

**Universidad de San Carlos de Guatemala**

Facultad de Ingeniería  
Escuela de Ciencias y Sistemas

**LABORATORIO - IPC 2**

Sección: P | Angel Miguel García Urizar

Segundo Semestre 2025

**Sistema de Biblioteca con POO en  
Python  
MANUAL TÉCNICO**

**Presentado por:**

Wilson Manuel Santos Ajcot

Carné: 2019-07179

santosajcot95@gmail.com

Fecha de entrega: 15/08/2025

Guatemala, Guatemala

# Contenido

## Índice

<b>Introducción Técnica</b>	<b>2</b>
Propósito y Audiencia . . . . .	2
Arquitectura y Metodología . . . . .	2
<b>Objetivos</b>	<b>2</b>
Objetivo General . . . . .	2
Objetivos Específicos . . . . .	2
<b>Especificaciones Técnicas</b>	<b>3</b>
Requisitos de Hardware . . . . .	3
Requisitos de Software . . . . .	3
<b>Arquitectura del Software</b>	<b>4</b>
Diagrama de Clases . . . . .	4
Estructura del Proyecto . . . . .	4
Descripción de Módulos y Clases . . . . .	4
Módulo <code>main.py</code> : . . . . .	5
Módulo <code>sistema_biblioteca.py</code> : . . . . .	5
Paquete <code>materiales</code> : . . . . .	7
<code>material_biblioteca.py</code> : . . . . .	7
<code>libro_fisico.py</code> : . . . . .	7
<code>libro_digital.py</code> : . . . . .	8
Sistema de biblioteca Flujo del Programa . . . . .	10
<b>Implementación y Uso</b>	<b>10</b>
Flujo de Desarrollo . . . . .	10
Uso del Sistema . . . . .	11
Instalación: . . . . .	11
Ejecución: . . . . .	11
Conclusiones Técnicas . . . . .	11
Recomendaciones Técnicas Futuras . . . . .	11
<b>Apéndices</b>	<b>11</b>
Código Fuente Completo . . . . .	11

# Introducción Técnica

## Propósito y Audiencia

Este manual describe los aspectos técnicos del Sistema de Gestión de Biblioteca, incluyendo su arquitectura, diseño y lógica interna. Su propósito es servir como una guía para desarrolladores, personal de mantenimiento y cualquier técnico que necesite comprender el código para su posterior mantenimiento o extensión.

## Arquitectura y Metodología

El sistema se desarrolló con una arquitectura orientada a objetos (POO), aplicando los principios de modularidad, encapsulamiento, herencia, polimorfismo y abstracción. La separación de responsabilidades en módulos y clases facilita su mantenimiento y asegura su escalabilidad.

# Objetivos

## Objetivo General

El objetivo principal del proyecto es aplicar los conceptos de Programación Orientada a Objetos (POO) en Python para desarrollar un sistema que gestione préstamos de libros, diferenciando entre materiales físicos y digitales.

## Objetivos Específicos

Se busca alcanzar los siguientes objetivos particulares:

- ▶ **Abstracción:** Modelar un sistema de biblioteca real para comprender cómo la abstracción permite definir estructuras de datos complejas.
- ▶ **Herencia:** Utilizar la herencia para definir diferentes tipos de materiales bibliográficos a partir de una clase base común.
- ▶ **Encapsulamiento:** Implementar una clase base que encapsule la información fundamental de los materiales, protegiendo sus atributos internos.
- ▶ **Polimorfismo:** Aplicar el polimorfismo para personalizar el comportamiento (ej. días de préstamo) según el tipo de libro, usando una única interfaz.

# Especificaciones Técnicas

## Requisitos de Hardware

Aquí se detallan los requisitos mínimos de hardware para ejecutar el sistema.

