PRACTICA DE LA UNIDAD 1

PROGRAMACION AVANZADA

Profesor: Jaime Alejandro Romero Sierra
Fátima Sánchez Domínguez

Cine

Justificación:

Este programa es necesario porque automatiza la gestión de un cine, mejorando la organización en la administración de funciones, reservas de asientos, control de salas y aplicación de promociones.
Las principales razones que lo justifican son:

- Permite a los usuarios reservar y cancelar asientos de manera organizada.
- Evita problemas de sobreventa al actualizar automáticamente los asientos disponibles.

- Evita conflictos al gestionar el uso de espacios y funciones de manera estructurada.
- Permite que los empleados consulten y actualicen la disponibilidad fácilmente.

- Los empleados pueden modificar descuentos y condiciones de promociones, atrayendo más clientes.

- Los empleados pueden registrar funciones y actualizar promociones.
- Los usuarios pueden ver su historial de reservas y realizar modificaciones.

- Permite agregar más salas, funciones y promociones sin afectar el sistema.
- Facilita la integración con otros servicios (pagos, notificaciones, etc.).

Clases utilizadas

1. Clase Persona

- Clase base para representar a una persona (usuario o empleado).
- Atributos: nombre, correo.
- Métodos:
  -registrar(): Agrega la persona a una lista de registros.
  -actualizar_datos(): Modifica la información de la persona.
  - personas_registradas(): Muestra la lista de personas registradas.

2. Clase Usuario (hereda de Persona)

- Representa a un cliente del cine que realiza reservas.
- Atributos: historial_reservas.
- Métodos:

-reservar(funcion, asientos): Permite reservar asientos en una función si hay disponibilidad.

-cancelar_reserva(funcion): Permite cancelar una reserva y devolver los asientos al sistema.

3. Clase Empleado (hereda de Persona)

- Representa a un empleado del cine que administra funciones y promociones.
- Atributos: rol (define su función en el cine).
- Métodos:

-agregar_funcion(funcion): Permite registrar una nueva función de cine.

-modificar_promocion(promocion, nuevo_descuento, nuevas_condiciones): Permite actualizar promociones.

4. Clase Espacio

- Representa un espacio físico dentro del cine.
- Atributos: capacidad, identificador.
- Métodos:

-descripcion(): Muestra información sobre el espacio.

5. Clase Sala (hereda de Espacio)

- Representa una sala de cine con un tipo específico.
- Atributos adicionales: tipo, disponibilidad.
- Métodos:

-consultar_disponibilidad(): Muestra si la sala está disponible o no.

