Benemérita Universidad Autónoma de Puebla

Facultad Ciencias de la Computación

Alumna: Samara Arias Gil

Matrícula: 202440031

Profesor: Jaime Alejandro Romero Sierra

Asignatura: Programación Avanzada

Puebla, Pue. A 12 de febrero de 2025

Programas UML

1. Biblioteca Digital

   CÓDIGO:

```python
class Material:
    def __init__(self, titulo, estado="disponible"):
        self.titulo = titulo
        self.estado = estado

    def cambiar_estado(self, nuevo_estado):
        self.estado = nuevo_estado

class Libro(Material):
    def __init__(self, titulo, autor, genero, estado="disponible"):
        super().__init__(titulo, estado)
        self.autor = autor
        self.genero = genero

class Revista(Material):
    def __init__(self, titulo, edicion, periodicidad, estado="disponible"):
        super().__init__(titulo, estado)
        self.edicion = edicion
        self.periodicidad = periodicidad

class MaterialDigital(Material):
    def __init__(self, titulo, tipo_archivo, enlace_descarga, estado="disponible"):
        super().__init__(titulo, estado)
        self.tipo_archivo = tipo_archivo
        self.enlace_descarga = enlace_descarga
```

✓ 0.0s Python

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

class Usuario(Persona):
    def __init__(self, nombre):
        super().__init__(nombre)
        self.materiales_prestados = []
        self.penalizaciones = []
        self.deuda_total = 0

    def devolver_material(self, material):
        for prestamo in self.materiales_prestados:
            if prestamo.material == material:
                prestamo.devuelto = True
                material.cambiar_estado("disponible")
                print(f"{self.nombre} ha devuelto {material.titulo}.")
                return
        print(f"{self.nombre} no tiene {material.titulo} en préstamo.")

class Bibliotecario(Persona):
    def agregar_material(self, material, sucursal):
        sucursal.catalogo.agregar_material(material)
        print(f"{self.nombre} ha agregado {material.titulo} a la sucursal {sucursal.nombre}.")

    def gestionar_prestamo(self, usuario, material, sucursal):
        if material.estado == "disponible":
            material.cambiar_estado("prestado")
            prestamo = Prestamo(usuario, material)
            usuario.materiales_prestados.append(prestamo)
            sucursal.prestamos.append(prestamo)
            print(f"{self.nombre} ha prestado {material.titulo} a {usuario.nombre}.")
        else:
            print(f"{material.titulo} no está disponible para préstamo.")

    def transferir_material(self, material, sucursal_origen, sucursal_destino):
        if material in sucursal_origen.catalogo.materiales:
            sucursal_origen.catalogo.materiales.remove(material)
            sucursal_destino.catalogo.materiales.append(material)
            print(f"{self.nombre} ha transferido {material.titulo} de {sucursal_origen.nombre} a {sucursal_destino.nombre}.")
        else:
            print(f"{material.titulo} no se encuentra en {sucursal_origen.nombre}.")

    def verificar_prestamos_vencidos(self, sucursal):
        for prestamo in sucursal.prestamos:
            prestamo.verificar_retraso()
```

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

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

    def agregar_material(self, material):
        self.materiales.append(material)
        print(f"Se ha agregado {material.titulo} al catálogo.")

    def buscar_materiales(self, titulo=None, autor=None, genero=None, tipo=None):
        resultados = []
        for material in self.materiales:
            if titulo and material.titulo.lower() == titulo.lower():
                resultados.append(material)
            elif isinstance(material, Libro) and autor and material.autor.lower() == autor.lower():
                resultados.append(material)
            elif isinstance(material, Libro) and genero and material.genero.lower() == genero.lower():
                resultados.append(material)
            elif isinstance(material, Revista) and tipo and material.periodicidad.lower() == tipo.lower():
                resultados.append(material)
            elif isinstance(material, MaterialDigital) and tipo and material.tipo_archivo.lower() == tipo.lower():
                resultados.append(material)

        return resultados
```

