

**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

Introducción Técnica	3
Propósito y Audiencia	3
Arquitectura y Metodología	3
Objetivos	3
Objetivo General	3
Objetivos Específicos	3
Especificaciones Técnicas	3
Requisitos de Hardware	3
Requisitos de Software	4
Arquitectura del Software	5
Diagrama de Clases	5
Estructura del Proyecto	5
Descripción de Módulos y Clases	6
Módulo <code>main.py</code> :	6
Módulo <code>sistema_biblioteca.py</code> :	7
Paquete <code>materiales</code> :	9
<code>material_biblioteca.py</code> :	9
<code>libro_fisico.py</code> :	9
<code>libro_digital.py</code> :	10
Diagrama de Flujo del Programa	12
Implementación y Uso	12
Flujo de Desarrollo	12
Uso del Sistema	12
Conclusiones y Recomendaciones	13
Conclusiones Técnicas	13
Recomendaciones Técnicas Futuras	13
Apéndices	13
Código Fuente Completo	13

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.

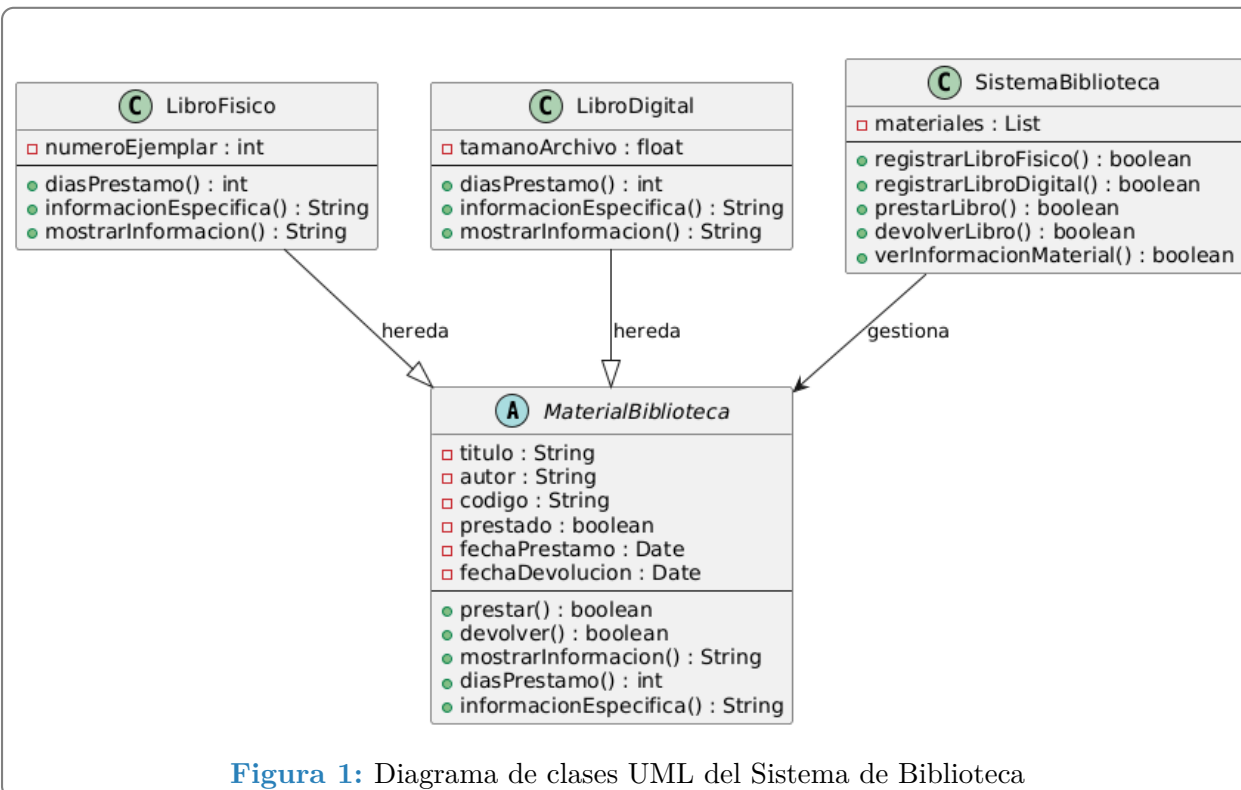
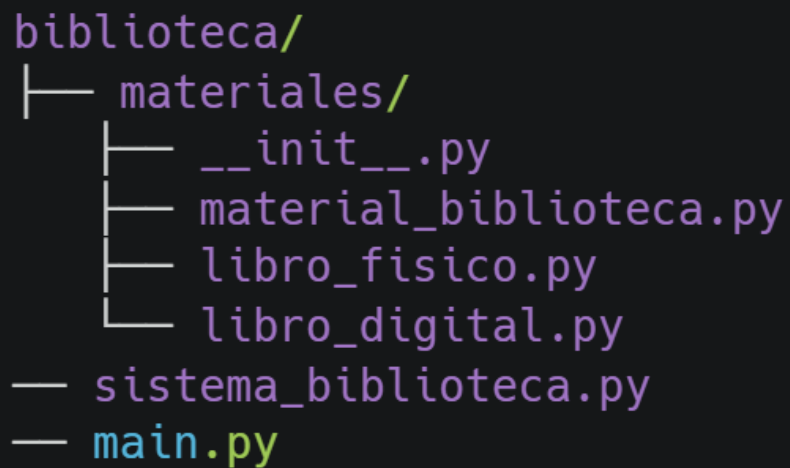