- ▶ **Procesador:** *Intel Core i3 o superior*
- ▶ **Memoria RAM:** *Mínimo 4 GB.*
- ▶ **Almacenamiento:** *Disco duro de 250 GB o superior.*

## Requisitos de Software

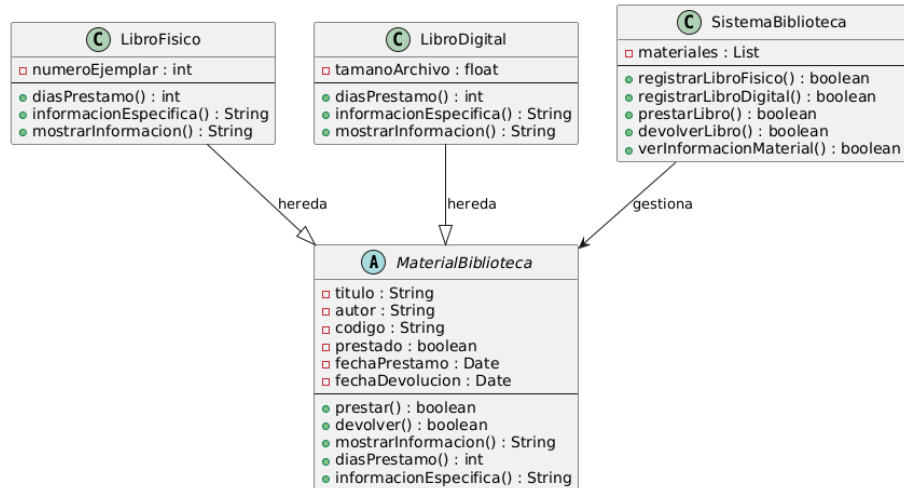
Se listan las dependencias de software y las versiones requeridas para el correcto funcionamiento del sistema y para el entorno de desarrollo.

- ▶ **Sistema Operativo:** El sistema ha sido probado y se garantiza su compatibilidad con las siguientes plataformas de 64 bits:
  - Windows 10 o superior
  - Distribuciones de Linux Ubuntu 20.04 LTS)
- ▶ **Lenguaje de Programación:** Python 3.9 o superior.
- ▶ **Librerías de Python:** Las siguientes librerías estándar son necesarias para la ejecución del programa. Se incluyen en la instalación por defecto de Python y no requieren pasos de instalación adicionales:
  - **random:** Utilizada para la generación de códigos únicos de materiales.
  - **string:** Empleada en conjunto con **random** para generar cadenas alfanuméricas.
  - **datetime:** Fundamenta la gestión de fechas y la duración de los préstamos.
  - **abc:** Esencial para la definición de la clase base abstracta **MaterialBiblioteca**.

# Arquitectura del Software

## Diagrama de Clases

La arquitectura del software se basa en un diseño orientado a objetos, donde las responsabilidades están claramente separadas en clases y módulos. El siguiente diagrama de clases ilustra la estructura principal del sistema y las relaciones entre sus componentes.



**Figura 1:** Diagrama de clases UML del Sistema de Biblioteca

## Estructura del Proyecto

El proyecto se ha organizado de forma modular para mejorar la legibilidad y el mantenimiento. La estructura de directorios es la siguiente:



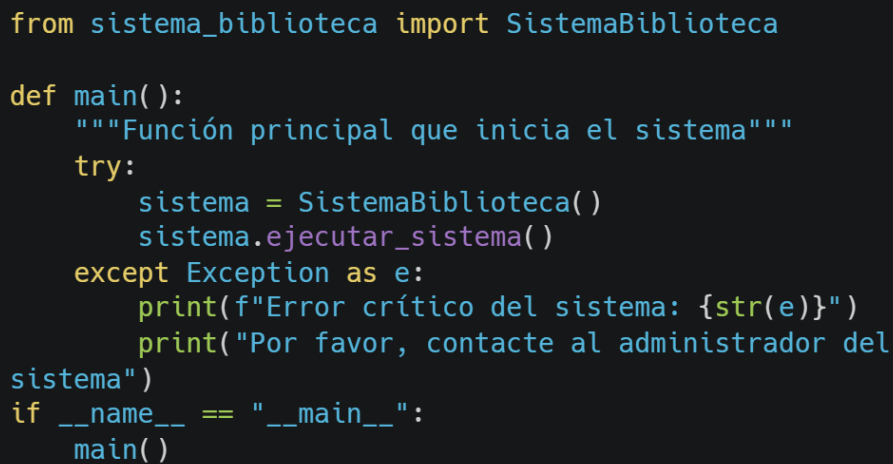
**Figura 2:** Estructura de la práctica

## Descripción de Módulos y Clases

A continuación se describe la funcionalidad de cada módulo y las clases que lo componen, explicando su rol en la arquitectura del sistema.