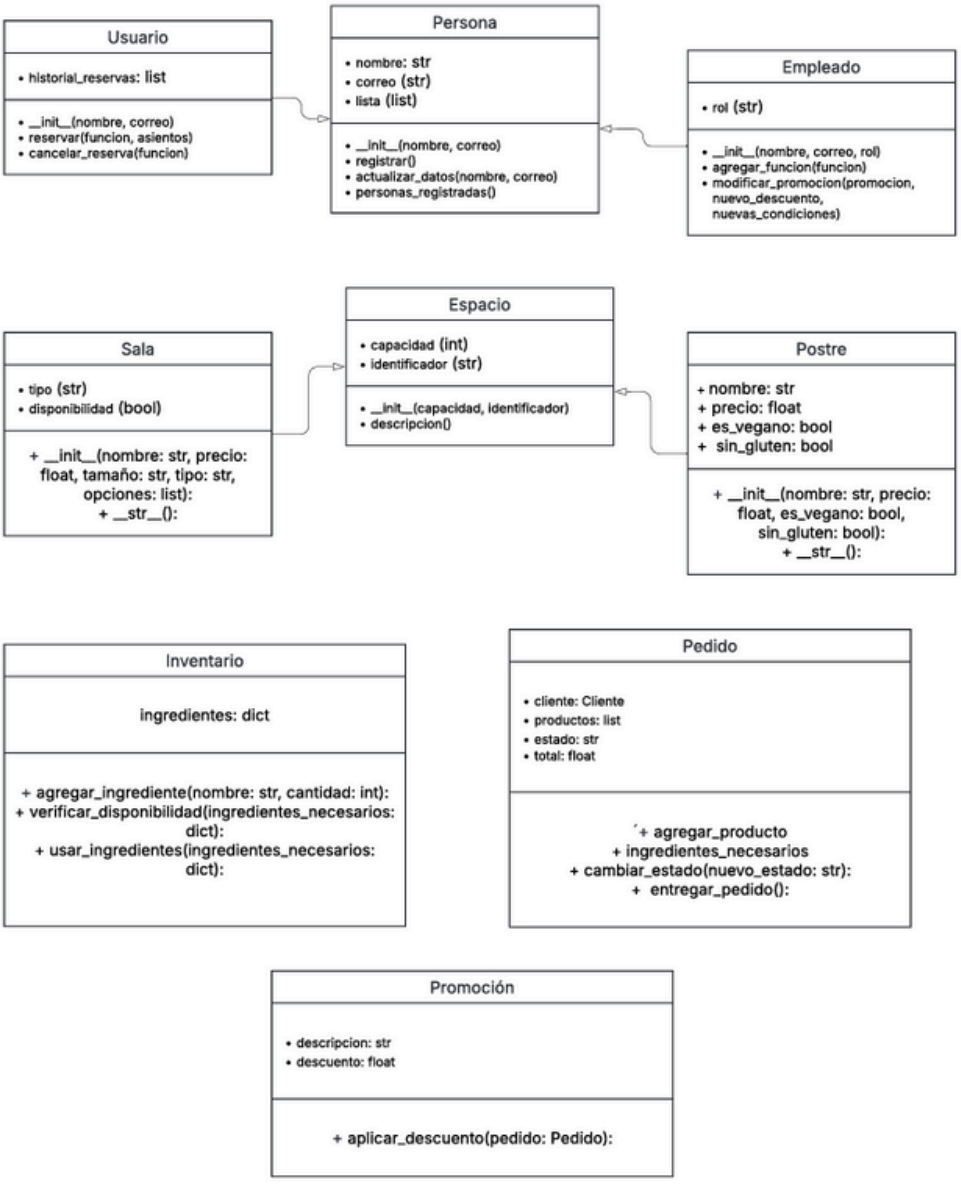
Código

```python
pelicula1 = Pelicula("Lalaland", "Romance", 136)
pelicula2 = Pelicula("Batman", "Accion", 195)

sala1 = Sala(100,"Sala 1","3DX")
sala2 = Sala(50,"Sala 2","Tradicional")

funcion1 = Funcion(pelicula1, sala1, "18:00")
funcion2 = Funcion(pelicula2, sala2, "20:00")

usuario1 = Usuario("Fernando Reyes", "ferzrey@email.com")
empleado1 = Empleado("Manuel Martínez", "manu.martinez@email.com", "Gerente")

usuario1.registrar()
empleado1.registrar()

usuario1.reservar(funcion1, 3)

usuario1.cancelar_reserva(funcion1)

promocion1 = Promocion(20, "Válido de lunes a jueves.")
promocion1.mostrar()
empleado1.modificar_promocion(promocion1, 30, "Válido todos los días antes de las 5 PM.")

Persona.personas_registradas()
```

✓ 0.0s

```
La persona Fernando Reyes ha sido registrada con el correo ferzrey@email.com
La persona Manuel Martínez ha sido registrada con el correo manu.martinez@email.com
Reserva realizada para 'Lalaland' en la sala Sala 1.
Reserva cancelada para 'Lalaland'.
Promoción: 20% de descuento. Condiciones: Válido de lunes a jueves.
Promoción modificada: 30% de descuento. Válido todos los días antes de las 5 PM..
Personas registradas
-Fernando Reyes - ferzrey@email.com
-Manuel Martínez - manu.martinez@email.com
```

