Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

Ciencia de datos

Vallarin Lopez Carlos

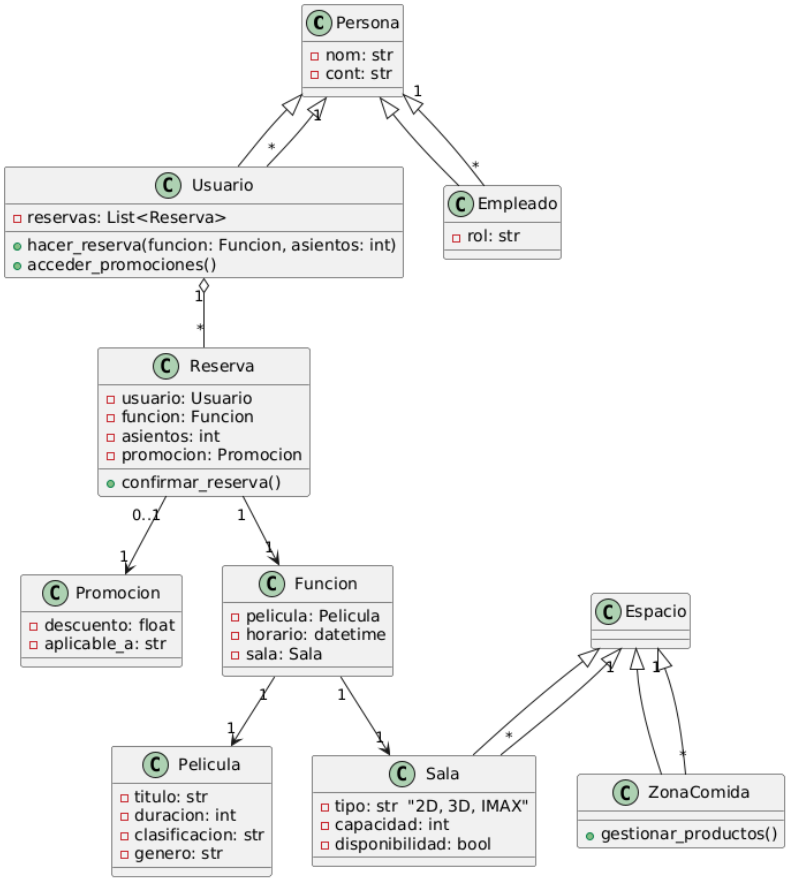
Programación Avanzada

Programación Orientada a Objetos

Tarea Primer Parcial

Primer programa: Sistema de reserva para cines

Diagrama UML



Código de Python

```python
from datetime import datetime
from typing import List, Optional

class Persona:
    list = []

    def __init__(self, nombre, contacto):
        self.nom = nombre
        self.cont = contacto

    def Registrar(self):
        Persona.list.append(self)
        print(f" {self.nom}, con el contacto {self.cont}")

    def actualizar_datos(self, nombre, correo):
        self.nom = nombre
        self.cont = correo
        print("Los datos han sido actualizados")
    @classmethod
    def personas_registradas(cls):
        print("Personas registradas:")
        for persona in cls.list:
            print(f"- {persona.nom} - {persona.cont}")
```

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

    def hacer_reserva(self, funcion, asientos):
        if funcion.verificar_disponibilidad(asientos):
            reserva = Reserva(self, funcion, asientos)
            self.reservas.append(reserva)
            funcion.reservar_asientos(asientos)
            print(f"Reserva realizada para la función {funcion.movie.title} en los asientos {asientos}.")
        else:
            print("No hay asientos disponibles para reserva.")

    def cancelar_reserva(self, reserva):
        if reserva in self.reservas:
            self.reservas.remove(reserva)
            reserva.funcion.liberar_asientos(reserva.asientos)
            print("Reserva cancelada.")
        else:
            print("Reserva no encontrada.")
```

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

    def agregar_funcion(self, funcion, funciones):
        funciones.append(funcion)
        print(f"Función {funcion.movie.title} agregada.")

    def agregar_pelicula(self, pelicula, peliculas):
        peliculas.append(pelicula)
        print(f"Película {pelicula.title} agregada.")

    def agregar_promocion(self, promocion, promociones):
        promociones.append(promocion)
        print(f"Promoción {promocion.codigo} agregada.")
```

```python
class espacio:
    def __init__(self, tamaño, identificador):
        self.tam = tamaño
        self.id = identificador

    def descripcion(self):
        print(f"La sala mide {self.tam}, se identifica con {self.id}")
```

```python
class Sala(espacio):
    def __init__(self, tamaño, identificador, tipo):
        super().__init__(tamaño, identificador)
        self.tipo = tipo
        self.asientos_ocupados = []

    def verificar_disponibilidad(self, asientos):
        return all(asiento not in self.asientos_ocupados for asiento in asientos)

    def reservar_asientos(self, asientos):
        self.asientos_ocupados.extend(asientos)

    def liberar_asientos(self, asientos):
        for asiento in asientos:
            if asiento in self.asientos_ocupados:
                self.asientos_ocupados.remove(asiento)

    def consultar_disponibilidad(self):
        if self.verificar_disponibilidad(range(1, self.tam + 1)):
            print("La sala está disponible.")
        else:
            print("La sala no está disponible.")
```

```python
class zona_de_comida(espacio):
    menu = []

    def __init__(self, tamaño, identificador, productos, precios):
        super().__init__(tamaño, identificador)
        self.product = productos
        self.price = precios

    def agregar_al_menu(self):
        zona_de_comida.menu.append(self)
        print(f"En el menú tenemos {self.product} a {self.price}")
```

```python
class Pelicula:
    def __init__(self, titulo, duracion, clasificacion, genero):
        self.title = titulo
        self.dur = duracion
        self.clas = clasificacion
        self.gen = genero

    def detalles(self):
        print(f"La película {self.title}, con duración de {self.dur}, clasificación {self.clas}, es {self.gen}")
```

```python
class Funcion:
    def __init__(self, hora, sala, pelicula):
        self.hora = hora
        self.sala = sala
        self.movie = pelicula

    def verificar_disponibilidad(self, asientos):
        return self.sala.verificar_disponibilidad(asientos)

    def reservar_asientos(self, asientos):
        self.sala.reservar_asientos(asientos)

    def liberar_asientos(self, asientos):
        self.sala.liberar_asientos(asientos)
```