✓ 0.0s Pytho

```python
from datetime import datetime, timedelta

class Prestamo:
    def __init__(self, usuario, material):
        self.usuario = usuario
        self.material = material
        self.fecha_prestamo = datetime.now()
        self.fecha_devolucion = self.fecha_prestamo + timedelta(days=14)
        self.devuelto = False

    def verificar_retraso(self):
        if not self.devuelto and datetime.now() > self.fecha_devolucion:
            dias_retraso = (datetime.now() - self.fecha_devolucion).days
            multa = dias_retraso * 2
            Penalizacion(self.usuario, multa)
            print(f"{self.usuario.nombre} tiene una multa de ${multa} por retraso.")

class Penalizacion:
    def __init__(self, usuario, monto):
        self.usuario = usuario
        self.monto = monto
        self.aplicar_penalizacion()

    def aplicar_penalizacion(self):
        print(f"Se ha aplicado una penalización de ${self.monto} a {self.usuario.nombre}.")
        self.usuario.penalizaciones.append(self)
        self.usuario.deuda_total += self.monto
```

```python
libro1 = Libro("Cien años de soledad", "Gabriel García Márquez", "Realismo mágico")
revista1 = Revista("National Geographic", "Edición 2025", "Mensual")
material_digital1 = MaterialDigital("Curso de Python", "PDF", "www.descarga.com")

sucursal1 = Sucursal("Sucursal Centro")
sucursal2 = Sucursal("Sucursal Norte")

bibliotecario1 = Bibliotecario("Juan Pérez")
bibliotecario1.agregar_material(libro1, sucursal1)
bibliotecario1.agregar_material(revista1, sucursal1)
bibliotecario1.agregar_material(material_digital1, sucursal1)

usuario1 = Usuario("Ana Gómez")
bibliotecario1.gestionar_prestamo(usuario1, libro1, sucursal1)

prestamo1 = Prestamo(usuario1, libro1)
sucursal1.prestamos.append(prestamo1)

prestamo1.fecha_devolucion -= timedelta(days=16)

bibliotecario1.verificar_prestamos_vencidos(sucursal1)

usuario1.devolver_material(libro1)

bibliotecario1.transferir_material(revista1, sucursal1, sucursal2)
```

✓ 0.0s

```
Se ha agregado Cien años de soledad al catálogo.
Juan Pérez ha agregado Cien años de soledad a la sucursal Sucursal Centro.
Se ha agregado National Geographic al catálogo.
Juan Pérez ha agregado National Geographic a la sucursal Sucursal Centro.
Se ha agregado Curso de Python al catálogo.
Juan Pérez ha agregado Curso de Python a la sucursal Sucursal Centro.
Juan Pérez ha prestado Cien años de soledad a Ana Gómez.
Se ha aplicado una penalización de $4 a Ana Gómez.
Ana Gómez tiene una multa de $4 por retraso.
Ana Gómez ha devuelto Cien años de soledad.
Juan Pérez ha transferido National Geographic de Sucursal Centro a Sucursal Norte.
```

JUSTIFICACIÓN:

Las bibliotecas digitales y físicas requieren una gestión eficiente de materiales, usuarios y sucursales para garantizar el correcto funcionamiento del servicio de préstamos y devoluciones. Este sistema es necesario para organizar el inventario de libros, revistas y materiales digitales, facilitar la búsqueda de materiales en todas las sucursales, controlar los préstamos y devoluciones, incluyendo sanciones por retrasos, administrar la logística entre sucursales permitiendo transferencias de materiales y automatizar la gestión de penalizaciones para usuarios morosos. Este software asegurará una biblioteca eficiente, accesible y bien administrada.

Clases a ocupar

La clase Material representa cualquier material disponible en la biblioteca. Sus atributos incluyen titulo, que almacena el nombre del material, y estado, que indica si está "disponible" o "prestado". Su método cambiar_estado(nuevo_estado: str) permite modificar su estado. Esta clase es necesaria como base para los diferentes tipos de materiales y permite un manejo genérico del catálogo.

La clase Libro hereda de Material y representa un libro dentro de la biblioteca. Sus atributos adicionales incluyen autor, que almacena el nombre del autor, y genero, que

clasifica el libro según su temática. Esta clase es necesaria para registrar información específica sobre los libros.

La clase Revista también hereda de Material y representa una revista dentro de la biblioteca. Sus atributos adicionales incluyen edicion, que indica el número de edición, y periodicidad, que define la frecuencia de publicación (mensual, semanal, anual, etc.). Se diferencia de los libros por su periodicidad y edición, lo que justifica su existencia.

La clase MaterialDigital hereda de Material y representa materiales digitales accesibles en la biblioteca. Sus atributos adicionales incluyen tipo_archivo, que especifica el formato del archivo (PDF, EPUB, etc.), y enlace_descarga, que almacena la URL para acceder al material. Esta clase es necesaria para facilitar el acceso a recursos en línea y diferenciarlos de los materiales físicos.