### Módulo `main.py`:

Este es el punto de entrada principal del sistema. Su única responsabilidad es inicializar la aplicación. Importa la clase `SistemaBiblioteca` y la ejecuta, manteniendo el código principal limpio y desacoplado de la lógica de negocio. Esto facilita la ejecución y prueba del sistema.



```
from sistema_biblioteca import SistemaBiblioteca

def main():
    """Función principal que inicia el sistema"""
    try:
        sistema = SistemaBiblioteca()
        sistema.ejecutar_sistema()
    except Exception as e:
        print(f"Error crítico del sistema: {str(e)}")
        print("Por favor, contacte al administrador del sistema")
if __name__ == "__main__":
    main()
```

**Figura 3:** Código del módulo `main.py`

## Módulo sistema\_biblioteca.py:

Este es el cerebro del sistema. Contiene la clase **SistemaBiblioteca**, que es la responsable de la lógica de negocio y la gestión de todos los materiales. Sus métodos encapsulan las funcionalidades principales como el registro de materiales, préstamos, devoluciones y listado, interactuando con las clases de materiales sin necesidad de conocer sus detalles internos.

```

1  # -*- coding: utf-8 -*-
2  from django.shortcuts import render, HttpResponseRedirect
3  from django.urls import reverse, HttpResponseRedirect
4
5  class SistemaDeInformacion:
6
7      """
8      Clase que maneja el sistema completo de la biblioteca.
9      Implementa todas las funcionalidades requeridas.
10
11      """
12
13      def __init__(self):
14          self.usuario_logueado = None
15
16      def registrar_usuario(self, nombre, apellido, correo, password):
17          """
18          Registra un nuevo usuario en el sistema.
19          """
20          # Verificar si el correo ya existe
21          if self.usuario_logueado == correo:
22              return False
23
24          # Crear el usuario
25          usuario = Usuario(nombre, apellido, correo, password)
26          usuario.save()
27
28          # Enviar correo de confirmación
29          self.enviar_correo_confirmacion(usuario)
30
31          return True
32
33      def login_usuario(self, correo, password):
34          """
35          Inicia sesión de un usuario en el sistema.
36          """
37          # Buscar al usuario
38          usuario = Usuario.objects.filter(correo=correo).first()
39
40          # Verificar la contraseña
41          if usuario and usuario.password == password:
42              self.usuario_logueado = correo
43              return usuario
44          return False
45
46      def logout_usuario(self):
47          """
48          Termina la sesión de un usuario.
49          """
50          self.usuario_logueado = None
51
52      def buscar_libros(self, titulo):
53          """
54          Busca libros en la biblioteca por título.
55          """
56          libros = Libro.objects.filter(titulo__icontains=titulo)
57          return libros
58
59      def agregar_libro(self, titulo, autor, genero, fecha_publicacion):
60          """
61          Agrega un nuevo libro a la biblioteca.
62          """
63          # Verificar si el libro ya existe
64          if self.buscar_libros(titulo):
65              return False
66
67          # Crear el libro
68          libro = Libro(titulo, autor, genero, fecha_publicacion)
69          libro.save()
70
71          return True
72
73      def prestar_libro(self, usuario, libro):
74          """
75          Presta un libro a un usuario.
76          """
77          # Verificar si el usuario ya tiene el libro prestado
78          if self.usuario_logueado == usuario.correo:
79              return False
80
81          # Verificar si el libro ya está prestado
82          if libro.prestado:
83              return False
84
85          # Prestar el libro
86          libro.prestado = True
87          libro.usuario_prestado = usuario
88          libro.save()
89
90          return True
91
92      def devolver_libro(self, usuario, libro):
93          """
94          Devuelve un libro a la biblioteca.
95          """
96          # Verificar si el usuario es el que prestó el libro
97          if self.usuario_logueado != usuario.correo:
98              return False
99
100         # Devolver el libro
101         libro.prestado = False
102         libro.usuario_prestado = None
103         libro.save()
104
105         return True
106
107     def actualizar_estado_libro(self, libro, estado):
108         """
109         Actualiza el estado de un libro.
110         """
111         libro.prestado = estado
112         libro.save()
113
114         return True
115
116     def eliminar_libro(self, libro):
117         """
118         Elimina un libro de la biblioteca.
119         """
120         libro.delete()
121
122         return True
123
124     def mostrar_detalle_usuario(self, correo):
125         """
126         Muestra los detalles de un usuario.