Figura 1: Diagrama de clases UML del Sistema de Biblioteca

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:



```
biblioteca/  
├── materiales/  
│   ├── __init__.py  
│   ├── material_biblioteca.py  
│   ├── libro_fisico.py  
│   └── libro_digital.py  
└── sistema_biblioteca.py  
└── main.py
```

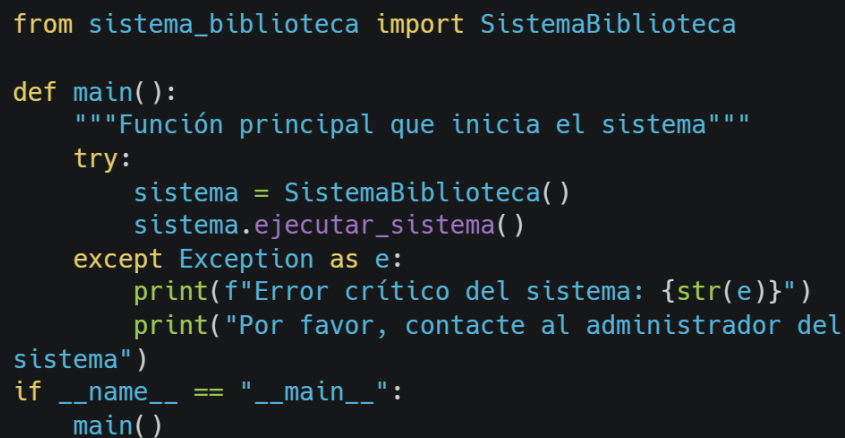
Figura 2: Estructura de la práctica

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.

```

"""
Módulo sistema_biblioteca.py
"""
from materiales.libro_fisico import LibroFisico
from materiales.libro_digital import LibroDigital
from sistema_biblioteca import SistemaBiblioteca

class SistemaBiblioteca:
    """
    Clase que gestiona el sistema completo de la biblioteca.
    Encapsula todas las funcionalidades requeridas.
    """

    def __init__(self):
        self._materiales = {}

    def registrar_libro_fisico(self):
        """
        Registra un nuevo libro físico en el sistema.
        """
        try:
            titulo = input("Ingrese el título del libro: ")
            autor = input("Ingrese el autor del libro: ")
            isbn = input("Ingrese el ISBN del libro: ")
            if not titulo:
                print("Error: El título no puede estar vacío.")
                return False
            if not autor:
                print("Error: El autor no puede estar vacío.")
                return False
            if not isbn:
                print("Error: El ISBN no puede estar vacío.")
                return False
            libro = LibroFisico(titulo, autor, isbn)
            self._materiales[isbn] = libro
            print(f"Libro físico registrado exitosamente.")
            return True
        except Exception as e:
            print(f"Error al registrar el libro físico: {e}")
            return False

    def registrar_libro_digital(self):
        """
        Registra un nuevo libro digital en el sistema.
        """
        try:
            titulo = input("Ingrese el título del libro: ")
            autor = input("Ingrese el autor del libro: ")
            isbn = input("Ingrese el ISBN del libro: ")
            if not titulo:
                print("Error: El título no puede estar vacío.")
                return False
            if not autor:
                print("Error: El autor no puede estar vacío.")
                return False
            if not isbn:
                print("Error: El ISBN no puede estar vacío.")
                return False
            libro = LibroDigital(titulo, autor, isbn)
            self._materiales[isbn] = libro
            print(f"Libro digital registrado exitosamente.")
            return True
        except Exception as e:
            print(f"Error al registrar el libro digital: {e}")
            return False

    def prestar_libro(self, isbn):
        """
        Presta un libro físico o digital.
        """
        if isbn not in self._materiales:
            print("Error: El libro no está registrado en la biblioteca.")
            return False
        material = self._materiales[isbn]
        if material.prestado:
            print("Error: El libro ya está prestado.")
            return False
        material.prestado = True
        print(f"Libro {material.titulo} prestado exitosamente.")
        return True

