



BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD CIENCIAS DE LA COMPUTACIÓN

PROGRAMACIÓN AVANZADA

M.C. JAIME ALEJANDRO ROMERO SIERRA

PRÁCTICA 1

ABRAHAM FUENTES LÓPEZ

12/02/2025

Sistema de Reservas para un Cine

Justificación de la necesidad del programa

El sector del entretenimiento ha evolucionado con la digitalización, y los cines requieren soluciones eficientes para la gestión de reservas, promociones y disponibilidad de salas. Un Sistema de Reservas para un Cine permite a los usuarios reservar boletos de manera anticipada, reduciendo filas y mejorando la experiencia del cliente. Además, facilita a los empleados la administración de funciones, asientos disponibles y promociones.

Este sistema optimiza la operación del cine al proporcionar un registro de reservas y cancelaciones, permitiendo un control efectivo de los asientos y asegurando una distribución justa. Además, mejora la comunicación entre empleados y clientes, ofreciendo información clara sobre horarios, descuentos y disponibilidad de salas.

Clases a ocupar:

1. Persona

- **Atributos:** nombre, correo
- **Métodos:** registrar(), actualizar_datos(), personas_registradas()
- **Descripción:** Clase base que almacena información de usuarios y empleados.

2. Usuario (hereda de Persona)

- **Atributos:** historial_reservas
- **Métodos:** reservar(), cancelar_reserva()
- **Descripción:** Permite a los usuarios reservar y cancelar boletos.

3. Empleado (hereda de Persona)

- **Atributos:** rol
- **Métodos:** agregar_funcion(), modificar_promocion()
- **Descripción:** Administra funciones y promociones dentro del sistema.

4. Espacio

- **Atributos:** capacidad, identificador
- **Métodos:** descripcion()

- **Descripción:** Representa los espacios dentro del cine.

5. Sala (hereda de Espacio)

- **Atributos:** tipo, disponibilidad
- **Métodos:** consultar_disponibilidad()
- **Descripción:** Define las salas del cine y su disponibilidad.

6. Película

- **Atributos:** título, género, duración
- **Descripción:** Representa la información de cada película disponible.

7. Función

- **Atributos:** pelicula, sala, hora, asientos_disponibles
- **Descripción:** Maneja el horario y disponibilidad de cada proyección.

8. Promoción

- **Atributos:** descuento, condiciones
- **Métodos:** mostrar()
- **Descripción:** Define y aplica descuentos en la compra de boletos.

```

1 class Persona:
2
3     lista=[]
4
5     def __init__(self,nombre,correo):
6         self.nombre=nombre
7         self.correo=correo
8
9     def registrar(self):
10        Persona.lista.append(self)
11        print(f"La persona {self.nombre} ha sido registrada con el correo {self.correo}")
12
13    def actualizar_datos(self,nombre,correo):
14        self.nombre=nombre
15        self.correo=correo
16        print(f"Los datos han sido actualizados")
17
18    @classmethod
19    def personas_registradas(cls):
20        print("Personas registradas")
21        for Persona in cls.lista:
22            print(f"-{Persona.nombre} - {Persona.correo}")
23
24 class Usuario(Persona):
25     def __init__(self, nombre, correo):
26         super().__init__(nombre, correo)
27         self.historial_reservas = []
28
29     def reservar(self, funcion, asientos):
30         if asientos <= funcion.asientos_disponibles:
31             funcion.asientos_disponibles -= asientos
32             self.historial_reservas.append({"funcion": funcion, "asientos": asientos})
33             print(f"Reserva realizada para '{funcion.pelicula.titulo}' en la sala {funcion.sala.identificador}.")
34         else:
35             print("No hay suficientes asientos disponibles.")
36
37     def cancelar_reserva(self, funcion):