Python

```python
class Reserva:
    def __init__(self, usuario, funcion, asientos):
        self.usuario = usuario
        self.funcion = funcion
        self.asientos = asientos

    def confirmar_reserva(self):
        print(f"Su reserva ha sido confirmada para la función {self.funcion.movie.title} en los asientos {self.asientos}.")

    def cancelar_reserva(self):
        self.usuario.cancelar_reserva(self)
```

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

    def aplicar_descuento(self, precio):
        return precio * (1 - self.descuento)
```

```python
p1 = Persona("Carlos", "carlosvallarin@gmail.com")
p2 = Persona("Gabito", "gabito@gmail.com")
p1.Registrar()
p2.Registrar()

sala_imax = Sala(100, "Sala IMAX", "IMAX")
pelicula = Pelicula("Avengers", 148, "PG-13", "Ciencia Ficción")
funcion = Funcion(datetime(2023, 10, 15, 20, 0), sala_imax, pelicula)

usuario = Usuario("Juan", "juan@example.com")
usuario.hacer_reserva(funcion, [1, 2, 3])

empleado = Empleado("Diego", "diego@example.com", "Taquillero")
empleado.agregar_funcion(funcion, [])
empleado.agregar_pelicula(pelicula, [])

promocion = Promocion("DESC10", 0.1, datetime(2023, 10, 20))
empleado.agregar_promocion(promocion, [])