La clase Persona es una clase base que representa a cualquier persona dentro del sistema. Su único atributo es nombre, que almacena el nombre de la persona. Esta clase es necesaria como base para diferenciar entre usuarios y bibliotecarios.

La clase Usuario hereda de Persona y representa a un usuario de la biblioteca. Sus atributos incluyen materiales_prestados, que es una lista de los materiales actualmente en préstamo, penalizaciones, que almacena un historial de sanciones, y deuda_total, que registra el monto total adeudado por penalizaciones. Sus métodos incluyen devolver_material(material: Material), que permite la devolución de un material prestado. Esta clase es necesaria para gestionar los préstamos y penalizaciones de los usuarios.

La clase Bibliotecario hereda de Persona y representa a un empleado de la biblioteca encargado de gestionar los préstamos y el inventario. Sus métodos incluyen agregar_material(material: Material, sucursal: Sucursal), que permite agregar nuevos materiales a una sucursal, gestionar_prestamo(usuario: Usuario, material: Material, sucursal: Sucursal), que facilita el proceso de préstamo de materiales, transferir_material(material: Material, sucursal_origen: Sucursal, sucursal_destino: Sucursal), que permite mover materiales entre sucursales, y verificar_prestamos_vencidos(sucursal: Sucursal), que revisa si hay materiales no devueltos en el plazo establecido. Esta clase es necesaria para administrar el inventario y las operaciones de préstamo.

La clase Prestamo relaciona a un usuario con un material en préstamo. Sus atributos incluyen usuario, que almacena la referencia al usuario que toma el préstamo, material, que guarda el material prestado, fecha_prestamo, que indica la fecha en la que se realizó el préstamo, fecha_devolucion, que almacena la fecha límite de devolución, y devuelto, que registra si el material ha sido regresado. Su método verificar_retraso() permite evaluar si un préstamo ha superado la fecha de devolución. Esta clase es necesaria para llevar un registro de los préstamos y evaluar posibles sanciones.

La clase Penalizacion representa una sanción aplicada a un usuario por la devolución tardía de un material. Sus atributos incluyen usuario, que almacena la referencia al usuario sancionado, y monto, que indica el valor de la multa aplicada. Su método aplicar_penalizacion() registra la penalización en la cuenta del usuario. Esta clase es necesaria para garantizar que los materiales sean devueltos a tiempo.

La clase Sucursal representa una sucursal de la biblioteca. Sus atributos incluyen nombre, que almacena el nombre de la sucursal, catalogo, que contiene el conjunto de materiales disponibles, y prestamos, que almacena los préstamos activos. Esta clase es necesaria para organizar los materiales en diferentes ubicaciones.

La clase Catalogo gestiona la búsqueda y organización de materiales en una sucursal. Sus atributos incluyen materiales, que almacena una lista de los materiales disponibles. Sus métodos incluyen agregar_material(material: Material), que permite agregar un nuevo material, y buscar_materiales(titulo: str = None, autor: str = None, genero: str = None, tipo: str = None), que facilita la búsqueda de materiales según diferentes criterios. Esta clase es necesaria para mejorar la accesibilidad y consulta del inventario en cada sucursal.

UML: https://lucid.app/lucidchart/7296b49c-47a0-49a4-9dee-84ef15656054/edit?viewport_loc=-1213%2C35%2C4189%2C2253%2CHWEp-vi-RSFO&invitationId=inv_a7d307ad-50b1-4b3b-b1a1-08d67f412a4f

2. Cafetería

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

```python
class Cliente(Persona):
    def __init__(self, nombre):
        super().__init__(nombre)
        self.historial = []

    def realizar_pedido(self, pedido):
        self.historial.append(pedido)
        print(f"{self.nombre} ha realizado un nuevo pedido.")

    def consultar_historial(self):
        print(f"Historial de pedidos de {self.nombre}:")
        for i, pedido in enumerate(self.historial, 1):
            print(f"{i}. {pedido}")
```

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

```python
class ProductoBase:
    def __init__(self, nombre, precio):
        self.nombre = nombre
        self.precio = precio

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