```

```

38         reserva = next((r for r in self.historial_reservas if r["funcion"] == funcion), None)
39         if reserva:
40             funcion.asientos_disponibles += reserva["asientos"]
41             self.historial_reservas.remove(reserva)
42             print(f"Reserva cancelada para '{funcion.pelicula.titulo}'.")
43         else:
44             print("No tienes una reserva para esta función.")
45
46     class Empleado(Persona):
47         def __init__(self, nombre, correo, rol):
48             super().__init__(nombre, correo)
49             self.rol = rol
50
51         def agregar_funcion(self, funcion):
52             print(f"Función agregada: {funcion.pelicula.titulo} a las {funcion.hora} en la sala {funcion.sala.identificador}.")
53
54         def modificar_promocion(self, promocion, nuevo_descuento, nuevas_condiciones):
55             promocion.descuento = nuevo_descuento
56             promocion.condiciones = nuevas_condiciones
57             print(f"Promoción modificada: {nuevo_descuento}% de descuento. {nuevas_condiciones}.")
58
59     class Espacio:
60         def __init__(self, capacidad, identificador):
61             self.capacidad = capacidad
62             self.identificador = identificador
63
64         def descripcion(self):
65             print(f"El edificio tiene tamaño {self.capacidad} y tiene id {self.identificador}")
66
67     class Sala(Espacio):
68         def __init__(self, capacidad, identificador, tipo):
69             super().__init__(capacidad, identificador)
70             self.tipo = tipo
71             self.disponibilidad = True
72

```

```

73     def ConsultarDisponibilidad(self):
74         if self.disponibilidad:
75             print("La sala esta disponible")
76         else:
77             print("La sala esta ocupada")
78
79     class Pelicula:
80         def __init__(self, titulo, genero, duracion):
81             self.titulo = titulo
82             self.genero = genero
83             self.duracion = duracion
84
85     class Funcion:
86         def __init__(self, pelicula, sala, hora, asientos_disponibles=None):
87             self.pelicula = pelicula
88             self.sala = sala
89             self.hora = hora
90             self.asientos_disponibles = asientos_disponibles or sala.capacidad
91
92     class Promocion:
93         def __init__(self, descuento, condiciones):
94             self.descuento = descuento
95             self.condiciones = condiciones
96
97         def mostrar(self):
98             print(f"Promoción: {self.descuento}% de descuento. Condiciones: {self.condiciones}")
99
100
101
102 pelicula1 = Pelicula("Matrix", "Ciencia Ficción", 136)
103 pelicula2 = Pelicula("Titanic", "Drama/Romance", 195)
104
105 sala1 = Sala(100, "Sala 1", "3DX")
106 sala2 = Sala(50, "Sala 2", "Tradicional")
107
108 funcion1 = Funcion(pelicula1, sala1, "18:00")


```

```

108 funcion1 = Funcion(pelicula1, sala1, "18:00")
109 funcion2 = Funcion(pelicula2, sala2, "20:00")
110
111 usuario1 = Usuario("Ana Pérez", "ana.perez@email.com")
112 empleado1 = Empleado("Luis Martínez", "luis.martinez@email.com", "Gerente")
113
114 usuario1.registrar()
115 empleado1.registrar()
116
117 usuario1.reservar(funcion1, 3)
118
119 usuario1.cancelar_reserva(funcion1)
120
121 promocion1 = Promocion(20, "Válido de lunes a jueves.")
122 promocion1.mostrar()
123 empleado1.modificar_promocion(promocion1, 30, "Válido todos los días antes de las 5 PM.")
124
125 Persona.personas_registradas()