Persona.personas_registradas()
```

```
 Carlos, con el contacto carlosvallarin@gmail.com
 Gabito, con el contacto gabito@gmail.com
Reserva realizada para la función Avengers en los asientos [1, 2, 3].
Función Avengers agregada.
Película Avengers agregada.
Promoción DESC10 agregada.
Personas registradas:
- Carlos - carlosvallarin@gmail.com
- Gabito - gabito@gmail.com
```

Necesidad del programa:

Este sistema es necesario para optimizar y automatizar la gestión de reservas, promociones y administración de salas en un cine. Este sistema de reservas es crucial para la operación eficiente de un cine moderno, mejorando la administración interna y ofreciendo una mejor experiencia al cliente. Actualmente, la gestión manual o con sistemas poco eficientes puede generar errores, como la sobreventa de boletos, falta de control en las promociones y dificultades en la administración del inventario de boletos y funciones.

Clases:

1. Clase Persona (Clase base para usuarios y empleados)

- Atributos:
    - nombre (str): Nombre de la persona.
    - apellido (str): Apellido de la persona.
    - id (str o int): Identificador único.
    - email (str): Correo electrónico.
    - (Opcional) telefono (str): Número de contacto.
- Métodos:
    - __init__(self, nombre, apellido, id, email, ...): Constructor para inicializar la información personal.
    - mostrar_info(self): Devuelve o imprime la información básica de la persona.

2. Clase Usuario (Hereda de Persona)

- Atributos:

    - (Hereda todos los de Persona)
    - (Opcional) historial_reservas (list): Registro de reservas realizadas.
- Métodos:
    - hacer_reserva(self, funcion, asientos, promocion=None): Permite crear una reserva para una función determinada.
    - ver_promociones(self): Muestra las promociones vigentes disponibles para el usuario.
    - cancelar_reserva(self, reserva): Permite anular una reserva previamente realizada.

3. Clase Empleado (Hereda de Persona)

- Atributos:
    - (Hereda todos los de Persona)
    - rol (str): Rol asignado al empleado (por ejemplo, "taquillero", "administrador", "limpieza").
- Métodos:
    - asignar_rol(self, nuevo_rol): Permite actualizar o asignar un rol al empleado.
    - gestionar_reserva(self, reserva): (Opcional) Método para modificar o confirmar reservas, especialmente si el empleado es administrador o taquillero.
    - mostrar_info(self): Puede extender o especializar la presentación de la información incluyendo el rol.

4. Clase Espacio (Clase base para espacios físicos del cine)

- Atributos:
    - codigo (str o int): Identificador único del espacio.
    - ubicacion (str): Ubicación o descripción del lugar dentro del cine.
- Métodos:
    - __init__(self, codigo, ubicacion): Inicializa el espacio.
    - mostrar_detalles(self): Muestra información básica del espacio.

5. Clase Sala (Hereda de Espacio)

- Atributos:
    - (Hereda codigo y ubicacion de Espacio)
    - tipo (str): Tipo de sala ("2D", "3D", "IMAX").
    - capacidad (int): Número total de asientos.
    - disponibilidad (bool o estructura de datos para manejo de horarios): Indica si la sala está disponible para reservar.
- Métodos:
    - mostrar_detalles(self): Muestra la información completa de la sala, incluyendo tipo, capacidad y estado.
    - actualizar_disponibilidad(self, estado): Actualiza la disponibilidad de la sala.
    - (Opcional) reservar_asiento(self, cantidad): Reduce la cantidad de asientos disponibles según la reserva.

6. Clase Zona Comida (Hereda de Espacio)

- Atributos:
    - (Hereda codigo y ubicacion de Espacio)
    - productos (dict o list): Lista o diccionario de productos disponibles con sus precios.
- Métodos:
    - agregar_producto(self, producto, precio): Agrega un nuevo producto al inventario.
    - eliminar_producto(self, producto): Elimina un producto existente.
    - mostrar_menu(self): Muestra la lista de productos y precios disponibles.

7. Clase Película

- Atributos:
    - titulo (str): Nombre de la película.
    - duracion (int): Duración en minutos.
    - clasificacion (str): Clasificación por edades o contenido.
    - genero (str): Género cinematográfico (acción, comedia, drama, etc.).
- Métodos:
    - __init__(self, titulo, duracion, clasificacion, genero): Inicializa la película.
    - mostrar_info(self): Devuelve o imprime los detalles de la película.

8. Clase Promoción

- Atributos:
    - nombre (str): Nombre identificador de la promoción.
    - descuento (float): Porcentaje de descuento (por ejemplo, 15.0 para 15%).
    - descripcion (str): Descripción breve de la promoción.
    - (Opcional) fecha_inicio y fecha_fin (date): Período de validez.
    - (Opcional) aplicable_a (list): Indica si la promoción es para ciertos usuarios, funciones u otros criterios.
- Métodos:
    - aplicar_promocion(self, precio_total): Devuelve el precio final tras aplicar el descuento.
    - mostrar_detalles(self): Imprime la información y condiciones de la promoción.

9. Clase Reserva

- Atributos:
    - codigo_reserva (str o int): Identificador único de la reserva.
    - usuario (Usuario): Usuario que realiza la reserva.
    - funcion (Funcion): Función asociada a la reserva.
    - promocion (Promoción, opcional): Promoción aplicada, en caso de existir.
    - numero_asientos (int): Número de asientos reservados.

- (Opcional) total (float): Total a pagar luego de aplicar promociones.

- Métodos:
  - confirmar_reserva(self): Confirma y guarda la reserva.
  - cancelar_reserva(self): Cancela la reserva y libera asientos.
  - calcular_total(self): Calcula el costo total, aplicando la promoción si corresponde.
  - mostrar_detalles(self): Imprime la información completa de la reserva.

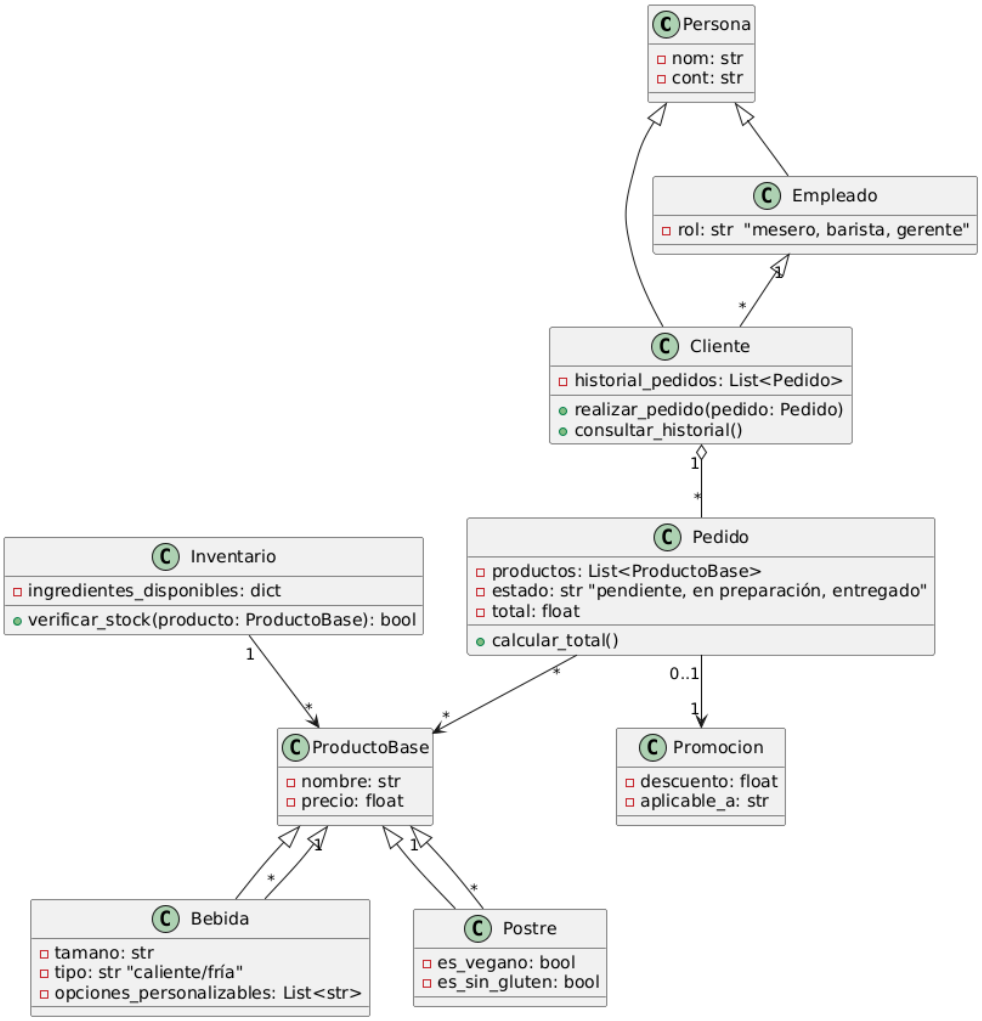
10. Clase Funcion

- Atributos:
  - codigo_funcion (str o int): Identificador único de la función.
  - pelicula (Película): Película que se proyecta.
  - sala (Sala): Sala donde se proyecta la película.
  - hora (datetime o str): Fecha y hora de la función.
  - asientos_disponibles (int): Número de asientos disponibles para la función.
- Métodos:
  - mostrar_info(self): Muestra detalles de la función (película, sala, hora y asientos disponibles).
  - reservar_asiento(self, cantidad): Actualiza la cantidad de asientos disponibles según se realice una reserva.

- actualizar_asientos(self, nuevos_asientos): Permite modificar el número de asientos disponibles (por ejemplo, tras un mantenimiento o cambio).

Segundo programa: Gestión de Pedidos en una Cafetería

Diagrama UML



Código de Python

