Unidad 1

Tema 3 — Listas enlazadas simples y dobles

Este tema introduce las **listas enlazadas**, estructuras dinámicas que permiten almacenar y gestionar colecciones de datos sin requerir memoria contigua. Se estudian las diferencias entre listas simples y dobles, sus operaciones principales y ventajas frente a los arrays.

Descripción del tema
A diferencia de los arrays, las listas enlazadas permiten crecer y decrecer dinámicamente. Se componen de nodos que almacenan datos y referencias a otros nodos. Este apartado abarca las listas *simplemente enlazadas* y las *doblemente enlazadas*, fundamentales en implementaciones de colas, pilas y estructuras más complejas.

Desarrollo del tema:

3.1 Concepto de lista enlazada

- Colección dinámica de nodos conectados por referencias.
- No requiere bloques contiguos de memoria.
- Adecuada para inserciones y eliminaciones frecuentes.

3.2 Lista simplemente enlazada

Cada nodo tiene:

1. **dato** (valor almacenado).
2. **referencia al siguiente nodo**.

Ejemplo en Java:

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

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

class ListaSimple {
    Nodo cabeza;

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

    public void imprimir() {
        Nodo actual = cabeza;
        while (actual != null) {
            System.out.print(actual.dato + " -> ");
            actual = actual.siguiente;
        }
        System.out.println("null");
    }
}

// Uso:
ListaSimple lista = new ListaSimple();
lista.insertarInicio(3);
lista.insertarInicio(2);
lista.insertarInicio(1);
lista.imprimir(); // 1 -> 2 -> 3 -> null
```

3.3 Lista doblemente enlazada

Cada nodo tiene:

1. **dato**.
2. **referencia al nodo siguiente**.
3. **referencia al nodo anterior**.

Ejemplo en Java:

```
class NodoDoble {
    int dato;
    NodoDoble siguiente, anterior;

    public NodoDoble(int dato) {
        this.dato = dato;
        this.siguiente = null;
        this.anterior = null;
    }
}

class ListaDoble {
    NodoDoble cabeza, cola;

    public void insertarFinal(int valor) {
        NodoDoble nuevo = new NodoDoble(valor);
        if (cabeza == null) {
            cabeza = cola = nuevo;
        } else {
            cola.siguiente = nuevo;
            nuevo.anterior = cola;
            cola = nuevo;
        }
    }

    public void imprimirAdelante() {
        NodoDoble actual = cabeza;
        while (actual != null) {
            System.out.print(actual.dato + " <-> ");
            actual = actual.siguiente;
        }
        System.out.println("null");
    }

    public void imprimirAtras() {
        NodoDoble actual = cola;
        while (actual != null) {
            System.out.print(actual.dato + " <-> ");
            actual = actual.anterior;
        }
        System.out.println("null");
    }
}
```

```
// Uso:
ListaDoble lista = new ListaDoble();
lista.insertarFinal(10);
lista.insertarFinal(20);
lista.insertarFinal(30);
lista.imprimirAdelante(); // 10 <-> 20 <-> 30 <-> null
lista.imprimirAtras();    // 30 <-> 20 <-> 10 <-> null
```

3.4 Comparación entre lista simple y doble

Característica	Lista simple	Lista doble
Memoria por nodo	Menor (1 referencia)	Mayor (2 referencias)
Recorrido	Solo hacia adelante	Adelante y atrás
Inserción/eliminación	Más sencilla	Más flexible
Complejidad	Menor código	Mayor implementación

3.5 Errores comunes

> **Errores frecuentes**
> - Olvidar actualizar referencias al eliminar un nodo. - Pérdida de referencia a la cabeza → pérdida total de la lista. - No manejar el caso especial de lista vacía.

📚 Bibliografía breve

- Lafore, R. (2018). *Estructuras de datos y algoritmos en Java*. Pearson.
- Goodrich, M., Tamassia, R. (2017). *Data Structures and Algorithms in Java*. Wiley.
- Cormen, T. et al. (2022). *Introduction to Algorithms*. MIT Press.