```

PROBLEMS 1 OUTPUT DEBUG CONSOLE **TERMINAL** PORTS JUPYTER

 Python Debug Console + v

```

PS C:\Users\abrah\OneDrive\Documentos\Nueva carpeta (2)> & 'c:\Users\abrah\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\abrah\.vscode\python.debugpy-2025.0.0-win32-x64\bundle\libs\debugpy\launcher' '59229' '--' 'C:\Users\abrah\OneDrive\Documentos\Nueva carpeta (2)\cine.py'
La persona Ana Pérez ha sido registrada con el correo ana.perez@email.com
La persona Luis Martínez ha sido registrada con el correo luis.martinez@email.com
Reserva realizada para 'Matrix' en la sala Sala 1.
Reserva cancelada para 'Matrix'.
Promoción: 20% de descuento. Condiciones: Válido de lunes a jueves.
Promoción modificada: 30% de descuento. Válido todos los días antes de las 5 PM..
Personas registradas
-Ana Pérez - ana.perez@email.com
-Luis Martínez - luis.martinez@email.com

```

Gestión de Pedidos en una Cafetería

Justificación de la necesidad del programa

En una cafetería con una alta demanda de pedidos y una variedad de productos, es fundamental contar con un sistema de gestión de pedidos que permita registrar clientes, procesar pedidos de manera eficiente y administrar el inventario en tiempo real.

Este sistema busca optimizar la operación diaria, evitando errores en los pedidos, asegurando la disponibilidad de ingredientes y mejorando la experiencia del cliente. Además, permite aplicar promociones y descuentos de manera automatizada, lo que incentiva la fidelización de los clientes.

Un sistema digitalizado de gestión de pedidos facilita el control y seguimiento de las transacciones, reduciendo el margen de error humano y mejorando la coordinación entre empleados y clientes.

Clases a ocupar:

1. Persona

- Atributos:
 - nombre: Nombre de la persona.
 - contacto: Información de contacto.
- Métodos:
 - registrar(): Registra la persona en la lista de usuarios.
 - personas_registradas(): Muestra la lista de personas registradas.

2. Cliente (Hereda de Persona)

- Atributos:
 - historial_pedidos: Lista de pedidos realizados.
- Métodos:
 - realizar_pedido(pedido): Registra y confirma un pedido si hay stock disponible.

- consultar_historial(): Muestra los pedidos realizados por el cliente.

3. Empleado (Hereda de Persona)

- Atributos:
 - rol: Función desempeñada en la cafetería.
- Métodos:
 - actualizar_inventario(inventario, ingrediente, cantidad): Modifica el stock de ingredientes en el inventario.

4. ProductoBase

- Atributos:
 - nombre: Nombre del producto.
 - precio: Precio del producto.

5. Bebida (Hereda de ProductoBase)

- Atributos:
 - tamano: Tamaño de la bebida.
 - tipo: Tipo de bebida (caliente/fría).
 - opciones: Lista de opciones adicionales.

6. Postre (Hereda de ProductoBase)

- Atributos:
 - vegano: Indica si es un producto vegano.
 - sin_gluten: Indica si es libre de gluten.

7. Inventario

- Atributos:
 - ingredientes: Diccionario con los ingredientes y su cantidad disponible.
- Métodos:
 - actualizar_stock(ingrediente, cantidad): Modifica la cantidad disponible de un ingrediente.
 - verificar_stock(ingredientes_necesarios): Verifica si hay suficiente stock para un pedido.
 - consumir_ingredientes(ingredientes_necesarios): Resta los ingredientes utilizados de la lista de inventario.

8. Pedido

- Atributos:
 - cliente: Cliente que realiza el pedido.
 - productos: Lista de productos en el pedido.
 - estado: Estado del pedido (Pendiente, En preparación, Listo).
 - total: Costo total del pedido.
 - inventario: Referencia al inventario para gestionar ingredientes.
- Métodos:
 - validar_pedido(): Verifica si hay suficiente stock para completar el pedido.
 - confirmar_pedido(): Confirma el pedido y actualiza el inventario.