127         """
128         usuario = Usuario.objects.filter(correo=correo).first()
129         return usuario
130
131     def mostrar_detalle_libro(self, titulo):
132         """
133         Muestra los detalles de un libro.
134         """
135         libro = Libro.objects.filter(titulo=titulo).first()
136         return libro
137
138     def mostrar_detalle_usuario_y_libro(self, correo, titulo):
139         """
140         Muestra los detalles de un usuario y un libro.
141         """
142         usuario = Usuario.objects.filter(correo=correo).first()
143         libro = Libro.objects.filter(titulo=titulo).first()
144         return usuario, libro
145
146     def mostrar_detalle_usuario_y_libros(self, correo):
147         """
148         Muestra los detalles de un usuario y todos los libros.
149         """
150         usuario = Usuario.objects.filter(correo=correo).first()
151         libros = Libro.objects.all()
152         return usuario, libros
153
154     def mostrar_detalle_usuario_y_libros_prestados(self, correo):
155         """
156         Muestra los detalles de un usuario y todos los libros prestados.
157         """
158         usuario = Usuario.objects.filter(correo=correo).first()
159         libros = Libro.objects.filter(prestado=True)
160         return usuario, libros
161
162     def mostrar_detalle_usuario_y_libros_disponibles(self, correo):
163         """
164         Muestra los detalles de un usuario y todos los libros disponibles.
165         """
166         usuario = Usuario.objects.filter(correo=correo).first()
167         libros = Libro.objects.filter(prestado=False)
168         return usuario, libros
169
170     def mostrar_detalle_usuario_y_libros_disponibles_y_prestados(self, correo):
171         """
172         Muestra los detalles de un usuario y todos los libros disponibles y prestados.
173         """
174         usuario = Usuario.objects.filter(correo=correo).first()
175         libros = Libro.objects.all()
176         return usuario, libros
177
178     def mostrar_detalle_usuario_y_libros_disponibles_y_prestados_y_disponibles(self, correo):
179         """
180         Muestra los detalles de un usuario y todos los libros disponibles, prestados y disponibles.
181         """
182         usuario = Usuario.objects.filter(correo=correo).first()
183         libros = Libro.objects.all()
184         return usuario, libros
185
186     def mostrar_detalle_usuario_y_libros_disponibles_y_prestados_y_disponibles_y_disponibles(self, correo):
187         """
188         Muestra los detalles de un usuario y todos los libros disponibles, prestados, disponibles y disponibles.
189         """
190         usuario = Usuario.objects.filter(correo=correo).first()
191         libros = Libro.objects.all()
192         return usuario, libros
193
194     def mostrar_detalle_usuario_y_libros_disponibles_y_prestados_y_disponibles_y_disponibles_y_disponibles(self, correo):
195         """
196         Muestra los detalles de un usuario y todos los libros disponibles, prestados, disponibles, disponibles y disponibles.
197         """
198         usuario = Usuario.objects.filter(correo=correo).first()
199         libros = Libro.objects.all()
200         return usuario, libros
201
202     def mostrar_detalle_usuario_y_libros_disponibles_y_prestados_y_disponibles_y_disponibles_y_disponibles_y_disponibles(self, correo):
203         """
204         Muestra los detalles de un usuario y todos los libros disponibles, prestados, disponibles, disponibles, disponibles y disponibles.
205         """
206         usuario = Usuario.objects.filter(correo=correo).first()
207         libros = Libro.objects.all()
208         return usuario, libros
209
210     def mostrar_detalle_usuario_y_libros_disponibles_y_prestados_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles(self, correo):
211         """
212         Muestra los detalles de un usuario y todos los libros disponibles, prestados, disponibles, disponibles, disponibles, disponibles y disponibles.
213         """
214         usuario = Usuario.objects.filter(correo=correo).first()
215         libros = Libro.objects.all()
216         return usuario, libros
217
218     def mostrar_detalle_usuario_y_libros_disponibles_y_prestados_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles(self, correo):
219         """
220         Muestra los detalles de un usuario y todos los libros disponibles, prestados, disponibles, disponibles, disponibles, disponibles, disponibles y disponibles.
221         """
222         usuario = Usuario.objects.filter(correo=correo).first()
223         libros = Libro.objects.all()
224         return usuario, libros
225
226     def mostrar_detalle_usuario_y_libros_disponibles_y_prestados_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles(self, correo):
227         """