```python
from typing import List, Optional
class Persona:
    def __init__ (self,nombre,contacto,):
        self.nom=nombre
        self.con=contacto
```

✓ 0.0s

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

    def realizar_pedido(self, pedido):
        self.historial_pedidos.append(pedido)
        print(f"Pedido realizado por {self.nom}: {pedido}")

    def consultar_historial(self):
        return self.historial_pedidos
```

✓ 0.0s Python

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

    def actualizar_inventario(self, inventario, ingrediente, cantidad):
        inventario.actualizar_stock(ingrediente, cantidad)
        print(f"Inventario actualizado por {self.nom}: {ingrediente} -> {cantidad}")
```

✓ 0.0s Python

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

✓ 0.0s

```python
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 []

    def agregar_personalizacion(self, personalizacion):
        self.personalizaciones.append(personalizacion)
```

✓ 0.0s

```python
class Postre(ProductoBase):
    def __init__(self, nombre, precio, vegano=False, sgluten=False):
        self.nom=nombre
        self.price=precio
        self.veg=vegano
        self.glu=sgluten
    def __str__(self):
        esvegano= "es vegano" if self.veg else "No es vegano"
        sin_gluten="sin gluten" if self.glu else "Con gluten"
        return f"el postre es {self.nom} el precio es ${self.price} {esvegano} y {sin_gluten}"
```

```python
postre1 = Postre("Tarta de chocolate", 10, vegano=True, sgluten=True)
postre2 = Postre("Cheesecake", 6, vegano=False, sgluten=False)
postre3 = Postre("Muffin de chocolate", 15, vegano=False, sgluten=True)
print(postre3)
```

✓ 0.0s

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

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

    def actualizar_stock(self, ingrediente, cantidad):
        if ingrediente in self.stock:
            self.stock[ingrediente] += cantidad
        else:
            self.stock[ingrediente] = cantidad

    def verificar_stock(self, ingrediente, cantidad):
        return self.stock.get(ingrediente, 0) >= cantidad
```

```python
class Pedido:
    def __init__(self,client):
        self.cliente=client
        self.prod=[]
        self.estado="pendiente"
        self.total=0
    def agregar_productos(self, producto):
        self.prod.append(producto)
        self.total+=producto.precio
        print(f"Producto agregado al pedido: {producto.nombre}")
    def calcular_total(self):
        return self.total
    def cambiar_estado(self, nuevo_estado):
        self.estado=nuevo_estado
        print(f"Estado del pedido cambiado a: {self.estado}")
    def __str__(self):
        return f"Pedido con {len(self.prod)} productos, Estado: {self.estado}"
```

```python
class Promocion:
    def __init__(self,descuento,condicion):
        self.desc=descuento
        self.cond=condicion
    def aplicar_descuento(self,pedido):
        if self.cond(pedido):
            pedido.total -= self.descuento
            print(f"descuento aplicado:{self.desc}")
```

```python
inventario = Inventario()
inventario.agregar_ingrediente("leche de almendra", 10)
inventario.agregar_ingrediente("azúcar", 20)

cliente = Cliente("Kenath", "23945")
empleado = Empleado("Alfredo", "64621", "barista")

bebida = Bebida("Café con leche", 3, "grande", "caliente")
bebida.agregar_personalizacion("leche de almendra")
bebida.agregar_personalizacion("sin azúcar")

pedido = Pedido(cliente)
pedido.agregar_productos(bebida)

if inventario.verificar_stock("leche de almendra", 1) and inventario.verificar_stock("azúcar", 0):
    cliente.realizar_pedido(pedido)
    pedido.cambiar_estado("en preparación")
else:
    print("No hay suficiente stock para realizar el pedido")

promocion = Promocion(1.0, lambda p: len(p.cliente.historial_pedidos) >= 5)
promocion.aplicar_descuento(pedido)

print(f"Total del pedido: {pedido.calcular_total()}")
```

✓ 0.0s

```
Producto agregado al pedido: Café con leche
Pedido realizado por Kenath: Pedido con 1 productos, Estado: pendiente
Estado del pedido cambiado a: en preparación
Total del pedido: 3
```

Necesidad del programa:

El programa es fundamental para mejorar la eficiencia operativa, brindar un mejor servicio al cliente y mantener un control adecuado del inventario y las promociones en la cafetería, así como también es mas practico y sencillo para los clientes y trabajadores

Clases

1. Clase Persona

Clase base para clientes y empleados.