```python
class Bebida(ProductoBase):
    def __init__(self, nombre, precio, tamaño, tipo, opciones=None):
        super().__init__(nombre, precio)
        self.tamaño = tamaño
        self.tipo = tipo
        self.opciones = opciones or []

    def __str__(self):
        opciones_str = ", ".join(self.opciones) if self.opciones else "Ninguna"
        return f"Bebida: {self.nombre} ({self.tipo}, {self.tamaño}) - Opciones: {opciones_str} - ${self.precio}"
```

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

    def __str__(self):
        caracteristicas = []
        if self.es_vegano:
            caracteristicas.append("Vegano")
        if self.sin_gluten:
            caracteristicas.append("Sin gluten")
        descripcion = ", ".join(caracteristicas) if caracteristicas else "Regular"
        return f"Postre: {self.nombre} ({descripcion}) - ${self.precio}"
```

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

    def agregar_ingrediente(self, nombre, cantidad):
        self.ingredientes[nombre] = self.ingredientes.get(nombre, 0) + cantidad
        print(f"Se agregaron {cantidad} unidades de {nombre} al inventario.")

    def verificar_disponibilidad(self, ingredientes_necesarios):
        for ingrediente, cantidad in ingredientes_necesarios.items():
            if self.ingredientes.get(ingrediente, 0) < cantidad:
                return False
        return True

    def usar_ingredientes(self, ingredientes_necesarios):
        if self.verificar_disponibilidad(ingredientes_necesarios):
            for ingrediente, cantidad in ingredientes_necesarios.items():
                self.ingredientes[ingrediente] -= cantidad
            print("Ingredientes usados correctamente.")
            return True
        else:
            print("No hay suficientes ingredientes en el inventario.")
            return False
```

Python

```python
class Pedido:
    def __init__(self, cliente):
        self.cliente = cliente
        self.productos = []
        self.estado = "Pendiente"
        self.total = 0

    def agregar_producto(self, producto, inventario, ingredientes_necesarios={}):
        if self.estado == "Rechazado":
            print("El pedido ha sido rechazado, no se pueden agregar más productos.")
            return False

        if inventario.usar_ingredientes(ingredientes_necesarios):
            self.productos.append(producto)
            self.total += producto.precio
            print(f"{producto.nombre} agregado al pedido de {self.cliente.nombre}.")
        else:
            print(f"No se pudo agregar {producto.nombre} por falta de ingredientes.")
            self.estado = "Rechazado"
            self.productos = []
            self.total = 0
            return False
        return True

    def cambiar_estado(self, nuevo_estado):
        self.estado = nuevo_estado
        print(f"Estado del pedido cambiado a: {nuevo_estado}")

    def entregar_pedido(self):
        if self.estado == "Pendiente" and self.productos:
            self.estado = "Entregado"
            print(f"El pedido de {self.cliente.nombre} ha sido entregado.")
        else:
            print("El pedido no puede ser entregado porque no está completo o ya fue rechazado.")

    def __str__(self):
        productos_str = "\n".join([f"- {p.nombre}: ${p.precio}" for p in self.productos])
        return f"Pedido de {self.cliente.nombre} - Estado: {self.estado}\n{productos_str}\nTotal: ${self.total}"
```

```python
class Promocion:
    def __init__(self, descripcion, descuento):
        self.descripcion = descripcion
        self.descuento = descuento

    def aplicar_descuento(self, pedido):
        descuento_aplicado = pedido.total * (self.descuento / 100)
        pedido.total -= descuento_aplicado
        print(f"Se aplicó la promoción: {self.descripcion} - Descuento: ${descuento_aplicado:.2f}")
```

```
inventario = Inventario()
inventario.agregar_ingrediente("Leche", 10)
inventario.agregar_ingrediente("Café", 15)
inventario.agregar_ingrediente("Azúcar", 8)


cliente1 = Cliente("Ana Pérez")
empleado1 = Empleado("Luis Martínez", "Barista")


bebida1 = Bebida("Café Latte", 50, "Mediano", "Caliente", ["Leche de almendra", "Sin azúcar"])
postre1 = Postre("Brownie", 35, es_vegano=False, sin_gluten=True)


pedido1 = Pedido(cliente1)
pedido1.agregar_producto(bebida1, inventario, {"Leche": 1, "Café": 1})
pedido1.agregar_producto(postre1, inventario)


print(pedido1)


promo1 = Promocion("Descuento para clientes frecuentes", 10)
promo1.aplicar_descuento(pedido1)


print(pedido1)


pedido1.cambiar_estado("En preparación")
pedido1.cambiar_estado("Entregado")