228         Muestra los detalles de un usuario y todos los libros disponibles, prestados, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles y disponibles.
229         """
230         usuario = Usuario.objects.filter(correo=correo).first()
231         libros = Libro.objects.all()
232         return usuario, libros
233
234     def mostrar_detalle_usuario_y_libros_disponibles_y_prestados_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles(self, correo):
235         """
236         Muestra los detalles de un usuario y todos los libros disponibles, prestados, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles y disponibles.
237         """
238         usuario = Usuario.objects.filter(correo=correo).first()
239         libros = Libro.objects.all()
240         return usuario, libros
241
242     def mostrar_detalle_usuario_y_libros_disponibles_y_prestados_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles(self, correo):
243         """
244         Muestra los detalles de un usuario y todos los libros disponibles, prestados, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles y disponibles.
245         """
246         usuario = Usuario.objects.filter(correo=correo).first()
247         libros = Libro.objects.all()
248         return usuario, libros
249
250     def mostrar_detalle_usuario_y_libros_disponibles_y_prestados_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles(self, correo):
251         """
252         Muestra los detalles de un usuario y todos los libros disponibles, prestados, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles y disponibles.
253         """
254         usuario = Usuario.objects.filter(correo=correo).first()
255         libros = Libro.objects.all()
256         return usuario, libros
257
258     def mostrar_detalle_usuario_y_libros_disponibles_y_prestados_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles(self, correo):
259         """
260         Muestra los detalles de un usuario y todos los libros disponibles, prestados, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles y disponibles.
261         """
262         usuario = Usuario.objects.filter(correo=correo).first()
263         libros = Libro.objects.all()
264         return usuario, libros
265
266     def mostrar_detalle_usuario_y_libros_disponibles_y_prestados_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles(self, correo):
267         """
268         Muestra los detalles de un usuario y todos los libros disponibles, prestados, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles y disponibles.
269         """
270         usuario = Usuario.objects.filter(correo=correo).first()
271         libros = Libro.objects.all()
272         return usuario, libros
273
274     def mostrar_detalle_usuario_y_libros_disponibles_y_prestados_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles(self, correo):
275         """
276         Muestra los detalles de un usuario y todos los libros disponibles, prestados, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles y disponibles.
277         """
278         usuario = Usuario.objects.filter(correo=correo).first()
279         libros = Libro.objects.all()
280         return usuario, libros
281
282     def mostrar_detalle_usuario_y_libros_disponibles_y_prestados_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles(self, correo):
283         """
284         Muestra los detalles de un usuario y todos los libros disponibles, prestados, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles y disponibles.
285         """
286         usuario = Usuario.objects.filter(correo=correo).first()
287         libros = Libro.objects.all()
288         return usuario, libros
289
290     def mostrar_detalle_usuario_y_libros_disponibles_y_prestados_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles(self, correo):
291         """
292         Muestra los detalles de un usuario y todos los libros disponibles, prestados, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles y disponibles.
293         """
294         usuario = Usuario.objects.filter(correo=correo).first()
295         libros = Libro.objects.all()
296         return usuario, libros
297
298     def mostrar_detalle_usuario_y_libros_disponibles_y_prestados_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles(self, correo):
299         """
300         Muestra los detalles de un usuario y todos los libros disponibles, prestados, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles, disponibles y disponibles.
301         """
302         usuario = Usuario.objects.filter(correo=correo).first()
303         libros = Libro.objects.all()
304         return usuario, libros
305
306     def mostrar_detalle_usuario_y_libros_disponibles_y_prestados_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles_y_disponibles(self, correo):
307         """
308         Muestra los detalles de
```