    def devolver_libro(self, isbn):
        """
        Devuelve un libro físico o digital.
        """
        if isbn not in self._materiales:
            print("Error: El libro no está registrado en la biblioteca.")
            return False
        material = self._materiales[isbn]
        if not material.prestado:
            print("Error: El libro no está prestado.")
            return False
        material.prestado = False
        print(f"Libro {material.titulo} devuelto exitosamente.")
        return True

    def listar_materiales(self):
        """
        Lista todos los materiales registrados.
        """
        if not self._materiales:
            print("No hay materiales registrados en la biblioteca.")
            return
        for isbn, material in self._materiales.items():
            print(f"ISBN: {isbn} | Título: {material.titulo} | Autor: {material.autor} | Estado: {material.prestado}")

    def buscar_materiales(self, criterio):
        """
        Busca materiales por criterio.
        """
        resultados = []
        for isbn, material in self._materiales.items():
            if criterio == "titulo" and material.titulo.lower().find(criterio.lower()) != -1:
                resultados.append(material)
            elif criterio == "autor" and material.autor.lower().find(criterio.lower()) != -1:
                resultados.append(material)
            elif criterio == "isbn" and material.isbn == criterio:
                resultados.append(material)
        return resultados

    def menu(self):
        """
        Muestra el menú principal del sistema.
        """
        while True:
            print("\nMenú Principal del Sistema")
            print("1. Registrar libro físico")
            print("2. Registrar libro digital")
            print("3. Prestar libro")
            print("4. Devolver libro")
            print("5. Listar todos los materiales")
            print("6. Buscar materiales")
            print("7. Salir")
            opcion = input("Seleccione una opción (1-7): ")
            if opcion == "1":
                self.registrar_libro_fisico()
            elif opcion == "2":
                self.registrar_libro_digital()
            elif opcion == "3":
                isbn = input("Ingrese el ISBN del libro a prestar: ")
                self.prestar_libro(isbn)
            elif opcion == "4":
                isbn = input("Ingrese el ISBN del libro a devolver: ")
                self.devolver_libro(isbn)
            elif opcion == "5":
                self.listar_materiales()
            elif opcion == "6":
                criterio = input("Ingrese el criterio de búsqueda: ")
                resultados = self.buscar_materiales(criterio)
                if resultados:
                    for material in resultados:
                        print(f"ISBN: {material.isbn} | Título: {material.titulo} | Autor: {material.autor} | Estado: {material.prestado}")
                else:
                    print("No se encontraron resultados.")
            elif opcion == "7":
                break
            else:
                print("Opción inválida. Por favor, seleccione una opción del 1 al 7.")
        print("Fin del programa. Gracias por usar el sistema de biblioteca.")

