PRÁCTICA
UNIDAD 1

Programación Avanzada

Profesor:
Romero Sierra Jaime Alejandro

Elaborado por:
Velez Ortega Eresto.

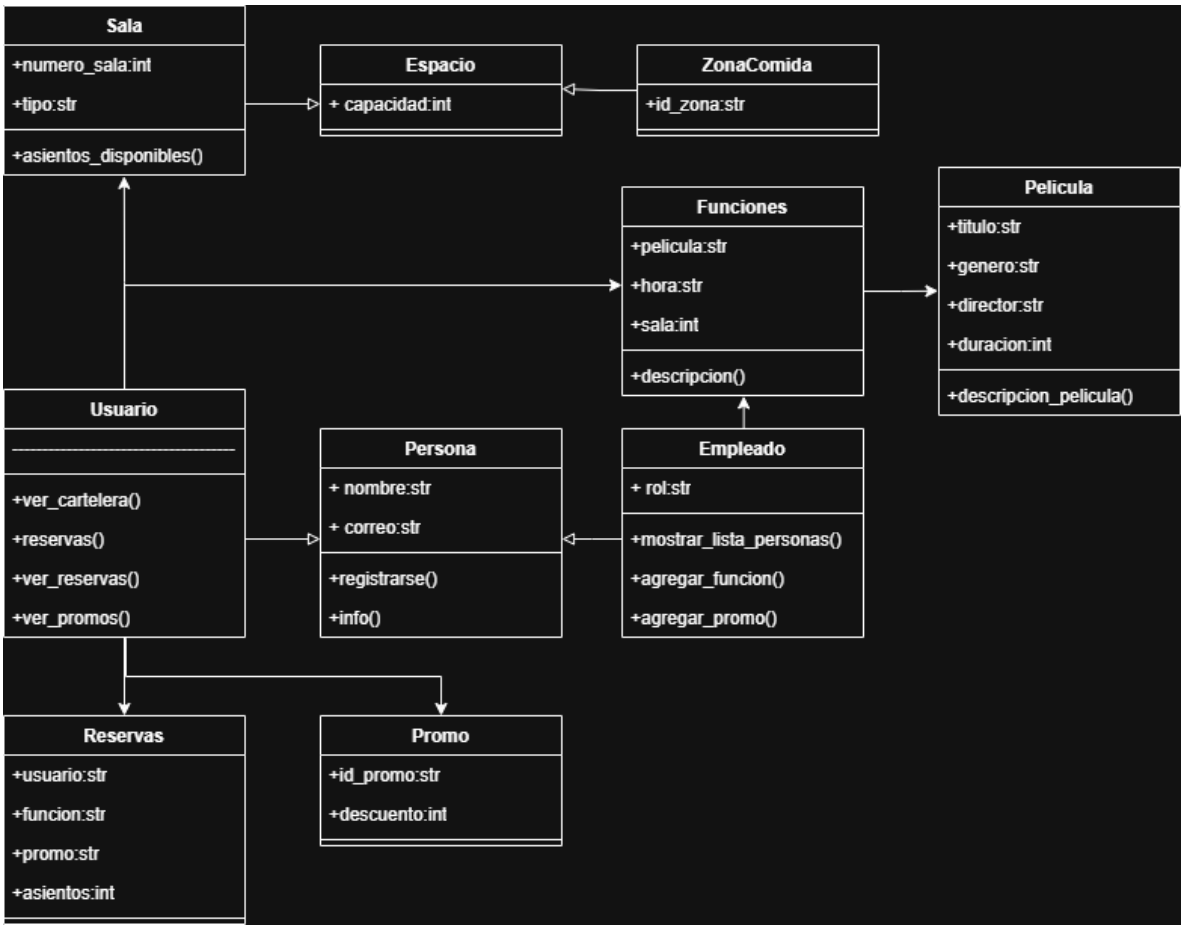
18 de Febrero del 2025
Programación Avanzada: Practica unidad 1

PRACTICA 1.1: CINE

Justificación:

Este sistema de gestión de cine se desarrolla para optimizar la administración de funciones, reservas y promociones, permitiendo una experiencia más eficiente y personalizada tanto para clientes como para empleados. Para los clientes, el sistema ofrece la comodidad de reservar entradas en línea, acceder a información actualizada sobre disponibilidad y disfrutar de recomendaciones personalizadas. Además, facilita la automatización de procesos, lo que reduce errores humanos y mejora la organización interna del cine, lo que se traduce en una mayor eficiencia y ahorro de costos.

Diagrama UML:



Clases usadas:

Clase Persona:

Atributos:

- Nombre: se pide el nombre para identificar al cliente
- Correo: se pide el correo para tener una forma de contactar al cliente de ser necesario
- Rol (asignado automáticamente): es auto asignado con rol de “cliente”, para que no se le tenga que agregar manualmente y se pueda diferenciar automáticamente de los empleados

Métodos:

- Registrarse: registra el objeto dentro de una lista para saber quien esta registrado
- Info: muestra la información personal del usuario registrado

```python
class Persona:
    listaPersonas = []

    def __init__(self, nombre, correo):
        self.nombre = nombre
        self.correo = correo
        self.rol = "cliente"

    def registrarse(self):
        Persona.listaPersonas.append(self)
        print("Usuario registrado correctamente")

    def info(self):
        print(f"{self.rol}: {self.nombre}, Correo: {self.correo}")
```

[1] ✓ 0.0s

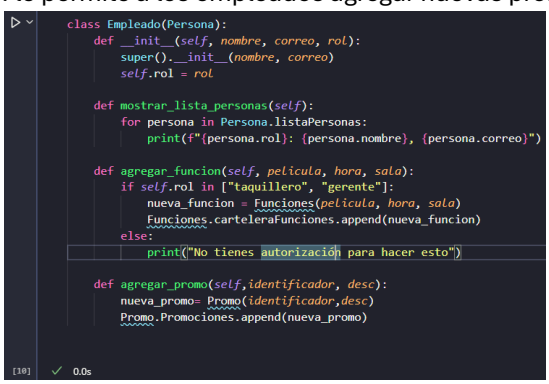
Clase Empleado:

Atributos:

- Nombre (heredado de la clase Persona)
- correo (heredado de la clase Persona)
- rol: en esta clase si se pide para que se lleve un mejor control sobre el personal además de servir para restringir uien puede acceder a ciertos metodos

Métodos:

- mostrar_lista_personas: esto es para mostrar las personas que se encuentran registradas (clientes y empleados por igual)
- agrear_funcion: este método permite a los empleados con el rol de tauillero o gerente (solo ellos) agregar nuevas funciones de peliculas
- agregar_promo: le permite a los empleados agregar nuevas promociones



```python
class Empleado(Persona):
    def __init__(self, nombre, correo, rol):
        super().__init__(nombre, correo)
        self.rol = rol

    def mostrar_lista_personas(self):
        for persona in Persona.listaPersonas:
            print(f"{persona.rol}: {persona.nombre}, {persona.correo}")

    def agregar_funcion(self, pelicula, hora, sala):
        if self.rol in ["taquillero", "gerente"]:
            nueva_funcion = Funciones(pelicula, hora, sala)
            Funciones.carteleraFunciones.append(nueva_funcion)
        else:
            print("No tienes autorización para hacer esto")

    def agregar_promo(self,identificador, desc):
        nueva_promo= Promo(identificador,desc)
        Promo.Promociones.append(nueva_promo)
```

[10] ✓ 0.0s

Clase Usuario:

Atributos:

- Nombre (heredado de la clase Persona)
- correo (heredado de la clase Persona)

Métodos:

- ver_cartelera: le muestra a el usuario las funciones disponibles
- reservar: le permite al usuario reservar asientos de una función (no se puede reservar más asientos de los que hay disponibles, ni reservar menos de 1)
- ver_reservas: muestra al usuario una lista de las reservas que ya a echo
- ver_promo: le permite a el usuario ver las promociones actuales del cine

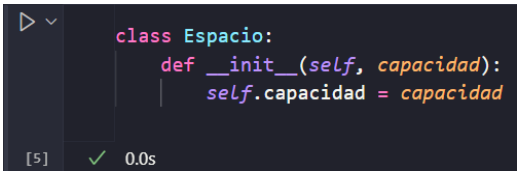
```
class Usuario(Persona):
    reservas_realizadas = []

    def __init__(self, nombre, correo):
        super().__init__(nombre, correo)

    def ver_cartelera(self):
        i=1
        for movie in (Funciones.carteleraFunciones):
            print(f"{i}.- {movie}")
            i+=1

    def reservar(self, funcion, promo, asientos):
        if asientos>0:
            asientos_disponibles = funcion.sala.capacidad - funcion.sala.asientosReservados
            if asientos_disponibles >= asientos:
                nueva_reserva = Reservas(self.nombre, funcion, promo, asientos)
                Usuario.reservas_realizadas.append(nueva_reserva)
                print(f"\nReserva realizada para la película {funcion.pelicula.titulo} a las {funcion.hora} en la sala {funcion.sala.numerosala} con {asientos} asientos")
                funcion.sala.asientosReservados += asientos
            else:
                print(f"No hay suficientes asientos disponibles, asietos disponibles: {asientos_disponibles}")
        else:
            print("no puedes reservar menos de 1 asiento")

    def ver_reservas(self):
        print(f"RESERVAS DE: {self.nombre}")
        for reserva in Usuario.reservas_realizadas:
            print(reserva)

    def ver_promos(self):
        for promocion in Promo.Promociones:
            print(promocion)

[4]  ✓ 0.0s
```

Clase Espacio:

Atributos:

- capacidad: para saber cual es la capacidad de personas del espacio

no se agregó métodos ya que se consideró que esta clase es muy general y el enfoque que se le dio al programa n o lo requería tan necesariamente (al igual que zona de comida)



```
class Espacio:
    def __init__(self, capacidad):
        self.capacidad = capacidad

[5]  ✓ 0.0s
```

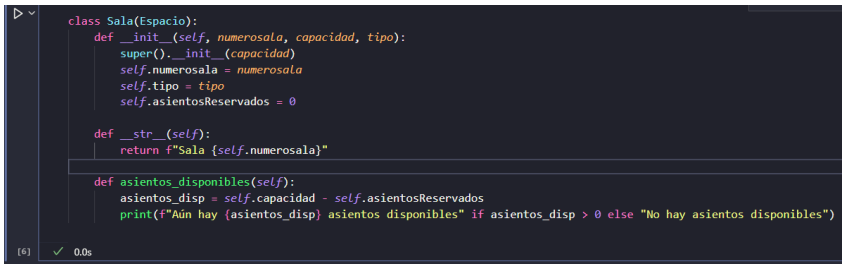
Clase Sala:

Atributos:

- capacidad (heredado de la clase Espacio)
- numerosala: usada como identificador para que a la hora de ver cartelera el usuario sepa en que sala se proyectara la película
- tipo; para saber si la sala es para experiencias 3D, IMAX, etc

Métodos:

- asiestos_disponibles: muestra cuantos hay disponibles en la sala en función a cuantos asientos han sido reservados–



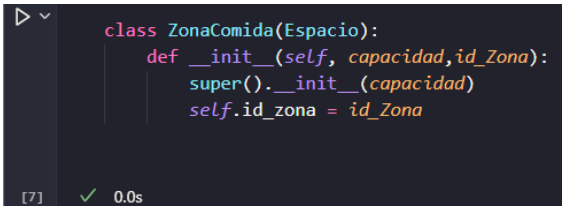
```
class Sala(Espacio):
    def __init__(self, numerosala, capacidad, tipo):
        super().__init__(capacidad)
        self.numerosala = numerosala
        self.tipo = tipo
        self.asientosReservados = 0

    def __str__(self):
        return f"Sala {self.numerosala}"

    def asientos_disponibles(self):
        asientos_disp = self.capacidad - self.asientosReservados
        print(f"Aún hay {asientos_disp} asientos disponibles" if asientos_disp > 0 else "No hay asientos disponibles")

[6]  ✓ 0.0s
```