**Figura 4:** Código del módulo sistema biblioteca.py

## Paquete materiales:

Este paquete agrupa las clases que representan los diferentes tipos de materiales bibliográficos. Su propósito es centralizar la definición de los materiales y sus comportamientos.

### material\_biblioteca.py:

Define la clase base abstracta **MaterialBiblioteca**. Actúa como un contrato para todas las subclases, asegurando que cualquier material futuro (físico, digital, etc.) tendrá los métodos y atributos comunes (título, autor, préstamo, etc.), promoviendo la consistencia y el polimorfismo.

```
# materiales/material_biblioteca.py
import random
import string
from abc import ABC, abstractmethod
from datetime import datetime, timedelta

class MaterialBiblioteca(ABC):
    """
    Clase abstracta que representa un material genérico de biblioteca.
    Implementa el concepto de Abstracción definiendo la estructura común
    para todos los materiales bibliográficos.
    """

    def __init__(self, titulo, autor):
        """
        Método de inicialización. Implementa Encapsulamiento
        """
        self.__titulo = titulo
        self.__autor = autor
        self.__codigo = self.__generar_codigo()
        self.__prestado = False
        self.__fecha_prestamo = None
        self.__fecha_devolucion = None

    def __generar_codigo(self):
        """
        Genera un código único de 8 caracteres alfanuméricos
        """
        return ''.join(random.choices(string.ascii_uppercase + string.digits, k=8))

    # Getters
    @property
    def titulo(self):
        return self.__titulo

    @property
    def autor(self):
        return self.__autor

    @property
    def codigo(self):
        return self.__codigo

    @property
    def prestado(self):
        return self.__prestado

    @property
    def fecha_prestamo(self):
        return self.__fecha_prestamo

    @property
    def fecha_devolucion(self):
        return self.__fecha_devolucion

    # Setters
    @titulo.setter
    def titulo(self, nuevo_titulo):
        if nuevo_titulo and nuevo_titulo.strip():
            self.__titulo = nuevo_titulo.strip()
        else:
            raise ValueError("El título no puede estar vacío")

    @autor.setter
    def autor(self, nuevo_autor):
        if nuevo_autor and nuevo_autor.strip():
            self.__autor = nuevo_autor.strip()
        else:
            raise ValueError("El autor no puede estar vacío")

    def prestar(self):
        """
        Presta el material si está disponible
        """
        if not self.__prestado:
            self.__prestado = True
            self.__fecha_prestamo = datetime.now()
            self.__fecha_devolucion = self.__fecha_prestamo + timedelta(days=self.dias_prestamo())
            return True
        return False

    def devolver(self):
        """
        Devuelve el material si está prestado
        """
        if self.__prestado:
            self.__prestado = False
            self.__fecha_prestamo = None
            self.__fecha_devolucion = None
            return True
        return False

    def mostrar_informacion(self):
        """
        Muestra información básica del material
        """
        estado = "Prestado" if self.__prestado else "Disponible"
        info = f"""
        INFORMACIÓN GENERAL
        Título: {self.__titulo}<2> |
        Autor: {self.__autor}<2> |
        Código: {self.__codigo}<2> |
        Estado: {estado}<2> """

        if self.__prestado and self.__fecha_devolucion:
            fecha_dev = self.__fecha_devolucion.strftime("%d/%m/%Y")
            info += f"<br>Info = {info} Devolver antes: {fecha_dev}<2> "

        info = f"<br>Info: {info}"
        return info

    @abstractmethod
    def dias_prestamo(self):
        """
        Define los días máximos de préstamo según el tipo de material
        """
        pass

    @abstractmethod
    def informacion_especifica(self):
        """
        Información específica según el tipo de material
        """
        pass
```

Figura 5: Código de la clase MaterialBiblioteca



**libro\_fisico.py:**

Implementa la clase `LibroFisico`. Hereda de `MaterialBiblioteca` y añade la lógica específica para un libro físico, como su número de ejemplar y la duración de su préstamo. Un programador puede entender fácilmente cómo se comportan los libros físicos al ver esta clase.

```
# materiales/libro_fisico.py

from material_biblioteca import MaterialBiblioteca

class LibroFisico(MaterialBiblioteca):
    """
    Clase que representa un libro físico.
    Hereda de MaterialBiblioteca e implementa características específicas.
    """

    def __init__(self, titulo, autor, numero_ejemplar):
        super().__init__(titulo, autor)
        self.__numero_ejemplar = numero_ejemplar

    @property
    def numero_ejemplar(self):
        return self.__numero_ejemplar

    @numero_ejemplar.setter
    def numero_ejemplar(self, nuevo_numero):
        if isinstance(nuevo_numero, int) and nuevo_numero > 0:
            self.__numero_ejemplar = nuevo_numero
        else:
            raise ValueError("El número de ejemplar debe ser un entero positivo")

    def dias_prestamo(self):
        """Los libros físicos se prestan por 7 días máximo"""
        return 7

    def informacion_especifica(self):
        """Información específica del libro físico"""
        return f"Número de Ejemplar: {self.__numero_ejemplar}"

    def mostrar_informacion(self):
        """Muestra información completa del libro físico"""
        info_base = super().mostrar_informacion()
        info_especifica = info_base.replace("|", "INFORMACIÓN GENERAL", 1)
        info_especifica = info_especifica.replace("|", "LIBRO FÍSICO", 1)

        lineas = info_especifica.split('\n')
        lineas.insert(-1, f"| Ejemplar Nº: {self.__numero_ejemplar:<15} |")
        lineas.insert(-1, f"| Días préstamo: {self.dias_prestamo():<13} |")

        return '\n'.join(lineas)
```

