestados materiales.
   - Atributos
     - `consultar_catalogo`(catalogo): Muestra el catálogo de materiales disponibles.
     - `penalizaciones` (int): Contador de penalizaciones del usuario.
   - Métodos
     - `aplicar_descuento(self, pedido)`: Aplica el descuento al pedido y recalcula el total.
     - `devolver_material(material)`: Permite al usuario devolver un material prestado. Si hay retraso en la devolución, se aplica una penalización.

7. Clase **Bibliotecario** (Hereda de la Clase **Persona**)
   Representa al bibliotecario que gestiona préstamos, materiales y transferencias entre sucursales.
   - Métodos
     - `agregar_material(sucursal, material)`: Agrega un material al catálogo de una sucursal.
     - `gestion_prestamo(usuario, material, sucursal)`: Gestiona el préstamo de materiales a los usuarios.
     - `transferir_material(material, sucursal_origen, sucursal_final)`: Transfiere materiales entre sucursales.

8. Clase **Sucursal**
   Representa una sucursal de la biblioteca.
   - Atributos

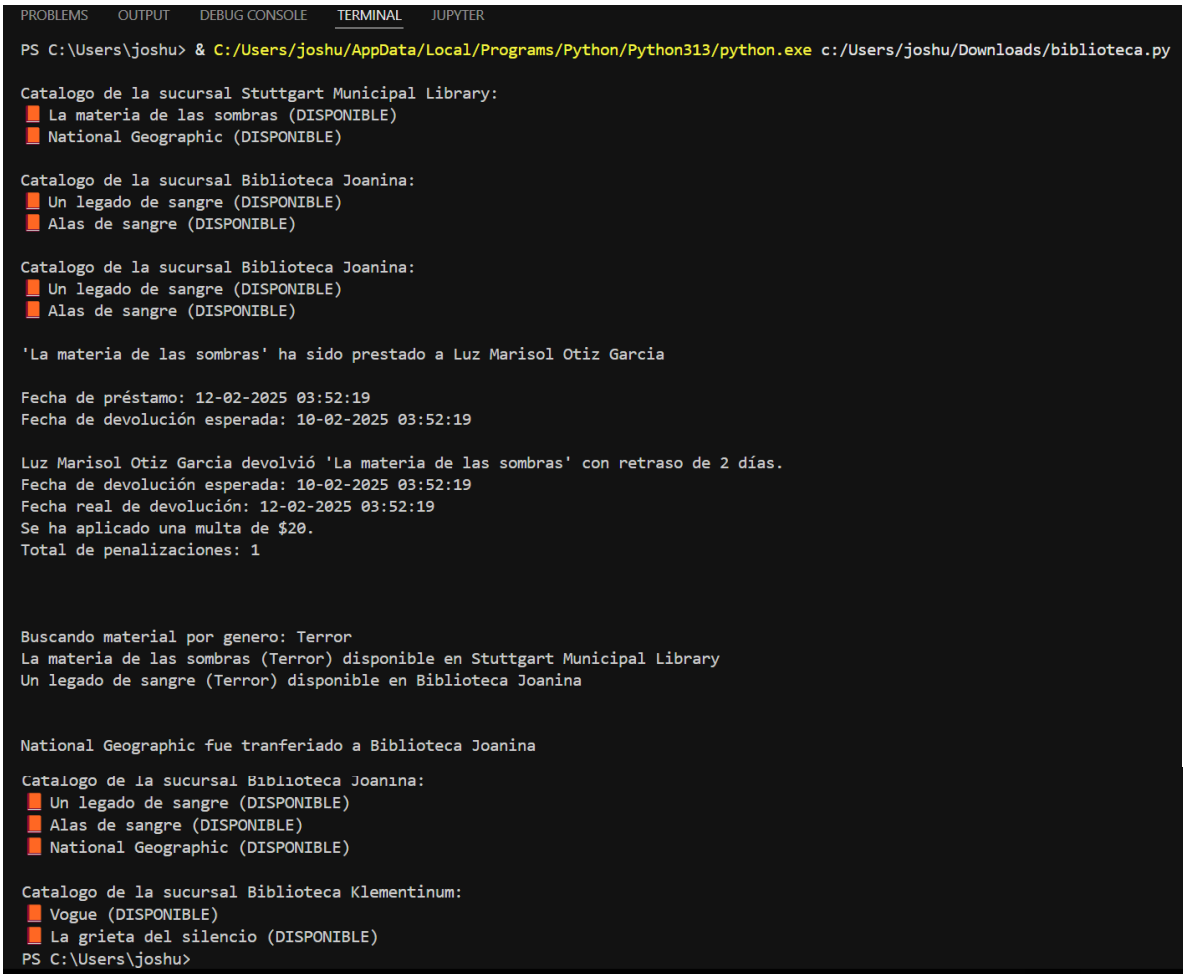
    - `nombre` (str): Nombre de la sucursal.
    - `catalogo` (list): Lista de materiales disponibles en la sucursal.
- Métodos
    - `agregar_material(material)`: Agrega un material a la sucursal.
    - `mostrarCatalogo()`: Muestra el catálogo de la sucursal.