- Atributos:
    - nombre (str)
    - contacto (str)
- Métodos:

- __init__(...): Inicializa los atributos básicos.
- mostrar_info(): Retorna o imprime la información personal.

1.1. Subclase Cliente

Hereda de Persona.

- Atributos adicionales:
  - historial_pedidos (list): Registro de pedidos realizados.
  - (Opcional) puntos_fidelidad (int): Acumulación de puntos por pedidos.
- Métodos:
  - realizar_pedido(pedido): Inicia un nuevo pedido.
  - consultar_historial(): Muestra el historial de pedidos.
  - (Opcional) aplicar_promocion_fidelidad(): Verifica y aplica promociones especiales para clientes frecuentes.

1.2. Subclase Empleado

Hereda de Persona.

- Atributos adicionales:
  - rol (str): Puede ser "mesero", "barista" o "gerente".
- Métodos:
  - gestionar_pedido(pedido): Permite actualizar el estado del pedido (por ejemplo, cambiar a “en preparación” o “entregado”).
  - actualizar_inventario(inventario, ingrediente, cantidad): Modifica el stock de un ingrediente en el inventario.

- (Opcional) mostrar_info(): Puede incluir información extra del rol.

2. Clase ProductoBase

Clase base para productos en la cafetería.

- Atributos:
  - nombre (str)
  - precio (float)
  - (Opcional) ingredientes (list o dict): Ingredientes básicos del producto.
- Métodos:
  - __init__(...): Inicializa el producto.
  - mostrar_info(): Muestra detalles del producto.

2.1. Subclase Bebida

Hereda de ProductoBase.

- Atributos adicionales:
  - tamano (str): Por ejemplo, "pequeño", "mediano", "grande".
  - tipo (str): "caliente" o "fría".
  - opciones_personalizables (dict): Opciones como "extra leche", "sin azúcar", etc. (Ejemplo: {"leche": "almendra", "azúcar": "sin azúcar"})
- Métodos:

- personalizar(opcion, valor): Permite agregar o modificar una opción de personalización en la bebida.
  Ejemplo: para configurar un “café con leche de almendra y sin azúcar”.
- mostrar_info(): Extiende la información base mostrando tamaño, tipo y opciones aplicadas.

2.2. Subclase Postre

Hereda de ProductoBase.

- Atributos adicionales:
  - es_vegano (bool)
  - sin_gluten (bool)
- Métodos:
  - mostrar_info(): Incluye en la información si el postre es apto para veganos o personas con intolerancia al gluten.

3. Clase Inventario

Encargada de gestionar la cantidad de ingredientes disponibles.

- Atributos:
  - ingredientes (dict): Clave = nombre del ingrediente, Valor = cantidad disponible.
    Ejemplo: {"café": 100, "leche": 50, "almendra": 20}
- Métodos:
  - verificar_stock(ingrediente, cantidad): Retorna True si hay stock suficiente, o False en caso contrario.

- actualizar_stock(ingrediente, cantidad): Suma o resta cantidad al stock (por ejemplo, tras un pedido o una reposición).
- agregar_ingrediente(ingrediente, cantidad): Agrega un nuevo ingrediente o aumenta el stock existente.
- mostrar_inventario(): Lista todos los ingredientes y sus cantidades.

4. Clase Pedido

Representa un pedido realizado en la cafetería.

- Atributos:
  - lista_productos (list): Lista de objetos de tipo ProductoBase (Bebida, Postre, etc.).
  - estado (str): Puede ser "pendiente", "en preparación" o "entregado".
  - total (float): Costo total del pedido, calculado en base a los productos y promociones aplicadas.
  - (Opcional) promocion_aplicada (Promoción): Promoción que se haya aplicado al pedido.
- Métodos:
  - agregar_producto(producto): Añade un producto a la lista del pedido.
  - quitar_producto(producto): Remueve un producto del pedido.
  - calcular_total(): Suma los precios de los productos, aplicando descuentos si existe una promoción.
  - actualizar_estado(nuevo_estado): Cambia el estado del pedido.

- mostrar_detalles(): Imprime o retorna los detalles del pedido, incluidos productos, estado y total.
- (Opcional) verificar_ingredientes(inventario): Recorre los productos del pedido y consulta en el inventario si hay stock suficiente para cada ingrediente; en caso negativo, rechaza el pedido o notifica al empleado.