**Figura 6:** Código de la clase `LibroFisico`

**libro\_digital.py:**

Implementa la clase LibroDigital. Similar a la clase anterior, hereda de MaterialBiblioteca para definir las características únicas de un libro digital, como el tamaño del archivo y la duración de su préstamo.

```
# materiales/libro_digital.py

from .material_biblioteca import MaterialBiblioteca

class LibroDigital(MaterialBiblioteca):
    """
    Clase que representa un libro digital.
    Hereda de MaterialBiblioteca e implementa características específicas.
    """

    def __init__(self, titulo, autor, tamaño_archivo):
        super().__init__(titulo, autor)
        self.__tamaño_archivo = tamaño_archivo

    @property
    def tamaño_archivo(self):
        return self.__tamaño_archivo

    @tamaño_archivo.setter
    def tamaño_archivo(self, nuevo_tamaño):
        if isinstance(nuevo_tamaño, (int, float)) and nuevo_tamaño > 0:
            self.__tamaño_archivo = nuevo_tamaño
        else:
            raise ValueError("El tamaño del archivo debe ser un número positivo")

    def dias_prestamo(self):
        """Los libros digitales se prestan por 3 días máximo"""
        return 3

    def informacion_especifica(self):
        """Información específica del libro digital"""
        return f"Tamaño de Archivo: {self.__tamaño_archivo} MB"

    def mostrar_informacion(self):
        """Muestra información completa del libro digital"""
        info_base = super().mostrar_informacion()
        info_especifica = info_base.replace("
",
        "
        INFORMACIÓN GENERAL
        LIBRO DIGITAL
        ")

        lineas = info_especifica.split('\n')
        lineas.insert(-1, f"| Archivo: {self.__tamaño_archivo:<18} MB |")
        lineas.insert(-1, f"| Días préstamo: {self.dias_prestamo():<13} |")

        return '\n'.join(lineas)
```

**Figura 7:** Código de la clase LibroDigital

## Sistema de biblioteca Flujo del Programa

La lógica de la aplicación sigue un flujo de menú interactivo. El siguiente diagrama ilustra la secuencia de interacciones del usuario.

