Benemérita Universidad Autónoma de Puebla

Facultad Ciencias de la Computación

Alumno: Reyes Fortis Yael Alejandro

Matrícula: 202454916

Profesor: Jaime Alejandro Romero Sierra

Asignatura: Programación Avanzada

Puebla, Pue. A 12 de Febrero de 2025

Programas

1. Biblioteca Digital

   CÓDIGO:

```python
class Libro(Material):
    def __init__(self, titulo, autor, genero):
        super().__init__(titulo)
        self.autor = autor
        self.genero = genero

    def __str__(self):
        return f"Libro: {self.titulo} por {self.autor} ({self.genero})"

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

    def __str__(self):
        return f"Revista: {self.titulo}, edicion: {self.edicion} ({self.periodicidad})"

class MaterialDigital(Material):
    def __init__(self, titulo, tipo_archivo, enlace):
        super().__init__(titulo)
        self.tipo_archivo = tipo_archivo
        self.enlace = enlace

    def __str__(self):
        return f"Material digital: {self.titulo} ({self.tipo_archivo}) enlace: {self.enlace}"
```

```python
from datetime import datetime, timedelta

#Clase para gestionar el material de la biblioteca
class Material:
    def __init__(self, titulo, estado='disponible'):
        self.titulo = titulo
        self.estado = estado

    def prestar(self):
        if self.estado == 'disponible':
            self.estado = 'prestado'
            return True
        return False

    def devolver(self):
        self.estado = 'disponible'

    def __str__(self):
        return f"{self.titulo} ({self.estado})"


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

    def __str__(self):
        return f"Libro: {self.titulo} por {self.autor} ({self.genero})"
```

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

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

    def transferir_material(self, material, otra_sucursal):
        if self.sucursal.eliminar_material(material):
            otra_sucursal.agregar_material(material)
            print(f'{self.nombre} transfirio "{material.titulo}" a "{otra_sucursal.nombre}')



#Clase para gestionar las sucursales
class Sucursal:
    def __init__(self, nombre):
        self.nombre = nombre
        self.catalogo = []


    def agregar_material(self, material):
        self.catalogo.append(material)
        print(f'{material.titulo} fue agregado a la biblioteca {self.nombre}')


    def eliminar_material(self, material):
        if material in self.catalogo:
            self.catalogo.remove(material)
            print(f'El {material.titulo} fue eliminado de la biblioteca {self.nombre}')
            return True
        print(f'No se encontro {material.titulo} en {self.nombre}')
        return False


    def mostrar_catalogo(self):
        print(f'Catalogo de la biblioteca "{self.nombre}":')
        if self.catalogo:
            for i, material in enumerate(self.catalogo, 1):
                print(f'{i}, {material}')
        else:
            print('La biblioteca no tiene materiales.')
```

```python
#Clase para gestionar al usuario y bibliotecario
class Persona:
    def __init__(self, nombre, apellido):
        self.nombre = nombre
        self.apellido = apellido


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

    def pedir_prestado(self,material,fecha_prestamo,dias_prestamo=7):
        if material.prestar():
            fecha_devolucion = fecha_prestamo + timedelta(days=dias_prestamo)
            self.materiales_prestados.append((material, fecha_prestamo, fecha_devolucion))
            print(f"{self.nombre} pidio prestado '{material.titulo}' el {fecha_prestamo.strftime('%Y-%m-%d')}, debe devolverlo antes del {fecha_devolucion.strftime('%Y-%m-%d')}")
            return True
        print(f"No se pudo prestar '{material.titulo}', no esta disponible.")
        return False

    def devolver_material(self, material):
        for i, (j, fecha_prestamo, fecha_devolucion) in enumerate(self.materiales_prestados):
            if j == material:
                if datetime.now() > fecha_devolucion:
                    dias_retraso = (datetime.now() - fecha_devolucion).days
                    self.penalizaciones += dias_retraso
                    print(f"{self.nombre} ha devuelto '{material.titulo}' con {dias_retraso} dias de retraso. Penalizaciones acumuladas: {self.penalizaciones}")
                else:
                    print(f"{self.nombre} ha devuelto '{material.titulo}' a tiempo.")
                material.devolver()
                self.materiales_prestados.pop(i)
                return True
        print(f"'{material.titulo}' no esta en la lista de materiales prestados de {self.nombre}.")
        return False
```

```python
# Clase para gestionar penalizaciones
class Penalizacion:
    def __init__(self, usuario, dias_retraso):
        self.usuario = usuario
        self.dias_retraso = dias_retraso
        self.multa = dias_retraso * 1  # Multa de $1 por día de retraso

    def __str__(self):
        return f"Penalización para {self.usuario.nombre}: {self.dias_retraso} días de retraso, multa de ${self.multa}"



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 días
usuario1.pedir_prestado(libro1, fecha_prestamo)
print(penalizacion1)

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

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

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

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.