UML


Usuario
• historial_reservas: list
• __init__(nombre, correo)
• reservar(funcion, asientos)
• cancelar_reserva(funcion)
Persona
• nombre: str
• correo (str)
• lista (list)
• __init__(nombre, correo)
• registrar()
• actualizar_datos(nombre, correo)
• personas_registradas()
Empleado
• rol (str)
• __init__(nombre, correo, rol)
• agregar_funcion(funcion)
• modificar_promocion(promocion, nuevo_descuento, nuevas_condiciones)
Sala
• tipo (str)
• disponibilidad (bool)
+ __init__(nombre: str, precio: float, tamaño: str, tipo: str, opciones: list):
+ __str__():
Espacio
• capacidad (int)
• identificador (str)
• __init__(capacidad, identificador)
• descripcion()
Postre
+ nombre: str
+ precio: float
+ es_vegano: bool
+ sin_gluten: bool
+ __init__(nombre: str, precio: float, es_vegano: bool, sin_gluten: bool):
+ __str__():
Inventario
ingredientes: dict
+ agregar_ingrediente(nombre: str, cantidad: int):
+ verificar_disponibilidad(ingredientes_necesarios: dict):
+ usar_ingredientes(ingredientes_necesarios: dict):
Pedido
• cliente: Cliente
• productos: list
• estado: str
• total: float
´+ agregar_producto
+ ingredientes_necesarios
+ cambiar_estado(nuevo_estado: str):
+ entregar_pedido():
Promoción
• descripcion: str
• descuento: float
+ aplicar_descuento(pedido: Pedido):

Cafetería

Justificación:

Este programa es un sistema para manejar pedidos en un negocio que ofrece productos, bebidas y postres, con el objetivo de gestionar clientes, empleados, inventarios y promociones

- Los clientes pueden realizar pedidos y el sistema verifica la disponibilidad de los ingredientes, también permite seguir el estado del pedido.
- El programa tiene una clase inventario que permite a los empleados agregar ingredientes y verificar su disponibilidad, eso ayuda asegurar que el inventario este organizado
- El sistema gestiona las promociones, por lo que es posible aplicar descuentos
- Reduce la carga de trabajo de los empleados lo que permite que el personal se pueda centrar en otras tareas como la atención al cliente

clases a ocupar :

1

- **Persona** es la clase base de la que heredan tanto Cliente como Empleado, lo que permite compartir el atributo nombre entre ambos.
- **Cliente** y Empleado son clases especializadas que agregan comportamientos específicos, como gestionar pedidos o actualizar el inventario.
- **ProductoBase** es la clase base para los productos como Bebida y Postre, lo que permite reutilizar código para productos similares pero con características específicas, como los ingredientes o las opciones de personalización.

Cliente:

- Los clientes pueden realizar pedidos mediante el método (realizar_pedido()). Se verifica la disponibilidad de ingredientes en el inventario y se aplica un descuento si tienen puntos de fidelidad suficientes.

.

Empleado:

- Los empleados gestionan el inventario, son capaces de agregar ingredientes cuando se terminan.

ProductoBase y sus subclases **(Bebida y Postre):**

- Los productos tienen atributos comunes como nombre, precio y ingredientes, que son definidos en la clase base ProductoBase.
- **Bebida** y **Postre** extienden esta funcionalidad agregando características adicionales, como tamaño o tipo para las bebidas, y propiedades como vegano y sin_gluten para los postres.

Inventario:

- La clase Inventario gestiona los ingredientes disponibles, permitiendo verificar su disponibilidad antes de aceptar un pedido y actualizarlos después de procesar el pedido. El sistema permite asegurar que siempre haya suficiente stock de los ingredientes necesarios.

Pedido:

- Los pedidos son gestionados mediante la clase Pedido, que permite agregar productos, calcular el total, y gestionar su estado.
- La función de verificar disponibilidad en el inventario y actualizar el stock después de un pedido también está incluida.