Clase ZonaComida:

Atributos:

- capacidad (heredado de la clase Espacio)
- id_Zona: un identificador para llevar una mejor organización

no se crearon métodos ya que en el enfoque usado una zona de comida no requiere de un control mayor a tener un identificador en caso de que el personal necesite ubicarlo



```
class ZonaComida(Espacio):
    def __init__(self, capacidad,id_Zona):
        super().__init__(capacidad)
        self.id_zona = id_Zona

[7]  ✓ 0.0s
```

Clase Película:

Atributos:

- titulo: para saber que película es
- genero: para indicar que género es
- duración: para indicar la duración de la película
- director: para indicar quien es el director de la película

Métodos:

- **descripción_pelicula: muestra los datos de la pelicula**

```python
class Pelicula:
    def __init__(self, titulo, genero, duracion, director):
        self.titulo = titulo
        self.genero = genero
        self.duracion = duracion
        self.director = director

    def __str__(self):
        return self.titulo

    def descripcion_pelicula(self):
        print(f"pelicula: {self.titulo}, genero: {self.genero}, director: {self.director}, duracion: {self.duracion}")
```

[8] ✓ 0.0s

Clase Promo:

Atributos:

- idProm: usado como identificador de la promoción
- descuento: indica que porcentaje se descontara a una película

Métodos:

No incluí ningún método más que el __str__ para dar una breve descripción de la promoción

```python
class Promo:

    Promociones = []

    def __init__(self, idProm, descuento):
        self.idProm = idProm
        self.descuento = descuento

    def __str__(self):
        return f"Promoción: {self.idProm}, Descuento: {self.descuento}"
```

[9] ✓ 0.0s

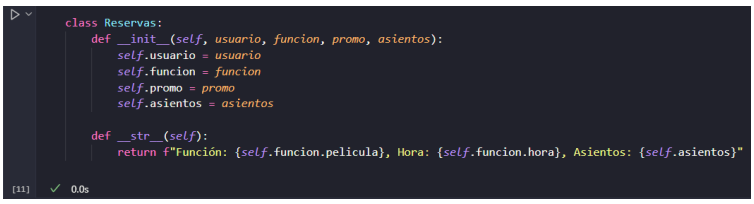
Clase Promo:

Atributos:

- usuario: usado para saber a que usuario pertenece la reserva y guardarlo en su lista de reservas
- funcion: para saber a que función y por lo tentó a que sala
- promo: para saber cuánto descuento tendrá
- asientos: para saber cuantos asientos se tienen que descontar de la sala

Métodos:

Al igual que la anterior clase no incluí ningún método mas que el __str__



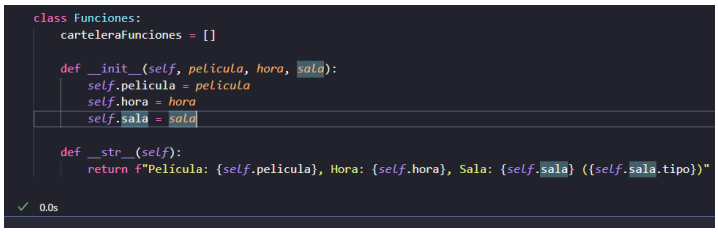
Clase Funciones:

Atributos:

- pelicula: para saber que película se proyectara durante la función
- hora: para indicar a qué hora será la función
- sala: para indicar en donde se proyectará la película

Métodos:

Al igual que las anteriores clases no incluí ningún método más que el __str__ , ya que la función se guarda en una lista em la misma clase



Ejemplo de uso:

Ernesto = Usuario("Ernesto Velez Ortega", "netovelesz2019@gmail.com")

Ernesto.registrarse()

Ernesto.info()

santiago = Empleado("santiago lopez carreon", "pepitoloqendo123@gmail.com", "taquillero")

santiago.registrarse()

santiago.info()

```
santiago.mostrar_lista_personas()


#peliculas
elresplandor=Pelicula("el resplandor", "terror psicologico", 150, "setefin quing")


#salas
salaIMAX1 = Sala("A150",150, "IMAX")
sala3d = Sala("B150",90, "3D")
salaIMAX2 = Sala("A200",50, "IMAX")


santiago.agregar_funcion(elresplandor, "15:30", salaIMAX2)
santiago.agregar_promo("A150", -150)


Ernesto.ver_cartelera()


Ernesto.ver_promos()
pelicula_a_reservar = Funciones.carteleraFunciones[0]
Ernesto.reservar(pelicula_a_reservar, "A150",50)
```

Link al proyecto:

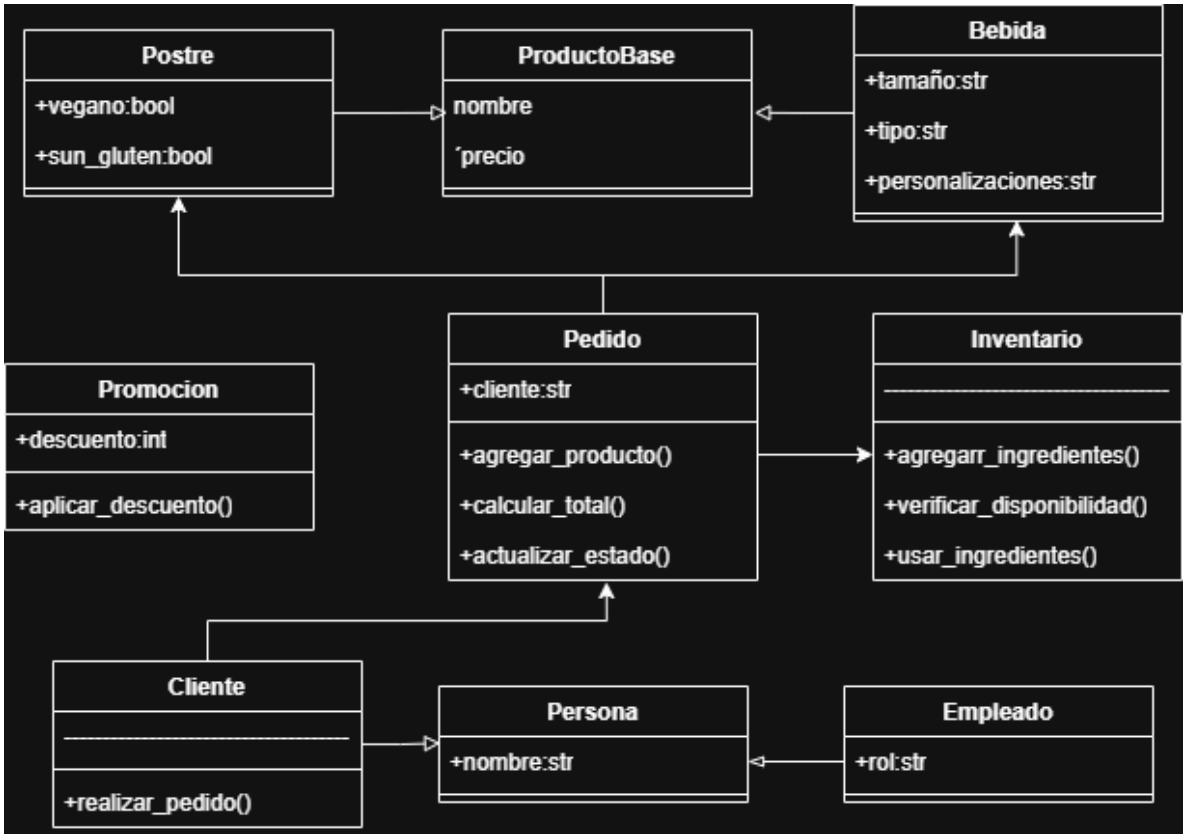
https://github.com/netobunobi/ProgramacionAvanzada/blob/main/cine.ipynb

PRACTICA 1.2: CAFETERIA

Justificación:

Este sistema de gestión de pedidos en una cafetería se desarrolla para optimizar la administración de clientes, empleados, productos e inventario, permitiendo un servicio más eficiente y personalizado. Los clientes pueden realizar pedidos personalizados, mientras que los empleados pueden gestionar el inventario y garantizar la disponibilidad de ingredientes. Además, el sistema permite aplicar promociones para clientes frecuentes, lo que mejora la experiencia del usuario y optimiza el uso de recursos.

Diagrama UML:



Clases usadas:

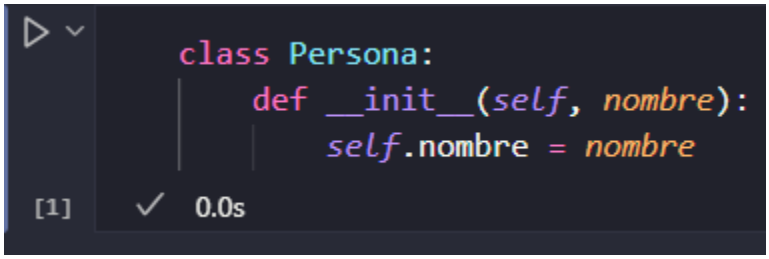
Clase Persona:

Atributos:

- Nombre: identifica a la persona.

Métodos:

No incluye métodos específicos, ya que sirve como base para otras clases.



Clase Cliente (hereda de Persona):

Atributos:

- Historial de pedidos: almacena los pedidos realizados por el cliente.

Métodos:

- Realizar pedido: registra un nuevo pedido en el historial del cliente.

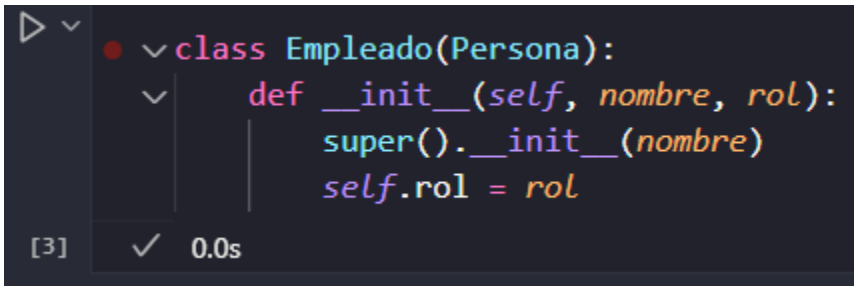
Clase Empleado (hereda de Persona):

Atributos:

- Rol: determina si el empleado es mesero, barista o gerente. }

Métodos:

No incluye métodos específicos, ya que su función principal es representar un empleado en el sistema.



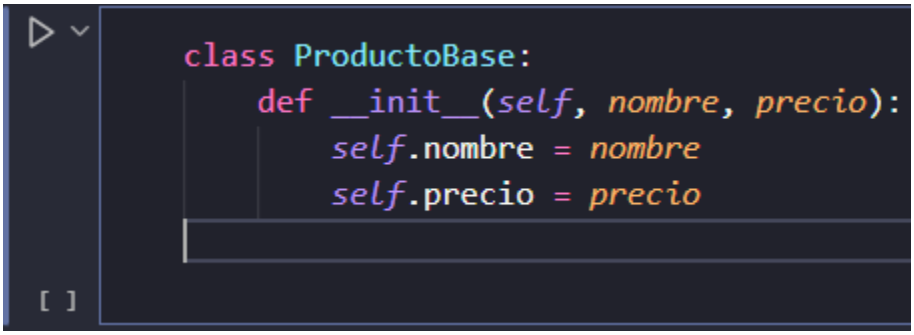
Clase ProductoBase:

Atributos:

- Nombre: identifica el producto.
- Precio: indica el costo del producto.