cliente1.realizar_pedido(pedido1)
cliente1.consultar_historial()
```

UML: https://lucid.app/lucidchart/3b75ff7f-6ad5-48ec-88ca-59cf99d72b01/edit?viewport_loc=-783%2C308%2C2844%2C1530%2CHWEp-vi-RSFO&invitationId=inv_e72376b4-5d7d-4171-aa98-34b3642d81f4

JUSTIFICACIÓN:

La justificación del programa radica en la necesidad de gestionar de manera eficiente y organizada el proceso de pedidos en un entorno como un restaurante, tienda o cualquier otro establecimiento que ofrezca productos a sus clientes. En este contexto, el sistema debe cubrir varios aspectos clave, como la interacción con los clientes, el control del inventario, la gestión de productos y la aplicación de descuentos, de manera automatizada y sencilla.

Primero, se necesita un control efectivo de los **clientes** y sus **pedidos**. Los clientes tienen un historial de compras que se debe registrar y consultar de manera eficiente para asegurar una experiencia de usuario fluida y personalizada. La clase **Cliente** permite hacer esto al almacenar el historial de pedidos, facilitando tanto la creación de nuevos pedidos como la consulta de los anteriores.

En paralelo, la **gestión de empleados** es necesaria para asignar responsabilidades en el sistema, como tomar pedidos o preparar productos. Esto se logra mediante la clase

Empleado, que distingue a cada empleado por su rol, permitiendo así una correcta distribución de las tareas en el proceso de atención al cliente.

La clase **ProductoBase** y sus subclases **Bebida** y **Postre** permiten estructurar y organizar los productos que se ofrecen, adaptándose a las diversas características y opciones de cada tipo de producto. Esto facilita su administración y visualización, permitiendo que el cliente vea exactamente lo que está comprando y qué opciones tiene.

El **inventario** es un aspecto clave para garantizar que siempre haya suficiente stock de ingredientes para preparar los productos. La clase **Inventario** asegura que los ingredientes estén disponibles antes de aceptar un pedido, evitando situaciones en las que un pedido no pueda cumplirse debido a la falta de productos. Además, al usar ingredientes al agregar productos al pedido, se mantiene un control dinámico de los recursos disponibles.

La clase **Pedido** es el centro del proceso de compra, donde se gestionan los productos que el cliente selecciona, se calcula el total a pagar y se actualiza el estado del pedido (pendiente, entregado, rechazado). Esto permite una gestión integral del flujo de trabajo dentro del sistema, asegurando que el pedido sea procesado correctamente desde su creación hasta su entrega.

Finalmente, las **promociones** son una parte esencial para ofrecer descuentos o beneficios adicionales a los clientes, incentivando la compra y mejorando la competitividad del establecimiento. La clase **Promocion** permite aplicar descuentos directamente a los pedidos, lo cual es útil para campañas especiales, descuentos por volumen o promociones limitadas.

En resumen, el programa es esencial para automatizar y gestionar de manera eficiente todos los aspectos relacionados con los pedidos, desde la interacción con el cliente hasta la entrega final del pedido, pasando por el control de inventario y la aplicación de promociones. Esto no solo mejora la eficiencia operativa, sino que también optimiza la experiencia del cliente, asegurando que sus pedidos se realicen de manera rápida, precisa y satisfactoria.

3. Sistema de reservas para Cine

```python
class Persona():

    lista=[]

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

    def registrar(self):
        Persona.lista.append(self)
        print(f"La persona {self.nombre} ha sido registrada con el correo {self.correo}")

    def actualizar_datos(self,nombre,correo):
        self.nombre=nombre
        self.correo=correo
        print(f"Los datos han sido actualizados")

    @classmethod
    def personas_registradas(cls):
        print("Personas registradas")
        for Persona in cls.lista:
            print(f"-{Persona.nombre} - {Persona.correo}")
```

```python
class Usuario(Persona):
    def __init__(self, nombre, correo):
        super().__init__(nombre, correo)
        self.historial_reservas = []

    def reservar(self, funcion, asientos):
        if asientos <= funcion.asientos_disponibles:
            funcion.asientos_disponibles -= asientos
            self.historial_reservas.append({"funcion": funcion, "asientos": asientos})
            print(f"Reserva realizada para '{funcion.pelicula.titulo}' en la sala {funcion.sala.identificador}.")
        else:
            print("No hay suficientes asientos disponibles.")