Promoción:

- La clase Promocion permite aplicar un descuento sobre el total de un pedido si se cumplen ciertas condiciones

Código

```
inventario = Inventario()
inventario.agregar_ingrediente("Cafe", 10)
inventario.agregar_ingrediente("Te", 11)
inventario.agregar_ingrediente("Leche de coco", 5)
inventario.agregar_ingrediente("Leche deslactosada", 13)
inventario.agregar_ingrediente("Huevos", 20)
inventario.agregar_ingrediente("Azúcar", 30)
inventario.agregar_ingrediente("Chocolate", 18)
inventario.agregar_ingrediente("Miel", 9)
inventario.agregar_ingrediente("Fresas", 16)
inventario.agregar_ingrediente("Harina", 15)
inventario.agregar_ingrediente("Hielo", 15)
```

✓ 0.0s

```
inventario.consultar_inventario()
```

✓ 0.0s

```
Inventario actual:
- Cafe: 10 unidades
- Te: 11 unidades
- Leche de coco: 5 unidades
- Leche deslactosada: 13 unidades
- Huevos: 20 unidades
- Azúcar: 30 unidades
- Chocolate: 18 unidades
- Miel: 9 unidades
- Fresas: 16 unidades
- Harina: 15 unidades
```

```
cliente1 = Cliente("Sugey")
cliente2 = Cliente("Jimmy")
cliente3 = Cliente("Karen")
empleado1 = Empleado("Julian", "Barista")
empleado2 = Empleado("Marta", "Mesera")
empleado3 = Empleado("Pedro", "Gerente")
```

✓ 0.0s

```
bebida1 = Bebida("Chai Latte", 55, "Mediano", "Caliente", {"Te": 1, "Leche de coco": 1, "Azúcar": 1, "Hielo": 1})
postre1 = Postre("Galleta de Chocolate", 15, personalizacion={"Azúcar": 1, "Chocolate": 1})
bebida2 = Bebida("Cafe Vegano", 35, "Pequeño", "Caliente", {"Café": 1, "Azúcar": 1})
postre2 = Postre("Pan Vegano", 30, personalizacion={"Azúcar": 1, "Harina": 1}, vegano=True)
bebida3 = Bebida("Café Latte", 50, "Mediano", "Caliente", {"Café": 1, "Leche deslactosada": 1, "Azúcar": 1})
postre3 = Postre("Brownie Sin Gluten", 30, personalizacion={"Azúcar": 1}, sin_gluten=True)
```

✓ 0.0s

```
bebida1.descripcion()
postre1.descripcion()
bebida2.descripcion()
postre2.descripcion()
bebida3.descripcion()
postre3.descripcion()
```

✓ 0.0s

```
Nombre: Chai Latte
Precio: $55
Tamaño: Mediano
Ingredientes:
- Agua
- Te
```

```
- Agua
- Te
- Leche de coco
- Azúcar
- Hielo
Nombre: Galleta de Chocolate
Precio: $15
Vegano: No
Sin gluten: No
Ingredientes:
- Harina
- Azúcar
- Chocolate
Nombre: Cafe Vegano
Precio: $35
Tamaño: Pequeño
Ingredientes:
- Agua
- Café
- Azúcar
Nombre: Pan Vegano
...
Sin gluten: Si
Ingredientes:
- Harina
- Azúcar
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```python
pedido1 = Pedido(cliente1)
pedido1.agregar_producto(bebida1)
pedido1.agregar_producto(postre1)

pedido2 = Pedido(cliente2)
pedido2.agregar_producto(bebida2)
pedido2.agregar_producto(postre2)

pedido3 = Pedido(cliente3)
pedido3.agregar_producto(bebida3)
pedido3.agregar_producto(postre3)
```

✓ 0.0s

```
Se ha añadido un nuevo pedido para Sugey
Se ha añadido Chai Latte con costo de $55 al pedido de Sugey
Se ha añadido Galleta de Chocolate con costo de $15 al pedido de Sugey
Se ha añadido un nuevo pedido para Jimmy
Se ha añadido Cafe Vegano con costo de $35 al pedido de Jimmy
Se ha añadido Pan Vegano con costo de $30 al pedido de Jimmy
Se ha añadido un nuevo pedido para Karen
Se ha añadido Café Latte con costo de $50 al pedido de Karen
Se ha añadido Brownie Sin Gluten con costo de $30 al pedido de Karen
```

```python
promo = Promocion("Descuento de 10% para clientes frecuentes", 10)
```

