Manual Técnico: Sistema de Biblioteca Digital

Curso: Lenguajes Formales y Autómatas

Implementación: Python

---

1. Introducción y Arquitectura del Programa

Objetivo del Programa

Este programa es un simulador de consola diseñado para gestionar y analizar registros de préstamos de una biblioteca. Su función principal es leer un archivo de texto (.lfa) con formato específico, procesar la información y generar una serie de reportes informativos tanto en la consola como en formato HTML.

Arquitectura Modular

El programa está diseñado siguiendo un enfoque modular, donde cada archivo (.py) tiene una responsabilidad clara y definida. Esta separación de responsabilidades hace que el código sea más organizado, fácil de entender y mantener.

La arquitectura se divide en cuatro capas principales:

1. **Modelos (modelos.py):** Define la estructura de nuestros datos.

2. **Parser (lector_lfa.py):** Se encarga de leer y traducir el archivo de entrada.

3. **Servicio (servicio_biblioteca.py):** Contiene toda la lógica de negocio y el manejo de los datos en memoria.

4. **Interfaz y Punto de Entrada (main.py):** Controla la interacción con el usuario y orquesta el flujo del programa.

---

2. Descripción de los Módulos

modelos.py - Los Moldes de Datos

Este archivo es la base de todo. Define las "plantillas" o "moldes" para la información que manejamos. En Python, utilizamos dataclasses para crear estas plantillas de forma eficiente.

- @dataclass: Es un decorador que automáticamente le añade a la clase métodos básicos como el constructor (__init__), lo que nos ahorra escribir código repetitivo.

- Prestamo: Cada objeto de esta clase representa una línea del archivo .lfa, con todos sus campos (ID de usuario, título, fechas, etc.).

- Usuario y Libro: Son clases más simples usadas para los reportes de datos únicos. Al definirlas como frozen=True, nos aseguramos de que puedan ser añadidas a conjuntos (sets) para eliminar duplicados de forma automática y eficiente.

---

lector_lfa.py - El Traductor Inteligente

Este módulo tiene la tarea más crítica y delicada: actuar como un analizador léxico básico (parser). Su única responsabilidad es leer el archivo .lfa y convertir su contenido de texto plano a una lista de objetos Prestamo que el resto del programa pueda entender.

- Lectura Segura: Utiliza with open(...) para garantizar que el archivo se cierre correctamente, incluso si ocurren errores.

- Procesamiento Línea por Línea: Recorre el archivo, ignorando la cabecera y las líneas en blanco.

- Manejo de Errores Robusto: Usa un bloque try-except para cada línea. Si una línea tiene un formato incorrecto (ej. un ID que no es número, una fecha mal escrita, o un número incorrecto de comas), el programa no se detiene. En su lugar, imprime un mensaje de error detallado en la consola (sys.stderr) y continúa con la siguiente línea, cumpliendo con un requisito clave del proyecto.

---

servicio_biblioteca.py - El Cerebro de la Operación

Esta es la capa de lógica de negocio. La clase BibliotecaService actúa como el "cerebro" que almacena todos los datos en la memoria una vez que han sido leídos por el lector_lfa.

- Almacenamiento en Memoria: Guarda todos los objetos Prestamo en una lista interna (self._prestamos).

- Lógica de Reportes: Contiene todos los métodos para procesar esa lista y generar los datos necesarios para los reportes:

- get_usuarios_unicos(): Utiliza un conjunto (set) para devolver una lista de usuarios sin repeticiones.

- get_prestamos_no_devueltos(): Utiliza una comprensión de listas para filtrar y devolver solo los préstamos sin fecha de devolución.

- get_estadisticas(): Es el método más avanzado. Usa la clase Counter del módulo collections de Python, una herramienta altamente optimizada para contar elementos en una lista. Esto permite encontrar el libro más prestado y el usuario más activo de forma muy eficiente.

---

main.py - El Director de Orquesta y la Interfaz

Este archivo es el punto de entrada que el usuario ejecuta. Tiene dos funciones principales:

1. Interfaz de Usuario: Presenta el menú principal en un bucle while True, captura la opción del usuario y la valida.

2. Orquestador: No contiene lógica de negocio. Su trabajo es "dirigir la orquesta": llama a los métodos de BibliotecaService para obtener los datos y luego (si es necesario) le pasa esos datos a generador_reportes para exportarlos.

El bloque if __name__ == "__main__": al final del archivo es una convención de Python que asegura que el código del menú solo se ejecute cuando el archivo es el programa principal.

---

3. Diagrama de Flujo del Programa

El flujo de ejecución principal del programa sigue una estructura de menú simple e interactivo.

(No se incluye el diagrama mermaid en PDF, solo descripción).

---

4. Conclusión

El programa utiliza las fortalezas de Python para crear una solución robusta y legible. La separación clara de responsabilidades, el manejo de errores resiliente y el uso de estructuras de datos eficientes (set, Counter) permiten que el programa cumpla con todos los requisitos del proyecto de una manera elegante y efectiva.