9. Promocion

- Atributos:
 - descripcion: Descripción de la promoción.
 - descuento: Porcentaje de descuento.
- Métodos:
 - aplicar_descuento(pedido): Aplica un descuento al total del pedido.


```

1 class Persona:
2     lista_personas = []
3
4     def __init__(self, nombre, contacto):
5         self.nombre = nombre
6         self.contacto = contacto
7
8     def registrar(self):
9         Persona.lista_personas.append(self)
10        print(f"{self.nombre} ha sido registrado correctamente.")
11
12    @classmethod
13    def personas_registradas(cls):
14        print("Personas registradas:")
15        for persona in cls.lista_personas:
16            print(f"- {persona.nombre} ({persona.contacto})")
17
18 class Cliente(Persona):
19     def __init__(self, nombre, contacto):
20         super().__init__(nombre, contacto)
21         self.historial_pedidos = []
22
23     def realizar_pedido(self, pedido):
24         if pedido.validar_pedido():
25             self.historial_pedidos.append(pedido)
26             pedido.confirmar_pedido()
27         else:
28             print("Pedido rechazado por falta de stock.")
29
30     def consultar_historial(self):
31         print(f"Historial de pedidos de {self.nombre}:")
32         for pedido in self.historial_pedidos:
33             print(f"- Pedido: {pedido.productos}, Estado: {pedido.estado}, Total: {pedido.total}")
34
35 class Empleado(Persona):
36     def __init__(self, nombre, contacto, rol):
37         super().__init__(nombre, contacto)
38         self.rol = rol
39
40     def actualizar_inventario(self, inventario, ingrediente, cantidad):
41         inventario.actualizar_stock(ingrediente, cantidad)
42
43 class ProductoBase:
44     def __init__(self, nombre, precio):
45         self.nombre = nombre
46         self.precio = precio
47
48 class Bebida(ProductoBase):
49     def __init__(self, nombre, precio, tamano, tipo, opciones=None):
50         super().__init__(nombre, precio)
51         self.tamano = tamano
52         self.tipo = tipo
53         self.opciones = opciones if opciones else []
54
55 class Postre(ProductoBase):
56     def __init__(self, nombre, precio, vegano=False, sin_gluten=False):
57         super().__init__(nombre, precio)
58         self.vegano = vegano
59         self.sin_gluten = sin_gluten
60
61 class Inventario:
62     def __init__(self):
63         self.ingredientes = {}
64
65     def actualizar_stock(self, ingrediente, cantidad):
66         self.ingredientes[ingrediente] = self.ingredientes.get(ingrediente, 0) + cantidad
67         print(f"Inventario actualizado: {ingrediente} -> {self.ingredientes[ingrediente]}")
68
69     def verificar_stock(self, ingredientes_necesarios):
70         return all(self.ingredientes.get(ing, 0) >= cant for ing, cant in ingredientes_necesarios.items())
71

```

```

72     def consumir_ingredientes(self, ingredientes_necesarios):
73         if self.verificar_stock(ingredientes_necesarios):
74             for ing, cant in ingredientes_necesarios.items():
75                 self.ingredientes[ing] -= cant
76             return True
77         return False
78
79 class Pedido:
80     def __init__(self, cliente, productos, inventario):
81         self.cliente = cliente
82         self.productos = productos
83         self.estado = "Pendiente"
84         self.total = sum(prod.precio for prod in productos)
85         self.inventario = inventario
86
87     def validar_pedido(self):
88         ingredientes_necesarios = {}
89         return self.inventario.verificar_stock(ingredientes_necesarios)
90
91     def confirmar_pedido(self):
92         if self.inventario.consumir_ingredientes({}):
93             self.estado = "En preparación"
94             print(f"Pedido confirmado para {self.cliente.nombre}. Total: ${self.total}")
95         else:
96             print("No se pudo procesar el pedido por falta de ingredientes.")
97
98 class Promocion:
99     def __init__(self, descripcion, descuento):
100         self.descripcion = descripcion
101         self.descuento = descuento
102
103     def aplicar_descuento(self, pedido):
104         pedido.total *= (1 - self.descuento / 100)
105         print(f"Promoción aplicada: {self.descripcion}. Nuevo total: ${pedido.total}")
106