✓ 0.0s

```
cliente1.realizar_pedido(pedido1, inventario, promo)
cliente2.realizar_pedido(pedido2, inventario, promo)
cliente3.realizar_pedido(pedido3, inventario, promo)
```

[20] ✓ 0.0s

```
Faltan ingredientes: Agua, Hielo
No hay suficiente stock para realizar el pedido.
Faltan ingredientes: Agua, Café
No hay suficiente stock para realizar el pedido.
Faltan ingredientes: Agua, Café
No hay suficiente stock para realizar el pedido.
```

```
pedido1.actualizar_estado("Entregado")
pedido2.actualizar_estado("Entregado")
pedido3.actualizar_estado("Entregado")
```

[21] ✓ 0.0s

```
Nuevo estado del pedido: Entregado
Nuevo estado del pedido: Entregado
Nuevo estado del pedido: Entregado
```

```
cliente1.consultar_historial()
cliente2.consultar_historial()
cliente3.consultar_historial()
```

[22] ✓ 0.0s

```
Historial pedidos de Sugey
Historial pedidos de Jimmy
Historial pedidos de Karen
```

```
inventario.consultar_inventario()
```

✓ 0.0s

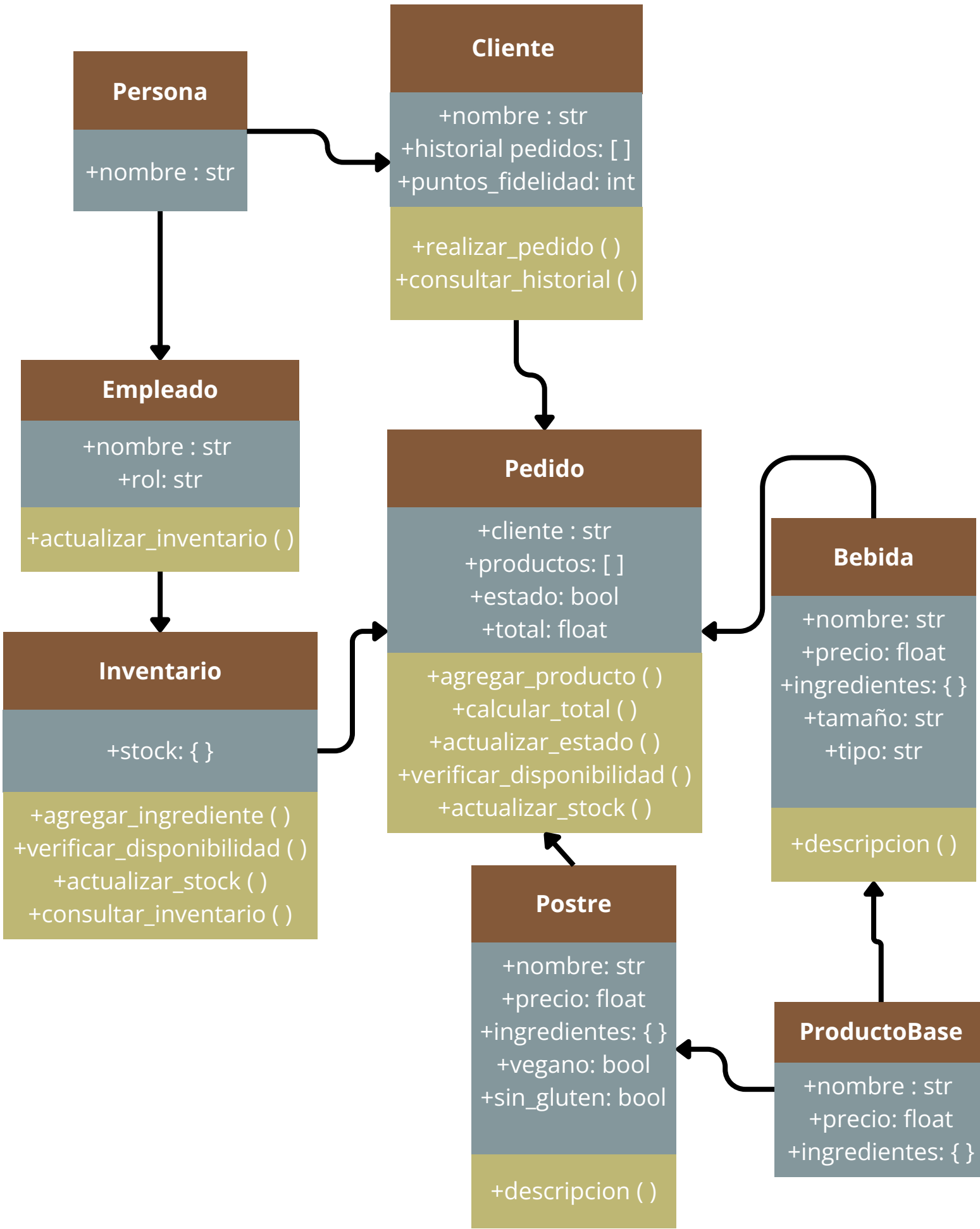
```
Inventario actual:
- Cafe: 10 unidades
- Te: 11 unidades
- Leche de coco: 5 unidades
- Leche deslactosada: 13 unidades
- Huevos: 20 unidades
- Azúcar: 30 unidades
- Chocolate: 18 unidades
- Miel: 9 unidades
- Fresas: 16 unidades
- Harina: 15 unidades
```

```
empleado1.actualizar_inventario(inventario, "Café", 5)
```

