Unidad 2

Taller — Implementación de listas con inserción y eliminación

Propósito: que el estudiante implemente y comprenda a profundidad las operaciones de **inserción** y **eliminación** en listas (simples, dobles o circulares), analizando los costos computacionales y reforzando su dominio en la manipulación de nodos en memoria dinámica.

> **Escenario guía (ejemplo):** Una lista que almacene estudiantes registrados en un curso, permitiendo insertar un nuevo estudiante en cualquier posición, y eliminarlo al finalizar su proceso académico.

1) Entregable principal — Implementación de lista

> **Qué es:** un programa en Java (u otro lenguaje estructurado) que implemente una **lista enlazada** con operaciones de inserción y eliminación en distintas posiciones.

1.1 Operaciones mínimas a implementar

- Insertar nodo al inicio.
- Insertar nodo al final.
- Insertar nodo en una posición específica.
- Eliminar nodo del inicio.
- Eliminar nodo del final.
- Eliminar nodo en una posición específica.

1.2 Estructura de la clase `Nodo`

```
class Nodo {
    int dato;
    Nodo siguiente;

    Nodo(int d) {
        this.dato = d;
        this.siguiente = null;
    }
}
```

1.3 Ejemplo de inserción

```
public void insertarInicio(int valor) {
    Nodo nuevo = new Nodo(valor);
    nuevo.siguiente = cabeza;
    cabeza = nuevo;
}
```

2) Ejercicios propuestos

Ejercicio 1 — Lista básica

Implementa una lista simple que permita **insertar al final** y **mostrar todos los elementos**.

Ejercicio 2 — Inserción múltiple

Modifica la lista para permitir insertar en cualquier posición dada.

Ejercicio 3 — Eliminación controlada

Agrega la operación para eliminar un nodo en una posición específica. Incluye control de errores cuando la posición no exista.

Ejercicio 4 — Lista doblemente enlazada

Implementa una lista doble con punteros a anterior y siguiente. Prueba la eliminación en ambos extremos.

Ejercicio 5 — Lista circular

Convierte tu lista en circular y prueba la inserción y eliminación en medio de la lista.

Ejercicio 6 — Caso práctico aplicado

Simula una lista de espera para un consultorio médico donde los pacientes ingresan y salen de la cola en tiempo real.

> **Nota:** documenta tu código con comentarios que expliquen las decisiones de implementación y analiza el costo de cada operación en notación O().

3) Análisis de complejidad

Operación	Lista simple	Lista doble
Insertar inicio	O(1)	O(1)
Insertar fin	O(n)	O(1) si hay puntero a último
Insertar posición	O(n)	O(n)
Eliminar inicio	O(1)	O(1)
Eliminar fin	O(n)	O(1) si hay puntero a último
Eliminar posición	O(n)	O(n)

4) Instrucciones paso a paso para la entrega

1. Crea un proyecto en tu lenguaje de preferencia (Java recomendado).

2. Implementa la clase `Nodo` y la clase `Lista`.
3. Programa todas las operaciones descritas en la sección 1.1.
4. Resuelve los **6 ejercicios propuestos** en archivos separados.
5. Documenta en un archivo `README.md` el análisis de cada ejercicio.
6. Incluye una tabla de complejidad para cada operación.
7. Sube tu entrega a un repositorio Git con la estructura indicada.

Checklist de calidad:

- ☐ Código modular y documentado.
- ☐ Inserción y eliminación funcionan en todos los casos.
- ☐ Casos de error controlados (posiciones inválidas, lista vacía).
- ☐ Complejidad analizada y justificada.
- ☐ Evidencias de ejecución (capturas de pantalla).

5) Entregables (carpeta sugerida)

```
taller-listas/
├─ src/
│  ├─ Nodo.java
│  ├─ ListaSimple.java
│  ├─ ListaDoble.java
│  ├─ ListaCircular.java
│  ├─ Ejercicio1.java … Ejercicio6.java
├─ README.md              # Descripción general y análisis
├─ COMPLEJIDAD.md         # Tabla y explicación O()
└─ evidencias/            # Capturas de ejecución
```

Cada archivo debe incluir fecha, versión y autor.