```

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 cedido de biblioteca.
    Implementa el concepto de Abstracción definiendo la estructura común
    para todos los materiales bibliográficos.
    """
    def __init__(self, titulo, autor):
        """Inicia el objeto. 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))

    @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

    @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}<3> |
        Autor: {self.__autor}<3> |
        Código: {self.__codigo}<3> |
        Estado: {estado}<3> |"""
        if self.__prestado and self.__fecha_devolucion:
            fecha_dev = self.__fecha_devolucion.strftime("%d/%m/%Y")
            info += f"<3> Fecha de Devolución antes: {fecha_dev}<3> |>
        info += "<3>
        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 |",
                                           " | LIBRO FÍSICO |")

        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 |", 1)
        info_especifica = info_especifica.replace(" |", " | LIBRO DIGITAL |", 1)

        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`

Diagrama de Flujo del Programa

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

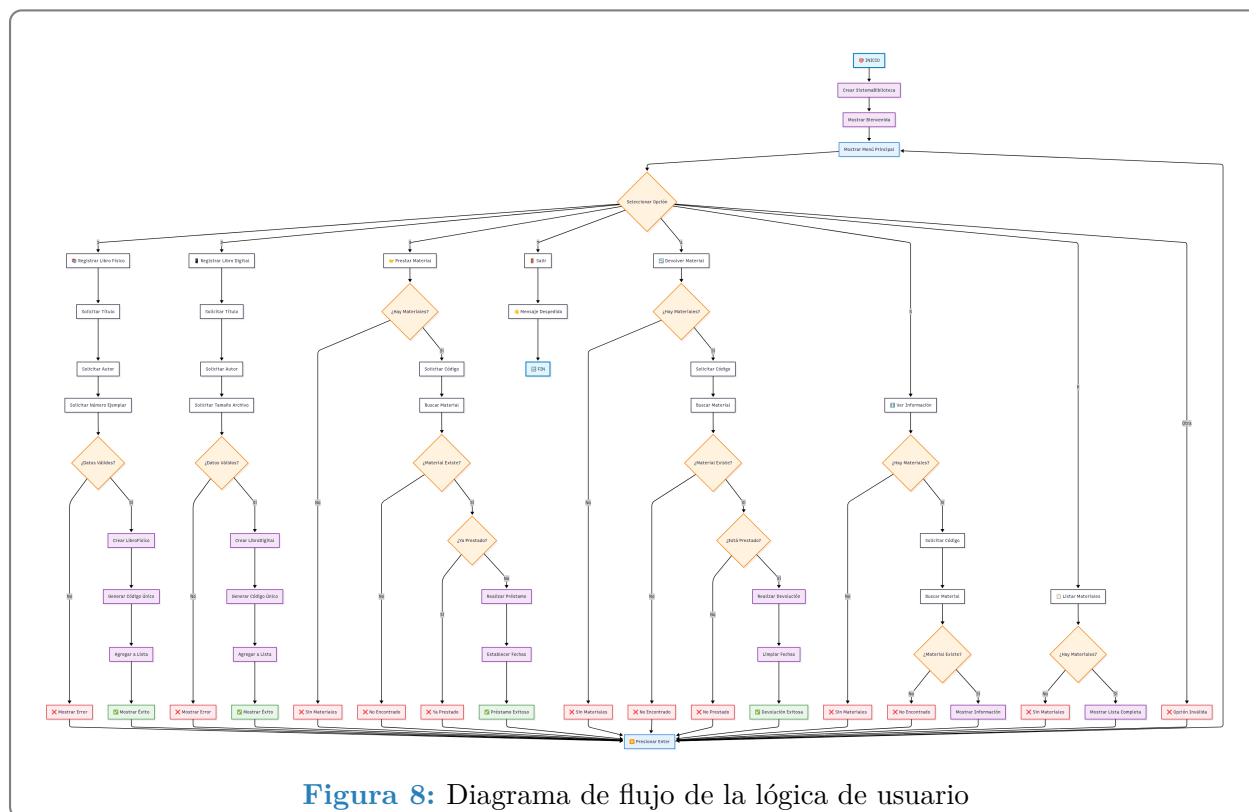


Figura 8: Diagrama de flujo de la lógica de usuario

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 y Recomendaciones

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