✓ 0.0s

```
Se han añadido 5 unidades de Café
```

UML


Persona
+nombre : str
Cliente
+nombre : str
+historial pedidos: [ ]
+puntos_fidelidad: int
+realizar_pedido ( )
+consultar_historial ( )
Empleado
+nombre : str
+rol: str
+actualizar_inventario ( )
Pedido
+cliente : str
+productos: [ ]
+estado: bool
+total: float
+agregar_producto ( )
+calcular_total ( )
+actualizar_estado ( )
+verificar_disponibilidad ( )
+actualizar_stock ( )
Bebida
+nombre: str
+precio: float
+ingredientes: { }
+tamaño: str
+tipo: str
+descripcion ( )
Inventario
+stock: { }
+agregar_ingrediente ( )
+verificar_disponibilidad ( )
+actualizar_stock ( )
+consultar_inventario ( )
Postre
+nombre: str
+precio: float
+ingredientes: { }
+vegano: bool
+sin_gluten: bool
+descripcion ( )
ProductoBase
+nombre : str
+precio: float
+ingredientes: { }

Biblioteca digital

Justificación:

Este sistema de gestión, busca mejorar la organización y el control de los recursos de la biblioteca con un programa que automatiza las principales funciones:

- Permite la gestión de libros, revistas y material digital.
- Da seguimiento a los materiales prestados, poniéndoles fechas de devolución y registrando los retrasos.
- Facilita el control de los prestamos realizados por cada usuario.
- Permite hacer transferencias de materiales entre diferentes bibliotecas o sucursales.

Clases utilizadas:

1. Clase Material

- Clase base que representa cualquier tipo de material en la biblioteca (libros, revistas, materiales digitales).
- Atributos: titulo, estado (disponible o prestado).
- Métodos:

+prestar(): Cambia el estado a "prestado" si está disponible.
+devolver(): Restablece el estado a "disponible".
+__str__(): Muestra información sobre el material.

2. Clase Libro (hereda de Material)

- Representa un libro en la biblioteca.
- Atributos adicionales: autor, género.
- Sobrescribe __str__() para mostrar detalles del libro.

3. Clase Revista (hereda de Material)

- Representa una revista en la biblioteca.
- Atributos adicionales: edición, periodicidad.
- Sobrescribe __str__() para mostrar detalles de la revista.

4. Clase MaterialDigital (hereda de Material)

- Representa un material digital como un PDF o un libro electrónico.
- Atributos adicionales: tipo_archivo, enlace.
- Sobrescribe __str__() para mostrar detalles del material digital.

