📘 Parcial – Estructura de Datos (Unidad 1)

Segunda parte (práctica): Diseña y codifica en **Java** un programa que integre **todos los temas vistos en la Unidad 1**: fundamentos de estructuras de datos, **arrays** (1D y 2D), **listas enlazadas** (simple y doble) y **programación estructurada** (secuencia, selección, iteración y modularidad). Entrega en un solo repositorio/carpeta con el código fuente y un README breve.

🎯 Objetivo

Construir un **sistema de gestión de biblioteca** que permita administrar un **catálogo** de libros, los **préstamos activos** de los usuarios y un **historial navegable** de operaciones, aplicando correctamente los conceptos de la Unidad 1.

🧩 Contexto del problema

La biblioteca “U1” necesita:

- Registrar un **catálogo** de libros (código, título, autor, stock).
- Administrar **préstamos** (usuario, código libro, fecha, estado).
- Mantener un **historial** de operaciones (ALTAS/BAJAS/PRÉSTAMOS/DEVOLUCIONES) que se pueda recorrer **hacia adelante y hacia atrás**.
- Consultas típicas: buscar por título/código, listar catálogo, ver préstamos activos, recorrer historial.

🛠️ Requisitos técnicos (por tema)

1) Fundamentos de Estructuras de Datos

- Explica **qué estructura** usas para cada módulo y **por qué** (lineal vs. dinámica; costos de inserción/búsqueda/recorrido).
- Incluye en comentarios del código la **complejidad temporal** de las operaciones principales (por ejemplo, búsqueda lineal O(n) en el catálogo).

2) Arrays y Tipos de Datos

Implementa el **catálogo** usando **arrays**:

- Un **array unidimensional** de `Libro` de **tamaño fijo** (configurable por constante).
- Una **matriz 2D** `int` para registrar **disponibilidad por sucursal** (filas = libros, columnas = sucursales).
- Operaciones mínimas (todas sobre arrays):
  - **Alta** de libro (si hay espacio).
  - **Baja** lógica por código (marca un flag en el objeto).
  - **Búsqueda** por **código** (índice directo si lo mantienes ordenado + búsqueda binaria) o por **título** (búsqueda lineal).
  - **Actualización** de stock y disponibilidad por sucursal.
- Maneja correctamente **índices base 0** y controla `ArrayIndexOutOfBoundsException`.

3) Listas Enlazadas (Simple y Doble)

- **Préstamos activos**: usa **Lista Enlazada Simple** de nodos `Prestamo` (campos mínimos: `int codigoLibro`, `String usuario`, `LocalDate fecha`, `boolean devuelto`).
  - **Insertar al inicio** y/o **al final**.
  - **Eliminar** (marcar devuelto y remover nodo).
  - **Recorrido** e impresión de préstamos activos.
- **Historial de operaciones**: usa **Lista Doblemente Enlazada** para registrar cada operación con marca de tiempo y tipo (`ALTA`, `BAJA`, `PRESTAMO`, `DEVOLUCION`).
  - Permite **recorrer hacia adelante y hacia atrás**.
  - Agrega al historial cada vez que se ejecute una operación del sistema.

- Considera casos especiales: lista vacía, primer/último nodo, actualización correcta de referencias `siguiente/anterior`.

4) Programación Estructurada (flujo y modularidad)

- Organiza el programa en **métodos** claros (modularidad):
  - `cargarLibro()`, `eliminarLibro()`, `buscarLibroPorTitulo()`, `prestarLibro()`, `devolverLibro()`, `listarPrestamos()`, `listarHistorialAdelante()`, `listarHistorialAtras()` …
- Usa **estructuras de control**: `if/switch` para menús y validaciones; `for/while` para recorrer arrays/listas.
- Evita métodos muy largos: cada método con **una responsabilidad**.
- Añade un **menú de consola** con un **bucle principal** que termine solo cuando se elija “Salir”.

🔎 Reglas y validaciones mínimas

- **No** uses `ArrayList`, `LinkedList` u otras colecciones de Java para reemplazar lo pedido: **implementa tus propias listas enlazadas**.
- Controla **desbordes** y **bajos** de stock, índices fuera de rango y entradas inválidas.
- Verifica **duplicados** de `codigo` al dar de alta.
- En préstamos: solo permite prestar si **stock > 0**; al devolver, **incrementa stock**.
- Cada operación registrada debe **añadirse al historial doblemente enlazado**.

📦 Entregables

- Código fuente (`.java`) organizado y **compilable**.
- `README.md` con:
  - Descripción corta del sistema.
  - Razones de elección de cada estructura y **complejidad** de operaciones clave.
  - Cómo compilar/ejecutar.
- **Video explicativo** (5–10 min) donde el estudiante muestre el **código implementado**, explique las **decisiones de diseño** (por qué se usaron arrays, listas enlazadas, etc.) y demuestre la **funcionalidad del programa** en ejecución.

- Ambos entregables (**código fuente y video**) deben ser:
    i. Subidos al **fork del repositorio Git** asignado.
    ii. Adjuntados en la **plataforma académica** para su validación final.
- (Opcional) `data_demo.txt` con libros iniciales para cargar.