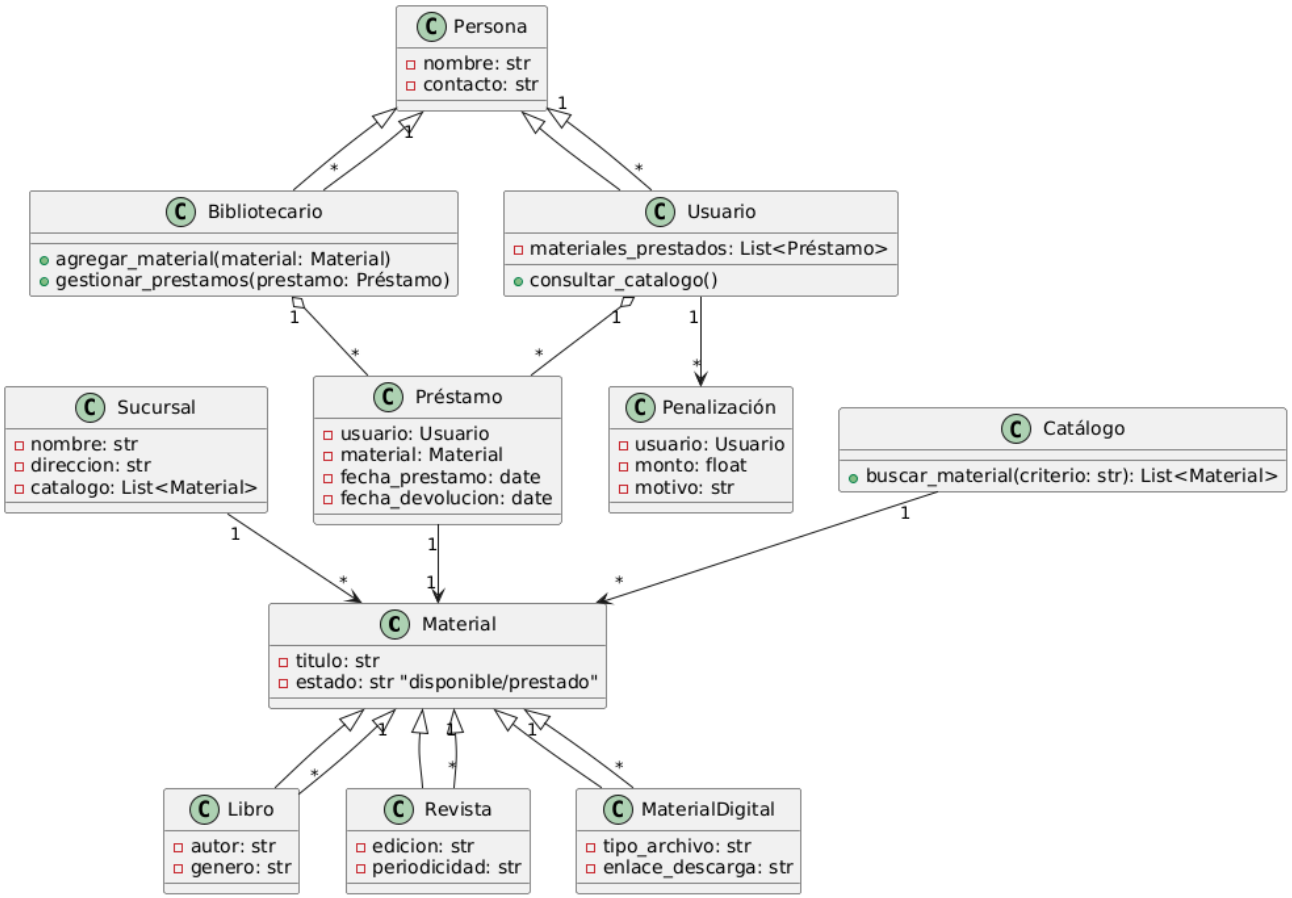
5. Clase Promoción

Para aplicar descuentos a pedidos o productos específicos.

- Atributos:
  - nombre (str): Identificador de la promoción.
  - descuento (float): Porcentaje de descuento (ej. 10.0 para un 10%).
  - descripcion (str)
  - (Opcional) fecha_inicio y fecha_fin (date): Periodo de validez.
  - (Opcional) condiciones (dict o función): Reglas para aplicar la promoción (por ejemplo, número mínimo de pedidos para clientes frecuentes).
- Métodos:
  - aplicar_promocion(total): Devuelve el total luego de aplicar el descuento.
  - mostrar_detalles(): Muestra la información y condiciones de la promoción.

Tercer programa: Biblioteca Digital

Diagrama UML



Código de Python

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

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

    def __repr__(self):
        return f"Material: {self.titulo}, Estado: {self.estado}"
```

0.0s

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

    def __str__(self):
        return f"Revista: {self.titulo}, Edición: {self.edicion}, Periodicidad: {self.periodicidad}, Estado: {self.estado}"
```

0.0s

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

    def __str__(self):
        return f"Libro: {self.titulo}, Autor: {self.autor}, Género: {self.genero}, Estado: {self.estado}"
```

✓ 0.0s Python

```python
:lass 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
    def __str__(self):
        return f"Material Digital: {self.titulo}, Tipo: {self.tipo_archivo}, Enlace: {self.enlace_descarga}, Estado: {self.est
```

✓ 0.0s Python

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

    def __str__(self):
        return self.nombre
```

✓ 0.0s

```python
class Usuario(Persona):
    def __init__(self, nombre):
        super().__init__(nombre)
        self.prestamos = []
        self.penalizaciones = 0

    def consultar_catalogo(self, catalogo):
        return catalogo.mostrar_materiales()

    def __str__(self):
        return f"Usuario: {self.nombre}, Préstamos activos: {len(self.prestamos)}, Multas: {self.penalizaciones}"
```

✓ 0.0s

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

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

    def remover_material(self, material):
        if material in self.catalogo:
            self.catalogo.remove(material)
            return True
        return False

    def buscar_material(self, titulo):
        return next((m for m in self.catalogo if m.titulo == titulo), None)

    def __str__(self):
        return f"Sucursal: {self.nombre}, Materiales disponibles: {len(self.catalogo)}"
```

✓ 0.0s