9. Clase **`Prestamo`**
   Representa un préstamo de material a un usuario.
   - Atributos
     - `usuario` (Usuario): Usuario que pidió prestado el material.
     - `material` (Material): Material prestado.
     - `fecha_prestamo` (datetime): Fecha en la que se prestó el material.
     - `fecha_devolucion` (datetime): Fecha esperada de devolución del material.

10. Clase **`Penalizacion`**
    Gestiona las penalizaciones por retrasos en las devoluciones.
    - Atributos
      - `COSTO_POR_DIA` (int): Costo de penalización por día de retraso (10 unidades monetarias).
    - Métodos
      - `aplicar_penalizacion(usuario, prestamo)`: Calcula la penalización basada en los días de retraso y la aplica al usuario.

11. Clase **`Catalogo`**
    Gestiona las penalizaciones por retrasos en las devoluciones.
    - Métodos
      - `buscar_material(sucursales, criterio, tipo_busqueda)`: Busca materiales en diferentes sucursales por título o género.

Capturas de pantalla del código ejecutado



Link del Github de los códigos

https://github.com/LuzOrtG/Practicauni1.git

Diagrama UML "Resevas para un Cine"

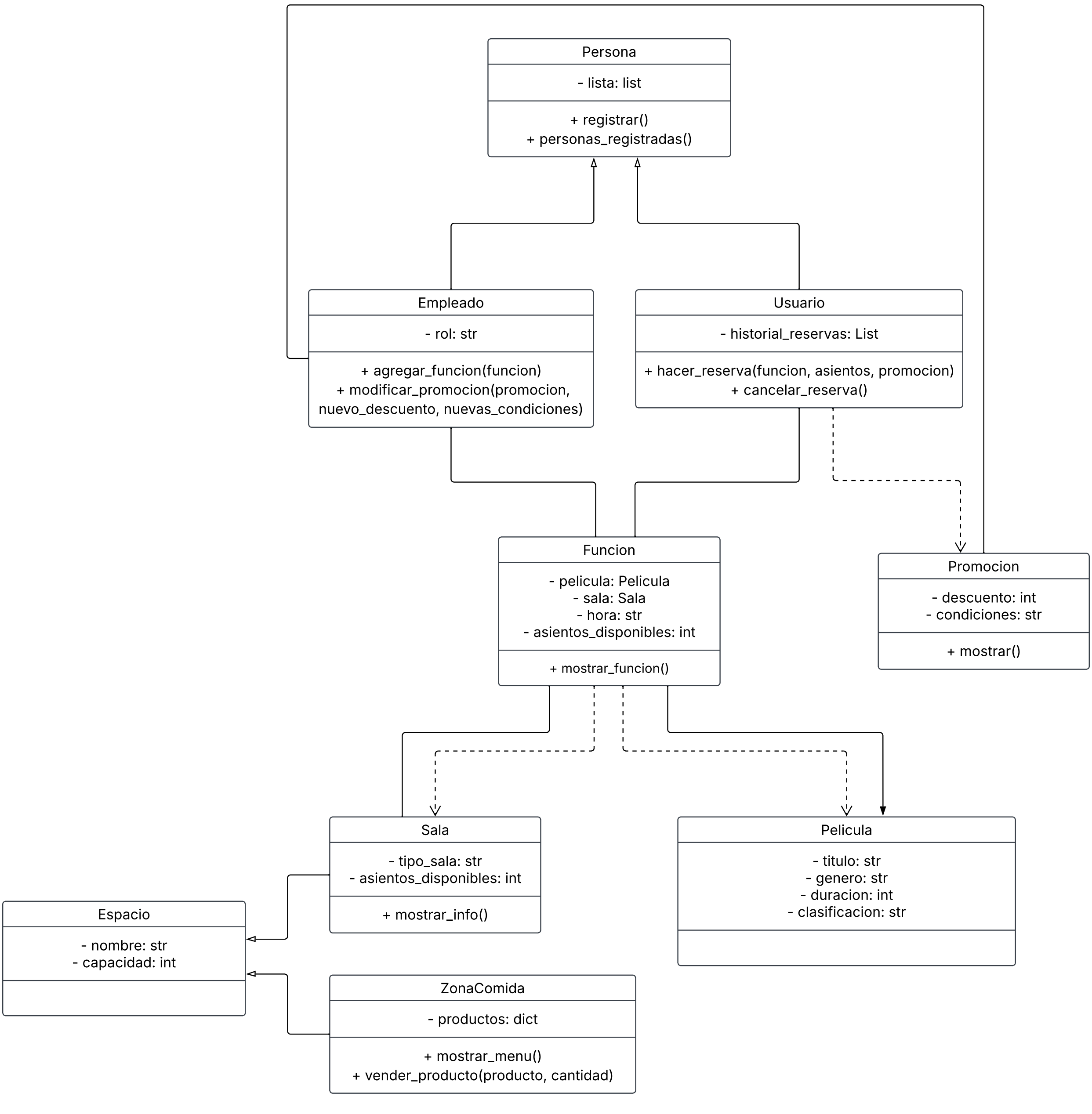
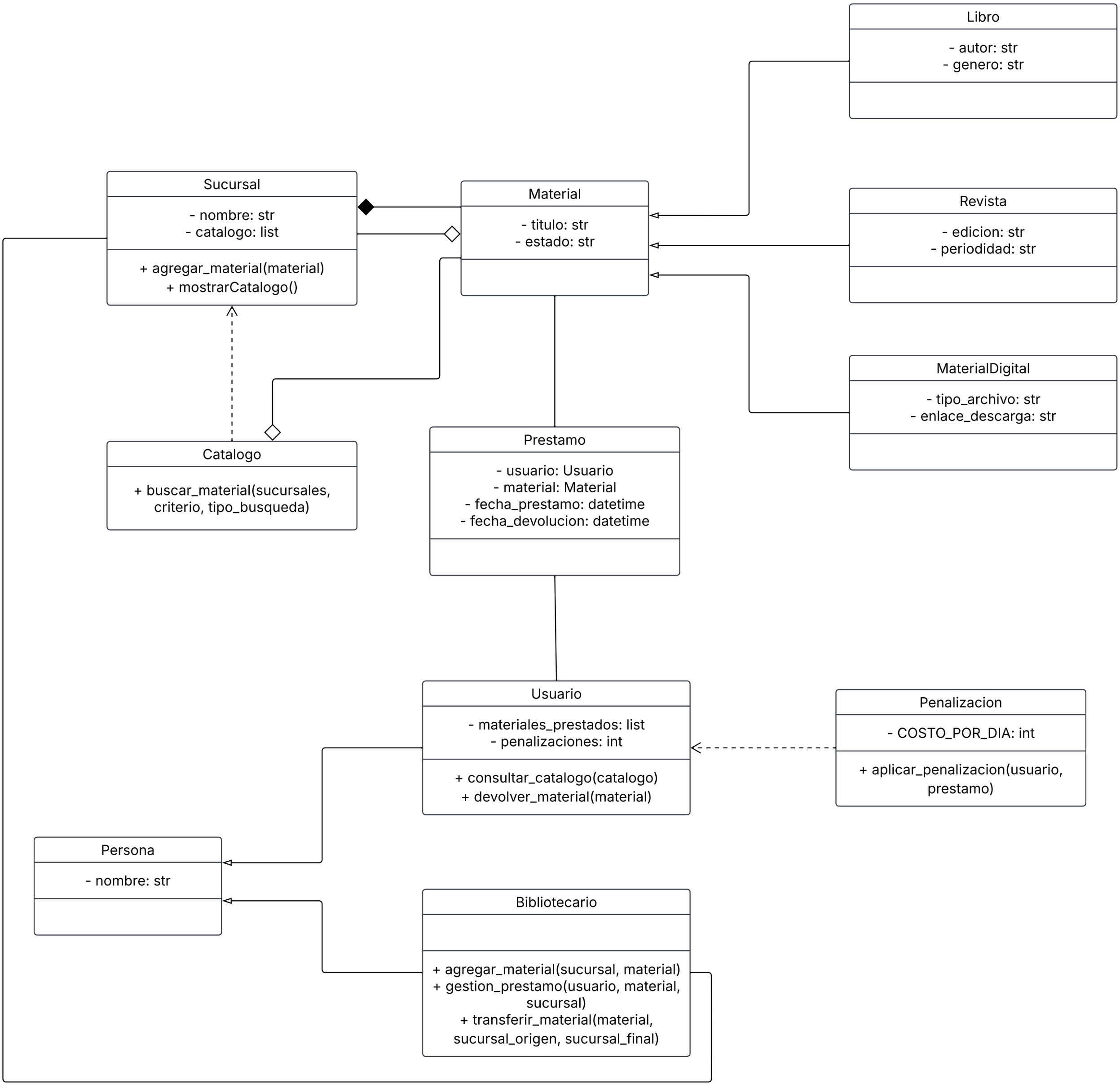


Diagrama UML "Biblioteca Digital"



Gestion de Pedidos de una Cafetería