    def cancelar_reserva(self, funcion):
        reserva = next((r for r in self.historial_reservas if r["funcion"] == funcion), None)
        if reserva:
            funcion.asientos_disponibles += reserva["asientos"]
            self.historial_reservas.remove(reserva)
            print(f"Reserva cancelada para '{funcion.pelicula.titulo}'.")
        else:
            print("No tienes una reserva para esta función.")
```

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

    def agregar_funcion(self, funcion):
        print(f"Función agregada: {funcion.pelicula.titulo} a las {funcion.hora} en la sala {funcion.sala.identificador}.")

    def modificar_promocion(self, promocion, nuevo_descuento, nuevas_condiciones):
        promocion.descuento = nuevo_descuento
        promocion.condiciones = nuevas_condiciones
        print(f"Promoción modificada: {nuevo_descuento}% de descuento. {nuevas_condiciones}.")
```

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

    def descripcion(self):
        print(f"El edificio tiene tamaño {self.capacidad} y tiene id {self.identificador}")
```

```python
class Sala(Espacio):
    def __init__(self,capacidad,identificador,tipo):
        super().__init__(capacidad,identificador)
        self.tipo=tipo
        self.disponibilidad=True

    def Consultardisponibilidad(self):
        if self.disponibilidad:
            print("La sala esta disponible")
        else:
            print("La sala esta ocupada")
```

```python
class ZonaComida(Espacio):
    def __init__(self, capacidad, identificador):
        super().__init__(capacidad, identificador)
        self.productos = {}

    def agregar_producto(self, nombre, precio):
        self.productos[nombre] = precio
        print(f"Producto agregado: {nombre} - ${precio:.2f}")

    def mostrar_productos(self):
        print(f"Productos disponibles en {self.identificador}:")
        for nombre, precio in self.productos.items():
            print(f"- {nombre}: ${precio:.2f}")

    def vender_producto(self, nombre, cantidad):
        if nombre in self.productos:
            total = self.productos[nombre] * cantidad
            print(f"Venta realizada: {cantidad}x {nombre} - Total: ${total:.2f}")
        else:
            print(f"El producto '{nombre}' no está disponible.")
```

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

```python
class Funcion:
    def __init__(self, pelicula, sala, hora, asientos_disponibles=None):
        self.pelicula = pelicula
        self.sala = sala
        self.hora = hora
        self.asientos_disponibles = asientos_disponibles or sala.capacidad
```

```python
class Promocion:
    def __init__(self, descuento, condiciones):
        self.descuento = descuento
        self.condiciones = condiciones

    def mostrar(self):
        print(f"Promoción: {self.descuento}% de descuento. Condiciones: {self.condiciones}")
```

```
pelicula1 = Pelicula("Matrix", "Ciencia Ficción", 136)
pelicula2 = Pelicula("Titanic", "Drama/Romance", 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("Ana Pérez", "ana.perez@email.com")
empleado1 = Empleado("Luis Martínez", "luis.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.")

zona_comida1 = ZonaComida(20, "Snack Bar")
zona_comida1.agregar_producto("Palomitas", 100)
zona_comida1.agregar_producto("Refresco", 60)
zona_comida1.agregar_producto("Chocolate", 15)

zona_comida1.mostrar_productos()
zona_comida1.vender_producto("Nachos", 2)
zona_comida1.vender_producto("Palomitas",3)


Persona.personas_registradas()
```

UML: https://lucid.app/lucidchart/f94ff332-b762-4982-94fa-cbe60e102e01/edit?view_items=.DpSEauKU4cQ&invitationId=inv_37a2e220-d42e-4525-996a-7727c280c042

JUSTIFICACIÓN:

El programa es necesario para gestionar de manera eficiente las reservas, funciones de cine, productos en zonas de comida y la interacción entre usuarios y empleados dentro de un establecimiento como un cine o teatro. Permite a los usuarios registrar sus datos, realizar y cancelar reservas, mientras que los empleados tienen la capacidad de administrar funciones, modificar promociones y gestionar el inventario de productos. Las clases están diseñadas para reflejar las entidades clave en este contexto. La clase **Persona** es la base para gestionar los datos comunes a usuarios y empleados, mientras que **Usuario** y **Empleado** amplían las funcionalidades para cubrir las necesidades específicas de cada tipo de persona. Las clases **Sala** y **ZonaComida** permiten gestionar los espacios disponibles, controlando la disponibilidad de las salas y los productos que se venden en las zonas de comida. **Pelicula** y **Funcion** representan las películas y funciones, respectivamente, facilitando la creación de reservas y la asignación de asientos. Finalmente, la clase **Promocion** permite aplicar descuentos a los usuarios bajo ciertas condiciones, agregando flexibilidad y atracción al sistema de ventas. En conjunto, estas clases forman un sistema integral que automatiza y organiza todas las operaciones dentro de un entorno de entretenimiento.