```python
from datetime import datetime, timedelta
class Prestamo:
    def __init__(self, usuario, material, dias_prestamo=7):
        self.usuario = usuario
        self.material = material
        self.fecha_prestamo = datetime.now()
        self.fecha_devolucion = self.fecha_prestamo + timedelta(days=dias_prestamo)
        material.estado = "prestado"

    def devolver(self):
        if datetime.now() > self.fecha_devolucion:
            dias_atraso = (datetime.now() - self.fecha_devolucion).days
            monto_multa = dias_atraso * 10  # Multa de 10 por día de atraso
            penalizacion.aplicar_multa(self.usuario, monto_multa)
            self.usuario.penalizaciones += monto_multa
        self.material.estado = "disponible"
        return "Material devuelto"

    def __str__(self):
        return f"Préstamo: {self.material.titulo}, Usuario: {self.usuario.nombre}, Fecha de devolución: {self.fecha_devolucion
```

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

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

    def gestionar_prestamo(self, usuario, material, sucursal):
        if material.estado == "disponible":
            prestamo = Prestamo(usuario, material)
            usuario.prestamos.append(prestamo)
            sucursal.remover_material(material)
            return f"Préstamo realizado: {material.titulo}"
        return "Material no disponible"

    def transferir_material(self, material, sucursal_origen, sucursal_destino):
        if material in sucursal_origen.catalogo:
            sucursal_origen.remover_material(material)
            sucursal_destino.agregar_material(material)
            return f"Material {material.titulo} transferido a {sucursal_destino.nombre}"
        return "Material no encontrado en la sucursal origen"
```

0.0s

```python
class Penalizacion:
    def __init__(self):
        self.registro = {}  # Almacena multas por usuario

    def aplicar_multa(self, usuario, monto):
        if usuario not in self.registro:
            self.registro[usuario] = 0
        self.registro[usuario] += monto

    def consultar_multa(self, usuario):
        return self.registro.get(usuario, 0)

    def __str__(self):
        return f"Registro de multas: {self.registro}"
```

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

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

    def buscar_por_titulo(self, titulo):
        return [m for m in self.materiales if titulo.lower() in m.titulo.lower()]
```

✓ 0.0s

```python
penalizacion = Penalizacion()
sucursal_central = Sucursal("Central")
sucursal_norte = Sucursal("Norte")

libro1 = Libro("Padre rico Padre pobre", "Robert Kiyosaki", "Finanzas")
revista1 = Revista("Muy Interesante Junior", "Edición 2025", "Mensual")

usuario1 = Usuario("Carlos Vallarin")
bibliotecario1 = Bibliotecario("Roberto gonzales")
bibliotecario1.agregar_material(sucursal_central, libro1)
bibliotecario1.agregar_material(sucursal_central, revista1)
print(bibliotecario1.transferir_material(libro1, sucursal_central, sucursal_norte))
print(bibliotecario1.gestionar_prestamo(usuario1, revista1, sucursal_central))


print(f"Sucursal Centro: {sucursal_central}")
print(f"Sucursal Norte: {sucursal_norte}")
print(f"Usuario: {usuario1}")
print(f"Préstamos activos de {usuario1.nombre}:")
for prestamo in usuario1.prestamos:
    print(f"  - {prestamo}")
print(f"Multas acumuladas por {usuario1.nombre}: {penalizacion.consultar_multa(usuario1)}")
```