5. Clase Persona

- Clase base que representa a una persona en el sistema.
- Atributos: nombre, apellido.

6. Clase Usuario (hereda de Persona)

- Representa un usuario que puede pedir materiales en préstamo.
- Atributos adicionales: materiales_prestados, penalizaciones.
- Métodos:

+pedir_prestado(): Permite a un usuario solicitar un material si está disponible.
+devolver_material(): Permite devolver un material y aplica penalización si hay retraso.

7. Clase Bibliotecario (hereda de Persona)

- Representa a un bibliotecario que administra materiales en una sucursal.
- Atributos adicionales: sucursal (biblioteca donde trabaja).
- Métodos:

+agregar_material(): Agrega un material a la sucursal.
+transferir_material(): Permite mover materiales entre sucursales.

8. Clase Sucursal

- Representa una biblioteca física con su propio catálogo.
- Atributos: nombre, catalogo (lista de materiales).
- Métodos:

+agregar_material(): Añade un material a la sucursal.
+eliminar_material(): Elimina un material del catálogo.
+mostrar_catalogo(): Muestra los materiales disponibles en la sucursal.

9. Clase Penalizacion

- Maneja las penalizaciones de los usuarios por retrasos en la devolución.
- Atributos: usuario, dias_retraso, multa (se calcula como $1 por día de retraso).
- Métodos:

+__str__(): Muestra información sobre la penalización

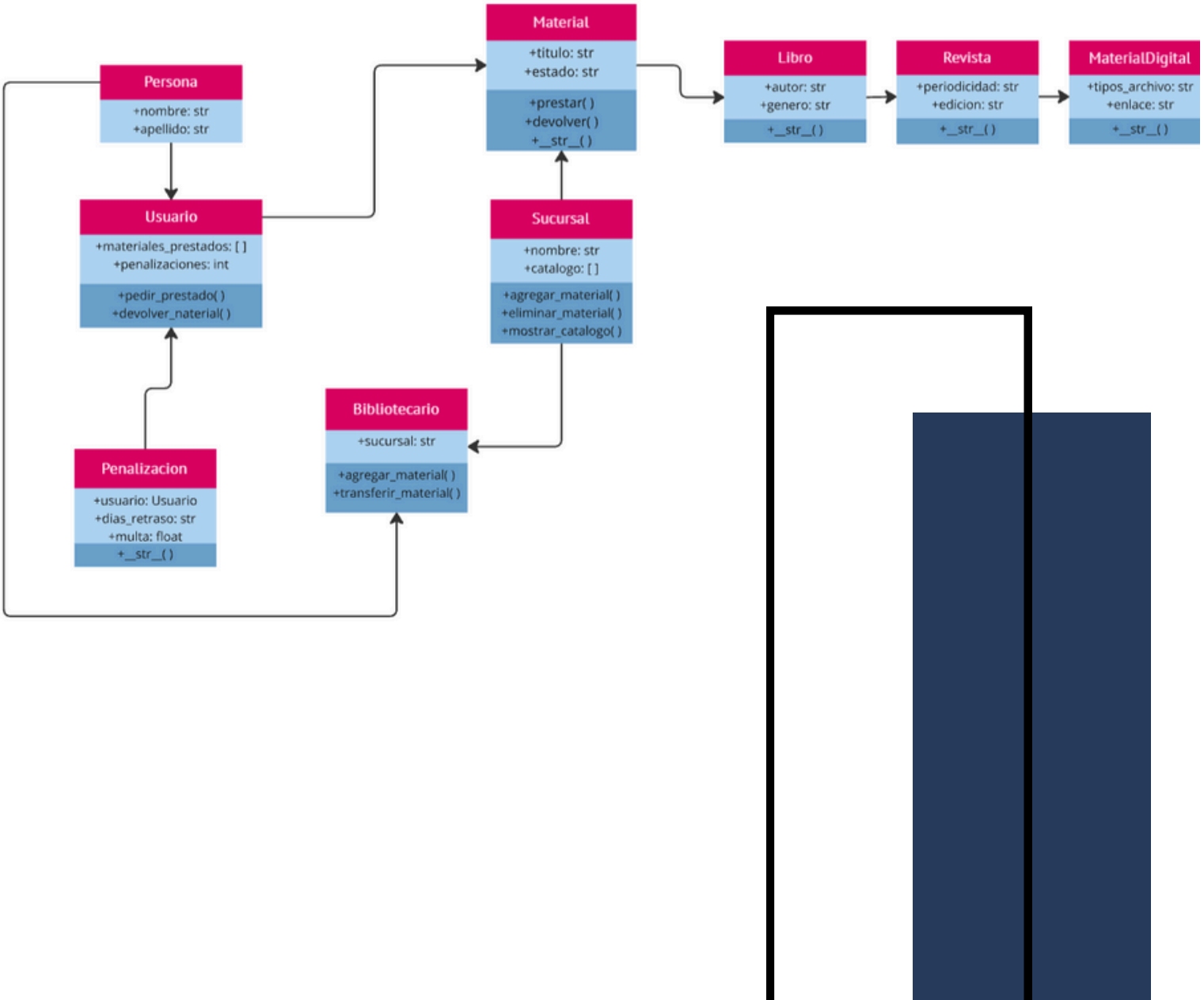
Código

```
libro1 = Libro("Cronicas del espacio", "Neil deGrasse Tyson", "Informativo")
revista1 = Revista("National Geographic", "Enero 2025", "Mensual")
digital1 = MaterialDigital("Data Sciencie for dummies", "PDF", "http://iakshs.com/python.pdf")


sucursal1 = Sucursal("Biblioteca Central")
sucursal2 = Sucursal("Biblioteca Sur")

usuario1 = Usuario("Pedro","Medina")
bibliotecario1 = Bibliotecario("Ana","Gonzalez", sucursal1)
penalizacion1 = Penalizacion(usuario1, 3)


#Agregar material
bibliotecario1.agregar_material(libro1)
bibliotecario1.agregar_material(revista1)
bibliotecario1.agregar_material(digital1)


#Hacer el prestamo con retraso
fecha_prestamo = datetime.now() - timedelta(days=10)  # Simular un préstamo hace 10 dias
usuario1.pedir_prestado(libro1, fecha_prestamo)
print(penalizacion1)

#Devolver el material con retraso
usuario1.devolver_material(libro1)

# Transferir material entre sucursales
bibliotecario1.transferir_material(revista1, sucursal2)

# Consultar catalogos
sucursal1.mostrar_catalogo()
sucursal2.mostrar_catalogo()
```

Código ejecutado:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\fatis> & C:/Users/fatis/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/fatis/OneDrive/Documentos/Python/Programacion Avanzada/Biblioteca_Digital.py"
Cronicas del espacio fue agregado a la biblioteca Biblioteca Central
National Geographic fue agregado a la biblioteca Biblioteca Central
Data Sciencie for dummies fue agregado a la biblioteca Biblioteca Central
Pedro pidio prestado 'Cronicas del espacio' el 2025-02-02, debe devolverlo antes del 2025-02-09
Penalizacion para Pedro: 3 dias de retraso, multa de $3
Pedro ha devuelto 'Cronicas del espacio' con 3 dias de retraso. Penalizaciones acumuladas: 3
El National Geographic fue eliminado de la biblioteca Biblioteca Central
National Geographic fue agregado a la biblioteca Biblioteca Sur
Ana transfirio "National Geographic" a "Biblioteca Sur
Catalogo de la biblioteca "Biblioteca Central":
1, Libro: Cronicas del espacio por Neil deGrasse Tyson (Informativo)
2, Material digital: Data Sciencie for dummies (PDF) enlace: http://iakshs.com/python.pdf
Catalogo de la biblioteca "Biblioteca Sur":
1, Revista: National Geographic, edicion: Enero 2025 (Mensual)
PS C:\Users\fatis>
```

UML



Link de GitHub:

https://github.com/fluvwonie/Practica1