Métodos:

No incluye métodos específicos, ya que sirve como base para otros productos.



Clase Bebida (hereda de ProductoBase):

Atributos:

- Tamaño: indica el tamaño de la bebida.

- Tipo: define si es caliente o fría.
- Personalizaciones: lista de opciones personalizadas como "extra leche" o "sin azúcar".

Métodos:

No incluye métodos adicionales.

```
class Bebida(ProductoBase):
    def __init__(self, nombre, precio, tamaño, tipo, personalizaciones=None):
        super().__init__(nombre, precio)
        self.tamaño = tamaño
        self.tipo = tipo
        self.personalizaciones = personalizaciones if personalizaciones else []
```

[5] ✓ 0.0s

Clase Postre (hereda de ProductoBase):

Atributos:

- Vegano: indica si el postre es apto para veganos.
- Sin gluten: señala si el postre es libre de gluten.

Métodos:

No incluye métodos adicionales.

```
class Postre(ProductoBase):
    def __init__(self, nombre, precio, vegano=False, sin_gluten=False):
        super().__init__(nombre, precio)
        self.vegano = vegano
        self.sin_gluten = sin_gluten
```

[6] ✓ 0.0s

Clase Inventario:

Atributos:

- Ingredientes: diccionario que almacena la cantidad disponible de cada ingrediente.

Métodos:

- Agregar ingrediente: incrementa la cantidad de un ingrediente en el inventario.
- Verificar disponibilidad: revisa si los ingredientes necesarios están en stock.
- Usar ingredientes: descuenta los ingredientes utilizados en un pedido.

```python
class Inventario:
    def __init__(self):
        self.ingredientes = {}

    def agregar_ingrediente(self, nombre, cantidad):
        if nombre in self.ingredientes:
            self.ingredientes[nombre] += cantidad
        else:
            self.ingredientes[nombre] = cantidad

    def verificar_disponibilidad(self, pedido):
        for producto in pedido.productos:
            for ingrediente in producto.personalizaciones:
                if self.ingredientes.get(ingrediente, 0) <= 0:
                    return False
        return True

    def usar_ingredientes(self, pedido):
        if self.verificar_disponibilidad(pedido):
            for producto in pedido.productos:
                for ingrediente in producto.personalizaciones:
                    self.ingredientes[ingrediente] -= 1
            return True
        return False
```

Clase Pedido:

Atributos:

- Cliente: referencia al cliente que realizó el pedido.

- Productos: lista de productos en el pedido.
- Estado: indica si el pedido está "pendiente", "en preparación" o "entregado".

Métodos:

- Agregar producto: añade un producto a la lista del pedido.
- Calcular total: suma los precios de los productos en el pedido.
- Actualizar estado: cambia el estado del pedido.

```python
class Pedido:
    def __init__(self, cliente):
        self.cliente = cliente
        self.productos = []
        self.estado = "pendiente"

    def agregar_producto(self, producto):
        self.productos.append(producto)

    def calcular_total(self):
        return sum(producto.precio for producto in self.productos)

    def actualizar_estado(self, estado):
        self.estado = estado
```

[8] ✓ 0.0s

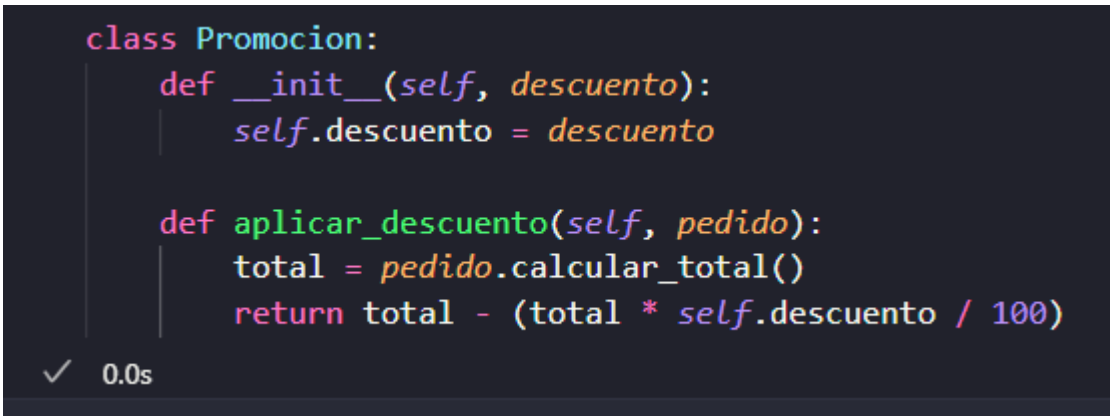
Clase Promoción:

Atributos:

- Descuento: porcentaje de descuento aplicado a un pedido.

Métodos:

- Aplicar descuento: reduce el total del pedido según el porcentaje de descuento.



Ejemplo de uso:

inventario = Inventario() inventario.agregar_ingrediente("leche de almendra", 5) inventario.agregar_ingrediente("azúcar", 5)

cliente = Cliente("Juan") pedido = Pedido(cliente) bebida = Bebida("Café", 2.5, "Grande", "Caliente", ["leche de almendra", "sin azúcar"])

pedido.agregar_producto(bebida)

if inventario.usar_ingredientes(pedido): cliente.realizar_pedido(pedido) print(f"Pedido realizado. Total: ${pedido.calcular_total()}") else: print("No hay suficientes ingredientes para este pedido.")