```
Material Padre rico Padre pobre transferido a Norte
Préstamo realizado: Muy Interesante Junior
Sucursal Centro: Sucursal: Central, Materiales disponibles: 0
Sucursal Norte: Sucursal: Norte, Materiales disponibles: 1
Usuario: Usuario: Carlos Vallarin, Préstamos activos: 1, Multas: 0
Préstamos activos de Carlos Vallarin:
  - Préstamo: Muy Interesante Junior, Usuario: Carlos Vallarin, Fecha de devolución: 2025-02-19 21:13:09.372538
Multas acumuladas por Carlos Vallarin: 0
```

Necesidad del programa:

El programa es esencial para mejorar la eficiencia operativa de la biblioteca, ofreciendo una experiencia de usuario más fluida y un manejo más efectivo de los recursos y préstamos, lo que a su vez contribuye a un mejor servicio a la comunidad, así como permite la conexión entre diferentes sucursales y permite optimizar las búsquedas.

Clases

1. Clase Material

Clase base para cualquier material en la biblioteca.

- Atributos:
    - id (str/int): Identificador único del material.
    - titulo (str): Título del material.
    - anio (int): Año de publicación (opcional según el material).
    - estado (str): Estado del material (por ejemplo, "disponible", "prestado").
      (Nota: para materiales digitales este atributo podría no ser relevante de la misma forma.)
- Métodos:
    - __init__(self, id, titulo, anio, estado="disponible"): Constructor para inicializar el material.
    - mostrar_info(self): Devuelve o imprime la información básica del material.
    - cambiar_estado(self, nuevo_estado): Actualiza el atributo estado del material.

1.1. Subclase Libro (hereda de Material)

- Atributos adicionales:
    - autor (str)
    - genero (str)

- Métodos:
    - __init__(self, id, titulo, anio, autor, genero, estado="disponible"): Inicializa el libro, llamando al constructor de la clase base.
    - mostrar_info(self): Muestra información específica del libro, incluyendo autor y género.

1.2. Subclase Revista (hereda de Material)

- Atributos adicionales:
    - edicion (str o int): Número o identificación de la edición.
    - periodicidad (str): Frecuencia de publicación (por ejemplo, "semanal", "mensual").
- Métodos:
    - __init__(self, id, titulo, anio, edicion, periodicidad, estado="disponible"): Inicializa la revista.
    - mostrar_info(self): Muestra detalles específicos de la revista, como edición y periodicidad.

1.3. Subclase MaterialDigital (hereda de Material)

- Atributos adicionales:
    - tipo_archivo (str): Formato del archivo (por ejemplo, PDF, EPUB).
    - enlace_descarga (str): URL o ruta para descargar el material digital.
- Métodos:
    - __init__(self, id, titulo, anio, tipo_archivo, enlace_descarga): Inicializa el material digital.

    - mostrar_info(self): Muestra la información del material digital, incluyendo el enlace y el tipo de archivo.

2. Clase Persona

Clase base para usuarios y empleados.

- Atributos:
    - id (str/int): Identificador único.
    - nombre (str)
    - apellido (str)
    - email (str)
- Métodos:
    - __init__(self, id, nombre, apellido, email): Constructor básico.
    - mostrar_info(self): Muestra la información personal.

2.1. Subclase Usuario (hereda de Persona)

- Atributos adicionales:
    - materiales_prestados (list): Lista actual de préstamos activos.
    - historial_prestamos (list): Registro histórico de todos los préstamos realizados.
    - (Opcional) multa_acumulada (float): Total de penalizaciones acumuladas.
- Métodos:
    - consultar_catalogo(self, catalogo): Permite buscar y ver materiales disponibles en el catálogo.

- solicitar_prestamo(self, material, sucursal): Inicia un préstamo para un material en una sucursal determinada.
- devolver_material(self, prestamo): Registra la devolución de un material y actualiza el préstamo.
- (Opcional) registrar_penalizacion(self, penalizacion): Agrega una penalización al historial del usuario.

2.2. Subclase Bibliotecario (hereda de Persona)

- Atributos adicionales:
  - sucursal_asignada (Sucursal): La sucursal en la que trabaja el bibliotecario (opcional).
- Métodos:
  - agregar_material(self, material, sucursal): Agrega un nuevo material al catálogo de la sucursal.
  - gestionar_prestamo(self, prestamo, accion): Actualiza el estado de un préstamo (por ejemplo, aprobar o registrar devolución).
  - transferir_material(self, material, sucursal_origen, sucursal_destino): Permite mover un material de una sucursal a otra.

3. Clase Sucursal

Representa una sucursal de la biblioteca, con su propio catálogo de materiales.

- Atributos:
  - id (str/int): Identificador único de la sucursal.
  - nombre (str)

    - direccion (str)
    - catalogo (list o instancia de Catálogo): Colección de materiales disponibles en la sucursal.
- Métodos:
    - agregar_material(self, material): Añade un material al catálogo de la sucursal.
    - remover_material(self, material): Elimina un material del catálogo.
    - listar_materiales_disponibles(self): Retorna una lista de materiales cuyo estado sea "disponible".
    - buscar_material(self, criterio): Permite buscar materiales en la sucursal según un criterio dado.

4. Clase Prestamo

Relaciona un usuario con un material, registrando las fechas de préstamo y devolución.

- Atributos:
    - id (str/int): Identificador del préstamo.
    - usuario (Usuario): Usuario que solicita el préstamo.
    - material (Material): Material que se presta.
    - fecha_prestamo (datetime): Fecha en que se realizó el préstamo.
    - fecha_vencimiento (datetime): Fecha límite para la devolución.
    - fecha_devolucion (datetime, opcional): Fecha en que se devolvió el material (inicialmente None).

    - estado (str): Por ejemplo, "activo" o "finalizado".

- Métodos:
    - iniciar_prestamo(self): Registra la fecha de préstamo, marca el material como prestado y actualiza el estado.
    - finalizar_prestamo(self, fecha_devolucion): Registra la fecha de devolución, cambia el estado a "finalizado" y actualiza el estado del material.
    - verificar_retraso(self): Compara la fecha actual con fecha_vencimiento y retorna True si existe atraso.

5. Clase Penalizacion

Gestiona las multas por retrasos en la devolución de materiales.

- Atributos:
    - id (str/int): Identificador de la penalización.
    - usuario (Usuario): Usuario al que se le aplica la multa.
    - monto (float): Valor de la multa.
    - fecha_registro (datetime): Fecha en que se registró la penalización.
    - estado (str): Por ejemplo, "pendiente" o "pagada".
    - detalle (str): Descripción del motivo o cantidad de días de retraso.
- Métodos:
    - calcular_multa(self, dias_retraso): Calcula el monto de la multa en función de los días de atraso.

- registrar_penalizacion(self): Asocia la penalización al usuario y la guarda en el registro.
- pagar_penalizacion(self): Cambia el estado de la penalización a "pagada".

6. Clase Catalogo

Permite buscar materiales según diferentes criterios (autor, género, tipo, etc.) y a nivel global entre sucursales.

- Atributos:
  - materiales (list): Lista global de materiales disponibles en todas las sucursales.
    (Este atributo podría actualizarse mediante la integración de los catálogos locales de cada sucursal.)
- Métodos:
  - agregar_material(self, material): Añade un material al catálogo global.
  - buscar_por_autor(self, autor): Retorna una lista de materiales cuyo autor coincida.
  - buscar_por_genero(self, genero): Filtra y retorna materiales por género.
  - buscar_por_tipo(self, tipo_material): Filtra por tipo (por ejemplo, "Libro", "Revista", "MaterialDigital").
  - buscar_disponibles(self): Devuelve los materiales con estado "disponible".

- buscar_global(self, sucursales, criterio): Recorre el catálogo de todas las sucursales para encontrar materiales que cumplan con el criterio (por ejemplo, búsqueda por título o autor).

GITHUB https://github.com/CarlosVallarin/Programacion-Avanzada.git