```

```

108 inventario = Inventario()
109 inventario.actualizar_stock("Café", 10)
110 inventario.actualizar_stock("Leche", 5)
111
112 cliente1 = Cliente("Ana Pérez", "ana@email.com")
113 cliente1.registrar()
114
115 empleado1 = Empleado("Luis Martínez", "luis@email.com", "Barista")
116 empleado1.registrar()
117
118 bebida1 = Bebida("Café Latte", 4.5, "Grande", "Caliente", ["Leche de almendra"])
119 postre1 = Postre("Brownie", 3.0, sin_gluten=True)
120
121 pedido1 = Pedido(cliente1, [bebida1, postre1], inventario)
122 cliente1.realizar_pedido(pedido1)
123
124 promocion1 = Promocion("Descuento del 10% para clientes frecuentes", 10)
125 promocion1.aplicar_descuento(pedido1)
126
127 cliente1.consultar_historial()
128 Persona.personas_registradas()
129

```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER

Python Debug Console + -

```

Inventario actualizado: Café -> 10
Inventario actualizado: Leche -> 5
Ana Pérez ha sido registrado correctamente.
Luis Martínez ha sido registrado correctamente.
Pedido confirmado para Ana Pérez. Total: $7.5
Promoción aplicada: Descuento del 10% para clientes frecuentes. Nuevo total: $6.75
Historial de pedidos de Ana Pérez:
- Pedido: [<_main_.Bebida object at 0x000001B9554CCC20>, <_main_.Postre object at 0x000001B9554CCD70>], Estado: En preparación, Total: 6.75
Personas registradas:
- Ana Pérez (ana@email.com)
- Luis Martínez (luis@email.com)

```

Biblioteca Digital

Justificación de la necesidad del programa

Las bibliotecas tradicionales han evolucionado hacia plataformas digitales que permiten un acceso más eficiente a los materiales de lectura y consulta. Un sistema de biblioteca digital facilita la gestión de préstamos, devoluciones, sanciones por retrasos y administración de sucursales. Además, permite la consulta remota de libros digitales y revistas, optimizando la experiencia de los usuarios y reduciendo costos de mantenimiento físico.

El desarrollo de este programa responde a la necesidad de automatizar la gestión bibliotecaria, garantizando disponibilidad de materiales, trazabilidad de préstamos y aplicación de sanciones. También permite la transferencia de recursos entre sucursales, lo que mejora la distribución del material según la demanda de los usuarios.

Clases a ocupar:

1. **Material** (*Clase base*)
 - **Atributos:** titulo, estado (disponible/prestado)
 - **Métodos:** __init__(), cambiar_estado()
2. **Libro** (*Hereda de Material*)
 - **Atributos:** autor, género
 - **Métodos:** __init__()
3. **Revista** (*Hereda de Material*)
 - **Atributos:** edición, periodicidad
 - **Métodos:** __init__()
4. **MaterialDigital** (*Hereda de Material*)
 - **Atributos:** tipo_archivo, enlace_descarga
 - **Métodos:** __init__()
5. **Persona** (*Clase base para usuarios y bibliotecarios*)
 - **Atributos:** nombre, email
 - **Métodos:** __init__()

6. **Usuario** (*Hereda de Persona*)

- **Atributos:** materiales_prestados, penalizaciones
- **Métodos:** consultar_catalogo(), solicitar_prestamo(), devolver_material(), verificar_penalizaciones()

7. **Bibliotecario** (*Hereda de Persona*)

- **Atributos:** sucursal
- **Métodos:** agregar_material(), gestionar_prestamo(), transferir_material()

8. **Sucursal**

- **Atributos:** nombre, catalogo
- **Métodos:** __init__(), transferir_material()

9. **Préstamo**

- **Atributos:** usuario, material, fecha_prestamo, fecha_devolucion
- **Métodos:** __init__()

10. **Penalización**

- **Atributos:** usuario, monto
- **Métodos:** __init__(), mostrar_penalizacion()