Link al proyecto:

https://github.com/netobunobi/ProgramacionAvanzada/blob/main/Cafeteria.ipynb

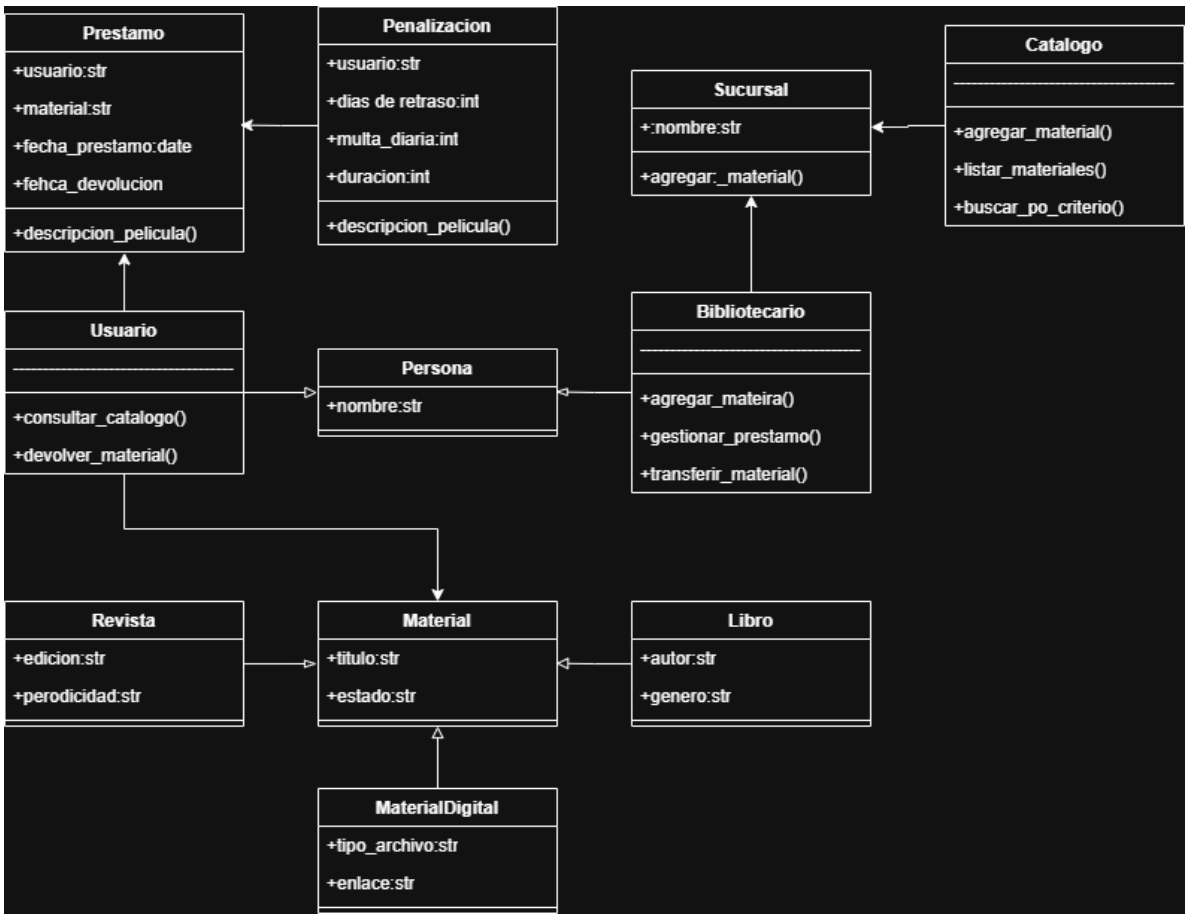
PRACTICA 1.3: BIBLIOTECA

Justificación:

Este sistema de gestión bibliotecaria se desarrolla con el objetivo de optimizar la administración de materiales, préstamos y sucursales, brindando una experiencia más eficiente tanto para usuarios como para bibliotecarios. Permite a los usuarios consultar el catálogo de manera sencilla, solicitar préstamos y gestionar devoluciones, mientras que los bibliotecarios pueden administrar el inventario, registrar préstamos y aplicar penalizaciones en caso de retrasos.

Además, el sistema facilita la organización interna al mantener un control preciso del estado de los materiales y permitir su transferencia entre sucursales, asegurando una mejor distribución de los recursos. La automatización de estos procesos no solo reduce errores humanos, sino que también mejora la disponibilidad y el acceso a la información, optimizando el funcionamiento general de la biblioteca y ofreciendo un servicio más ágil y eficaz.

DIAGRAMA UML



Clases usadas:

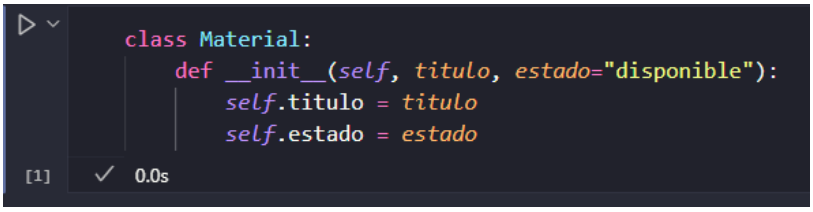
Clase Materia:

Atributos:

- Título: identifica el material.
- Estado: indica si el material está "disponible" o "prestado".

Métodos:

No incluye métodos adicionales, ya que sirve como base para otros materiales.



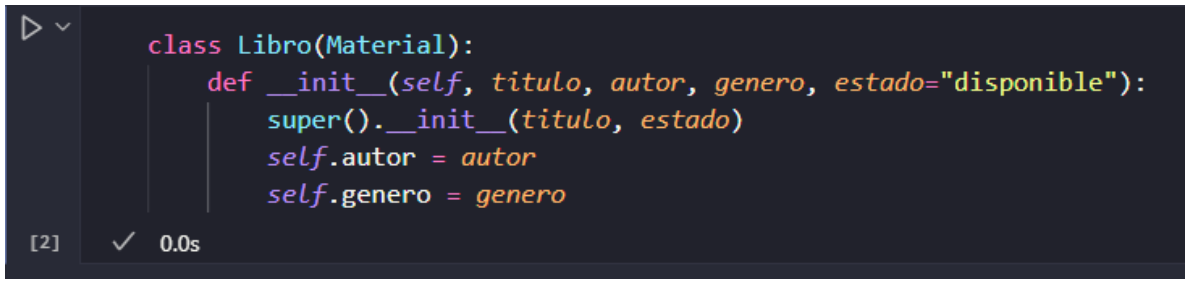
Clase Libro (hereda de Material):

Atributos:

- Autor: indica el autor del libro.
- Género: especifica el género literario.