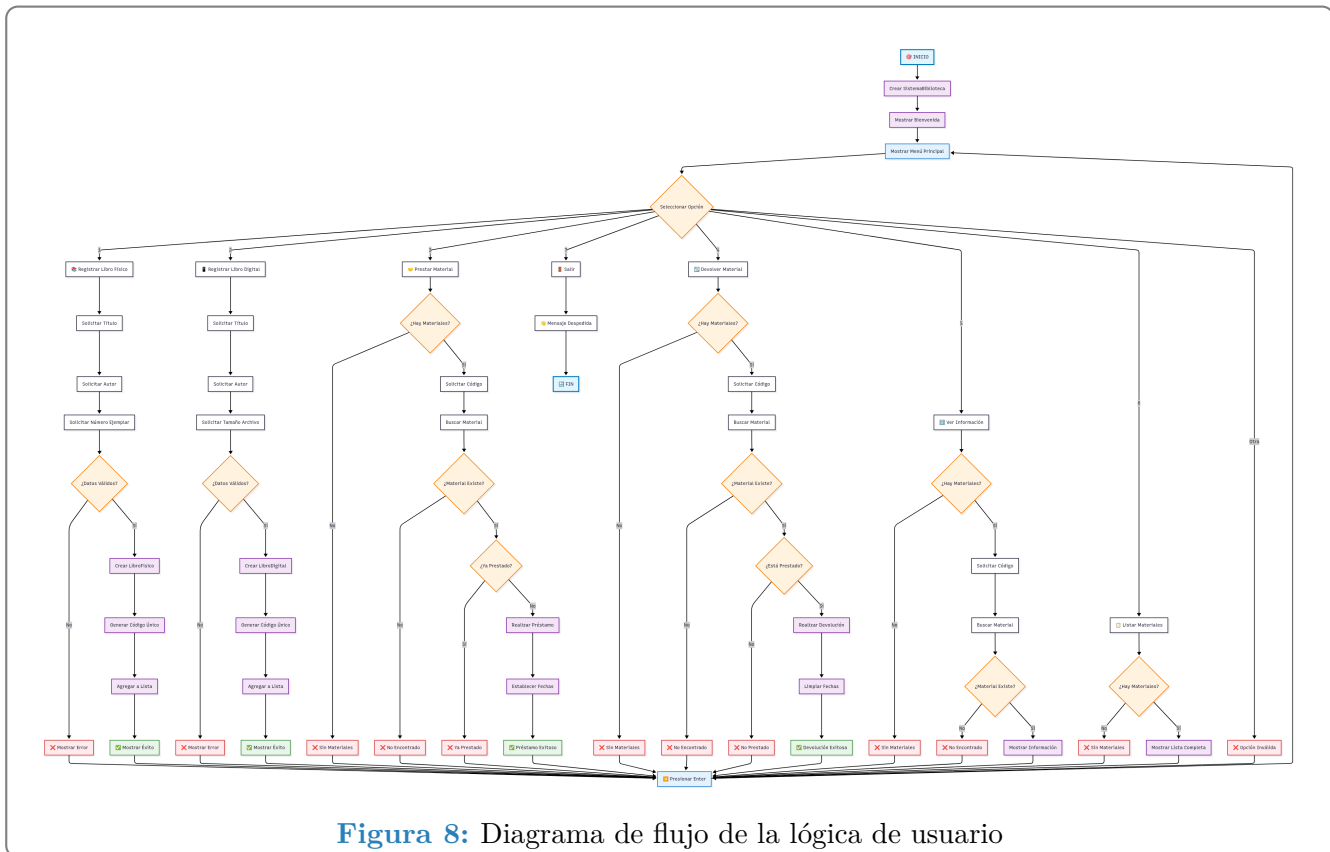


Figura 8: Diagrama de flujo de la lógica de usuario

## Implementación y Uso

### Flujo de Desarrollo

Para el desarrollo, se recomienda seguir los siguientes pasos:

- ▶ Clonar el repositorio de Git.
- ▶ Asegurarse de tener Python 3.x instalado.
- ▶ Abrir el proyecto en un IDE como Visual Studio Code o PyCharm.
- ▶ Las clases se encuentran en sus respectivos módulos para una fácil edición.

## Uso del Sistema

### Instalación:

1. Descargar la carpeta del proyecto.
2. Abrir una terminal en el directorio raíz del proyecto.

### Ejecución:

1. Ejecutar el script principal con el comando: `python main.py`
2. El menú interactivo aparecerá en la consola, permitiendo al usuario gestionar la biblioteca.

## Conclusiones Técnicas

- ▶ La modularización del código demostró ser una práctica efectiva para organizar las responsabilidades del sistema.
- ▶ El uso de la herencia y el polimorfismo permitió crear una base de código extensible y reutilizable para futuros tipos de materiales.

## Recomendaciones Técnicas Futuras

- ▶ Agregar una capa de persistencia para guardar los datos de la biblioteca en un archivo o base de datos.
- ▶ Implementar un sistema de pruebas unitarias automatizado para garantizar la fiabilidad del código.
- ▶ Extender la funcionalidad para incluir más tipos de materiales (revistas, periódicos, etc.).

## Apéndices

### Código Fuente Completo

El código fuente completo del proyecto está disponible en el repositorio de GitHub. Para acceder, clona el siguiente repositorio:

[https://github.com/WilsonJrSantos/IPC2\\_Practica1\\_201907179](https://github.com/WilsonJrSantos/IPC2_Practica1_201907179)