11. **Catálogo**

- **Atributos:** materiales
- **Métodos:** agregar_material(), buscar_materiales_disponibles(), buscar_por_autor(), buscar_por_genero(), buscar_por_tipo()

12. **Biblioteca**

- **Atributos:** sucursales
- **Métodos:** agregar_sucursal(), buscar_material_en_todas_sucursales()

```

1  from datetime import datetime, timedelta
2
3  class Material:
4      def __init__(self, titulo, estado='disponible'):
5          self.titulo = titulo
6          self.estado = estado
7
8      def prestar(self):
9          if self.estado == 'disponible':
10             self.estado = 'prestado'
11             return True
12             return False
13
14      def devolver(self):
15          self.estado = 'disponible'
16
17  class Libro(Material):
18      def __init__(self, titulo, autor, genero):
19          super().__init__(titulo)
20          self.autor = autor
21          self.genero = genero
22
23  class Revista(Material):
24      def __init__(self, titulo, edicion, periodicidad):
25          super().__init__(titulo)
26          self.edicion = edicion
27          self.periodicidad = periodicidad
28
29  class MaterialDigital(Material):
30      def __init__(self, titulo, tipo_archivo, enlace):
31          super().__init__(titulo, estado='disponible')
32          self.tipo_archivo = tipo_archivo
33          self.enlace = enlace
34
35  class Persona:
36      def __init__(self, nombre, correo):
37          self.nombre = nombre
38          self.correo = correo
39
40  class Usuario(Persona):
41      def __init__(self, nombre, correo):
42          super().__init__(nombre, correo)
43          self.materiales_prestados = []
44          self.penalizaciones = 0
45
46      def consultar_catalogo(self, catalogo):
47          catalogo.mostrar_materiales()
48
49      def solicitar_prestamo(self, material, bibliotecario):
50          bibliotecario.procesar_prestamo(self, material)
51
52      def devolver_material(self, material, bibliotecario):
53          bibliotecario.procesar_devolucion(self, material)
54
55  class Bibliotecario(Persona):
56      def __init__(self, nombre, correo, sucursal):
57          super().__init__(nombre, correo)
58          self.sucursal = sucursal
59
60      def agregar_material(self, material):
61          self.sucursal.agregar_material(material)
62
63      def procesar_prestamo(self, usuario, material):
64          if material.prestar():
65              usuario.materiales_prestados.append(material)
66              print(f"{usuario.nombre} ha tomado prestado '{material.titulo}'.")
67          else:
68              print("Material no disponible.")
69
70      def procesar_devolucion(self, usuario, material):
71          if material in usuario.materiales_prestados:

```

```

72         usuario.materiales_prestados.remove(material)
73         material.devolver()
74         print(f"{usuario.nombre} ha devuelto '{material.titulo}'.")
75     else:
76         print("Este usuario no tiene este material en préstamo.")
77
78     class Sucursal:
79         def __init__(self, nombre):
80             self.nombre = nombre
81             self.catalogo = []
82
83         def agregar_material(self, material):
84             self.catalogo.append(material)
85
86         def transferir_material(self, material, otra_sucursal):
87             if material in self.catalogo:
88                 self.catalogo.remove(material)
89                 otra_sucursal.agregar_material(material)
90                 print(f"'{material.titulo}' ha sido transferido a {otra_sucursal.nombre}.")
91
92     class Prestamo:
93         def __init__(self, usuario, material, fecha_prestamo, dias_prestamo=7):
94             self.usuario = usuario
95             self.material = material
96             self.fecha_prestamo = fecha_prestamo
97             self.fecha_devolucion = fecha_prestamo + timedelta(days=dias_prestamo)
98
99     class Penalizacion:
100         def __init__(self, usuario, dias_retraso):
101             self.usuario = usuario
102             self.dias_retraso = dias_retraso
103             self.multa = dias_retraso * 5
104             usuario.penalizaciones += self.multa
105
106         def mostrar_penalizacion(self):
107             def mostrar_penalizacion(self):
108                 print(f"{self.usuario.nombre} tiene una multa de ${self.multa} pesos por {self.dias_retraso} días de retraso.")
109
110     class Catalogo:
111         def __init__(self):
112             self.materiales = []
113
114         def agregar_material(self, material):
115             self.materiales.append(material)
116
117         def buscar_por_titulo(self, titulo):
118             return [m for m in self.materiales if titulo.lower() in m.titulo.lower()]
119
120         def mostrar_materiales(self):
121             for material in self.materiales:
122                 print(f"{material.titulo} - Estado: {material.estado}")
123
124     # Ejemplo de uso
125     sucursal1 = Sucursal("Biblioteca Facultad de Ingeniería")
126     sucursal2 = Sucursal("Biblioteca Facultad de Medicina")
127
128     bibliotecario = Bibliotecario("Ana Torres", "ana.torres@universidad.edu", sucursal1)
129     usuario = Usuario("Luis Gómez", "luis.gomez@alumnos.edu")
130
131     libro1 = Libro("Cálculo Integral", "James Stewart", "Matemáticas")
132     revista1 = Revista("Revista de Ciencia y Tecnología", "Marzo 2024", "Trimestral")
133
134     digital1 = MaterialDigital("Introducción a la IA", "PDF", "https://universidad.edu/ia-introduccion.pdf")
135     digital2 = MaterialDigital("Guía de Anatomía Humana", "EPUB", "https://universidad.edu/anatomia.epub")
136
137     bibliotecario.agregar_material(libro1)
138     bibliotecario.agregar_material(revista1)
139
140     usuario.solicitar_prestamo(libro1, bibliotecario)
141     usuario.devolver_material(libro1, bibliotecario)

```

```

124 # Ejemplo de uso
125 sucursal1 = Sucursal("Biblioteca Facultad de Ingeniería")
126 sucursal2 = Sucursal("Biblioteca Facultad de Medicina")
127
128 bibliotecario = Bibliotecario("Ana Torres", "ana.torres@universidad.edu", sucursal1)
129 usuario = Usuario("Luis Gómez", "luis.gomez@alumnos.edu")
130
131 libro1 = Libro("Cálculo Integral", "James Stewart", "Matemáticas")
132 revista1 = Revista("Revista de Ciencia y Tecnología", "Marzo 2024", "Trimestral")
133
134 digital1 = MaterialDigital("Introducción a la IA", "PDF", "https://universidad.edu/ia-introduccion.pdf")
135 digital2 = MaterialDigital("Guía de Anatomía Humana", "EPUB", "https://universidad.edu/anatomia.epub")
136
137 bibliotecario.agregar_material(libro1)
138 bibliotecario.agregar_material(revista1)
139
140 usuario.solicitar_prestamo(libro1, bibliotecario)
141 usuario.devolver_material(libro1, bibliotecario)
142
143 sucursal1.transferir_material(libro1, sucursal2)
144
145 penalizacion = Penalizacion(usuario, 5)
146 penalizacion.mostrar_penalizacion()

```

PROBLEMS 1 OUTPUT DEBUG CONSOLE **TERMINAL** PORTS JUPYTER

Python Debug Console + -

```

PS C:\Users\abrah\OneDrive\Documentos\Nueva carpeta (2)> & 'c:\Users\abrah\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\abrah\.vscode\python.debugpy-2025.0.0-win32-x64\bundled\libs\debugpy\launcher' '59336' '--' 'c:\Users\abrah\OneDrive\Documentos\Nueva carpeta (2)\biblioteca.py'
Luis Gómez ha tomado prestado 'Cálculo Integral'.
Luis Gómez ha devuelto 'Cálculo Integral'.
'Cálculo Integral' ha sido transferido a Biblioteca Facultad de Medicina.
Luis Gómez tiene una multa de $25 pesos por 5 días de retraso.
PS C:\Users\abrah\OneDrive\Documentos\Nueva carpeta (2)>

```