Métodos:

No incluye métodos adicionales.



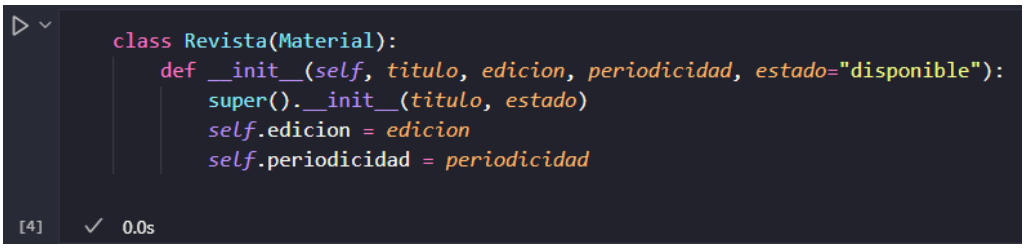
Clase Revista (hereda de Material):

Atributos:

- Edición: indica la edición de la revista.
- Periodicidad: define la frecuencia de publicación.

Métodos:

No incluye métodos adicionales.



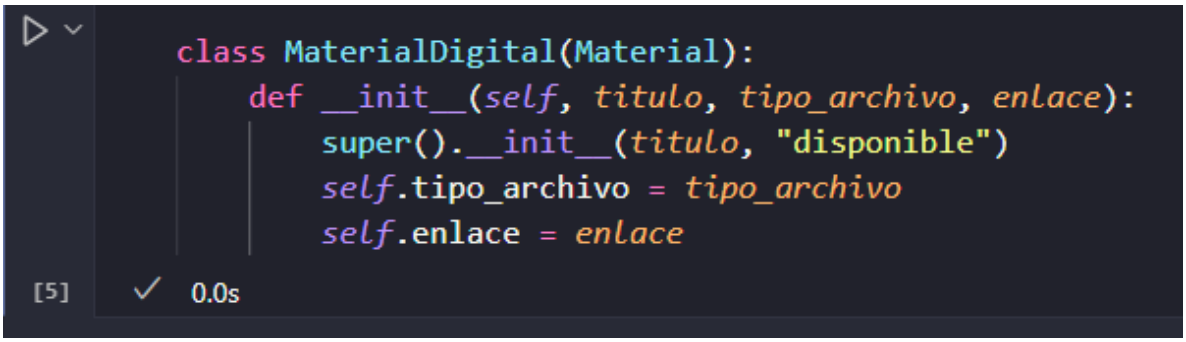
Clase MaterialDigital (hereda de Material):

Atributos:

- Tipo de archivo: especifica el formato digital.
- Enlace: URL para acceder al material digital.

Métodos:

No incluye métodos adicionales.



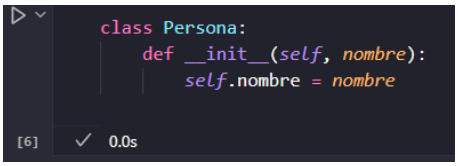
Clase Persona:

Atributos:

- Nombre: identifica a la persona.

Métodos:

No incluye métodos adicionales, ya que sirve como base para otras clases.



Clase Usuario (hereda de Persona):

Atributos:

- Préstamos: lista de materiales prestados al usuario.
- Penalizaciones: lista de penalizaciones acumuladas.

Métodos:

- Consultar catálogo: devuelve una lista de materiales disponibles.
- Devolver material: gestiona la devolución de un material y aplica penalización si hay retraso.

Clase Bibliotecario (hereda de Persona):

Atributos:

No incluye atributos adicionales.

Métodos:

- Agregar material: permite añadir nuevos materiales al catálogo.
- Gestionar préstamo: cambia el estado del material a "prestado" y lo asigna a un usuario.
- Transferir material: mueve un material de una sucursal a otra.

```python
class Bibliotecario(Persona):
    def __init__(self, nombre):
        super().__init__(nombre)

    def agregar_material(self, catalogo, material):
        catalogo.agregar_material(material)

    def gestionar_prestamo(self, prestamo):
        prestamo.material.estado = "prestado"
        prestamo.usuario.prestamos.append(prestamo)

    def transferir_material(self, material, sucursal_origen, sucursal_destino):
        if material in sucursal_origen.catalogo:
            sucursal_origen.catalogo.remove(material)
            sucursal_destino.catalogo.append(material)
```

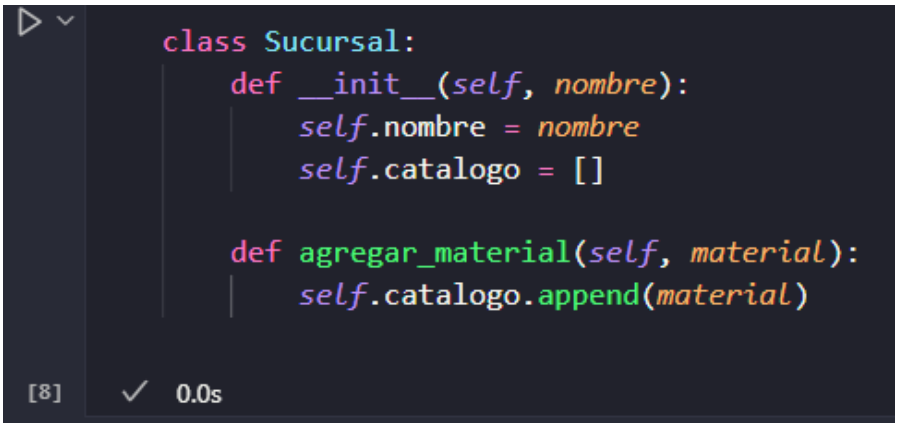
Clase Sucursal:

Atributos:

- Nombre: identifica la sucursal.
- Catálogo: lista de materiales disponibles en la sucursal.