2. Cafetería

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

class Cliente(Persona):
    def __init__(self, nombre):
        super().__init__(nombre)
        self.historial_pedidos = []
        self.puntos_fidelidad = 0

    def realizar_pedido(self, pedido, inventario, promocion=None):
        if pedido.verificar_disponibilidad(inventario):
            if promocion and self.puntos_fidelidad >= 50:
                promocion.aplicar_descuento(pedido)
            self.historial_pedidos.append(pedido)
            self.puntos_fidelidad += 10
            pedido.actualizar_stock(inventario)
            pedido.actualizar_estado("En preparación")
            print("------------------------------------------------------------")
            print("Productos:")
            for producto in pedido.productos:
                print(f"- {producto.nombre}: ${producto.precio}")
            print(f"Total con descuento: ${pedido.calcular_total()}")
            print("------------------------------------------------------------")
        else:
            print("------------------------------------------------------------")
            print("No hay suficiente stock para realizar el pedido.")
            print("------------------------------------------------------------")

    def consultar_historial(self):
        print("------------------------------------------------------------")
        print(f"Historial pedidos de {self.nombre}")
        print("------------------------------------------------------------")
        for pedido in self.historial_pedidos:
            print(f"Pedido con total: ${pedido.calcular_total()} - Estado: {pedido.estado}")
            print("------------------------------------------------------------")

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

    def actualizar_inventario(self, inventario, ingrediente, cantidad):
        inventario.agregar_ingrediente(ingrediente, cantidad)
        print("------------------------------------------------------------")
        print(f"Se han añadido {cantidad} unidades de {ingrediente}")
        print("------------------------------------------------------------")
```

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

class Bebida(ProductoBase):
    def __init__(self, nombre, precio, tamaño, tipo, personalizacion={}):
        super().__init__(nombre, precio)
        self.tamaño = tamaño
        self.tipo = tipo
        self.personalizacion = personalizacion
        self.ingredientes = {"Agua": 1}
        self.ingredientes.update(personalizacion)

    def descripcion(self):
        print("------------------------------------------------------------")
        print(f"Nombre: {self.nombre}")
        print(f"Precio: ${self.precio}")
        print(f"Tamaño: {self.tamaño}")
        print("Ingredientes:")
        for ingrediente in self.ingredientes:
            print(f"- {ingrediente}")
        print("------------------------------------------------------------")

class Postre(ProductoBase):
    def __init__(self, nombre, precio, personalizacion={}, vegano=False, sin_gluten=False):
        super().__init__(nombre, precio)
        self.vegano = vegano
        self.sin_gluten = sin_gluten
        self.personalizacion = personalizacion
        self.ingredientes = {"Harina": 1}
        self.ingredientes.update(personalizacion)

    def descripcion(self):
        print("------------------------------------------------------------")
        print(f"Nombre: {self.nombre}")
        print(f"Precio: ${self.precio}")
        print(f"Vegano: {'Si' if self.vegano else 'No'}")
        print(f"Sin gluten: {'Si' if self.sin_gluten else 'No'}")
        print("Ingredientes:")
        for ingrediente in self.ingredientes:
            print(f"- {ingrediente}")
        print("------------------------------------------------------------")
```

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

    def agregar_ingrediente(self, ingrediente, cantidad):
        self.stock[ingrediente] = self.stock.get(ingrediente, 0) + cantidad

    def verificar_disponibilidad(self, ingredientes):
        faltantes = [ing for ing, cantidad in ingredientes.items() if self.stock.get(ing, 0) < cantidad]
        if faltantes:
            print("------------------------------------------------------------")
            print(f"Faltan ingredientes: {', '.join(faltantes)}")
            print("------------------------------------------------------------")
            return False
        return True

    def actualizar_stock(self, ingredientes):
        for ing, cantidad in ingredientes.items():
            if ing in self.stock:
                self.stock[ing] -= cantidad

    def consultar_inventario(self):
        print("------------------------------------------------------------")
        print("Inventario actual:")
        for ing, cantidad in self.stock.items():
            print(f"- {ing}: {cantidad} unidades")
        print("------------------------------------------------------------")


class Pedido:
    def __init__(self, cliente):
        self.cliente = cliente
        self.productos = []
        self.estado = "Pendiente"
        self.total = 0
        print("------------------------------------------------------------")
        print(f"Se ha añadido un nuevo pedido para {self.cliente.nombre}")
        print("------------------------------------------------------------")


    def agregar_producto(self, producto):
        self.productos.append(producto)
        self.total += producto.precio
        print("------------------------------------------------------------")
        print(f"Se ha añadido {producto.nombre} con costo de ${producto.precio} al pedido de {self.cliente.nombre}")
        print("------------------------------------------------------------")
```

```
    def calcular_total(self):
        return self.total

    def actualizar_estado(self, nuevo_estado):
        self.estado = nuevo_estado
        print("------------------------------------------------------------")
        print(f"Nuevo estado del pedido: {self.estado}")
        print("------------------------------------------------------------")

    def verificar_disponibilidad(self, inventario):
        return all(inventario.verificar_disponibilidad(prod.ingredientes) for prod in self.productos)

    def actualizar_stock(self, inventario):
        for prod in self.productos:
            inventario.actualizar_stock(prod.ingredientes)

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

    def aplicar_descuento(self, pedido):
        pedido.total *= (1 - self.descuento / 100)


inventario = Inventario()
inventario.agregar_ingrediente("Café", 20)
inventario.agregar_ingrediente("Agua", 20)
inventario.agregar_ingrediente("Leche de almendra", 20)
inventario.agregar_ingrediente("Leche deslactosada", 20)
inventario.agregar_ingrediente("Leche entera", 20)
inventario.agregar_ingrediente("Azúcar", 20)
inventario.agregar_ingrediente("Chocolate", 20)
inventario.agregar_ingrediente("Harina", 20)
inventario.agregar_ingrediente("Fruta", 20)
inventario.agregar_ingrediente("Limon", 20)
inventario.consultar_inventario()
```

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}")

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}.")

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}")

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")

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

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.")

Persona.personas_registradas()
```

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.