Métodos:

- Agregar material: añade un material a la sucursal.



```python
class Sucursal:
    def __init__(self, nombre):
        self.nombre = nombre
        self.catalogo = []

    def agregar_material(self, material):
        self.catalogo.append(material)
```

[8] ✓ 0.0s

Clase Préstamo:

Atributos:

- Usuario: referencia al usuario que realiza el préstamo.
- Material: referencia al material prestado.
- Fecha de préstamo: indica la fecha en que se prestó el material.
- Fecha de devolución: indica la fecha límite de devolución.

Métodos:

No incluye métodos adicionales.

```
class Prestamo:
    def __init__(self, usuario, material, fecha_prestamo, fecha_devolucion):
        self.usuario = usuario
        self.material = material
        self.fecha_prestamo = fecha_prestamo
        self.fecha_devolucion = fecha_devolucion
```

[10] ✓ 0.0s

Clase Penalización:

Atributos:

- Usuario: referencia al usuario penalizado.
- Días de retraso: cantidad de días de retraso en la devolución.
- Multa total: monto total calculado con base en los días de retraso.

Métodos:

No incluye métodos adicionales.

```
class Penalizacion:
    def __init__(self, usuario, dias_retraso, multa_por_dia=5):
        self.usuario = usuario
        self.dias_retraso = dias_retraso
        self.multa_total = dias_retraso * multa_por_dia
```

[11] ✓ 0.0s

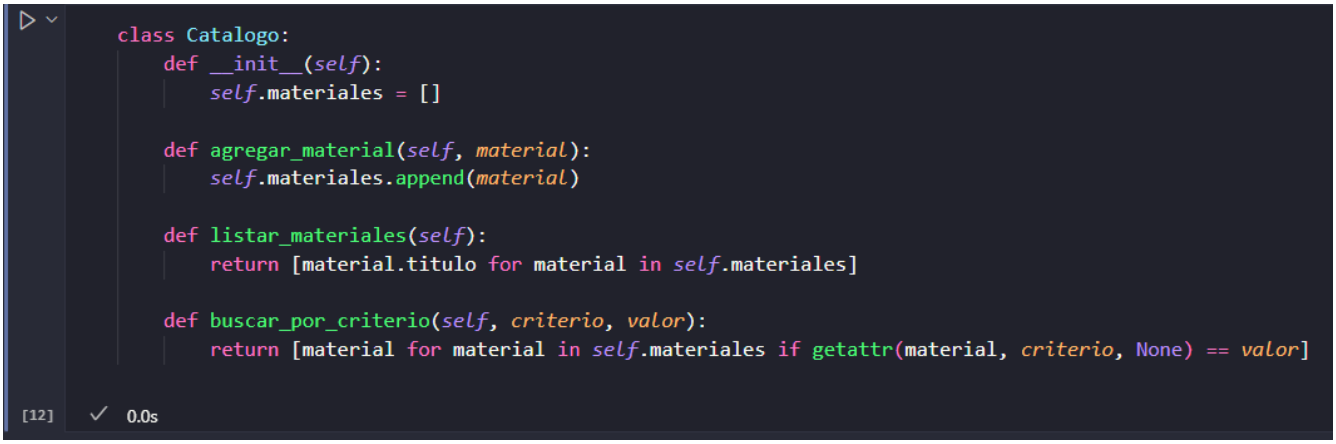
Clase Catálogo:

Atributos:

- Materiales: lista de materiales disponibles en el catálogo.

Métodos:

- Agregar material: añade un nuevo material al catálogo.
- Listar materiales: devuelve una lista de los títulos disponibles.
- Buscar por criterio: permite buscar materiales por un atributo específico.



```python
class Catalogo:
    def __init__(self):
        self.materiales = []

    def agregar_material(self, material):
        self.materiales.append(material)

    def listar_materiales(self):
        return [material.titulo for material in self.materiales]

    def buscar_por_criterio(self, criterio, valor):
        return [material for material in self.materiales if getattr(material, criterio, None) == valor]
```

[12] ✓ 0.0s

Ejemplo de uso:

from datetime import date, timedelta

catalogo = Catalogo() bibliotecario = Bibliotecario("Ana") usuario = Usuario("Carlos")

sucursal_a = Sucursal("Sucursal Centro") sucursal_b = Sucursal("Sucursal Norte")

libro = Libro("1984", "George Orwell", "Distopía") revista = Revista("National Geographic", "Edición 202", "Mensual") material_digital = MaterialDigital("Curso Python", "PDF", "www.ejemplo.com")

bibliotecario.agregar_material(catalogo, libro) bibliotecario.agregar_material(catalogo, revista) bibliotecario.agregar_material(catalogo, material_digital)

sucursal_a.agregar_material(libro) sucursal_a.agregar_material(revista) sucursal_b.agregar_material(material_digital)

print("Materiales en el catálogo:", usuario.consultar_catalogo(catalogo))

prestamo = Prestamo(usuario, libro, date(2024, 2, 1), date(2024, 2, 15)) bibliotecario.gestionar_prestamo(prestamo)

print(f"{usuario.nombre} ha tomado prestado: {[p.material.titulo for p in usuario.prestamos]}")

Simulación de devolución con retraso

fecha_devolucion_real = date(2024, 2, 20) usuario.devolver_material(prestamo, fecha_devolucion_real) print(f"{usuario.nombre} ha recibido una penalización de: {usuario.penalizaciones[0].multa_total} unidades monetarias")

Transferencia de material entre sucursales

bibliotecario.transferir_material(libro, sucursal_a, sucursal_b) print(f"El libro '1984' ha sido transferido a {sucursal_b.nombre}")

Link al proyecto:

https://github.com/netobunobi/ProgramacionAvanzada/blob/main/Biblioteca.ipynb