Corte 2

Tema 1 — Pilas (Stack)

Este tema presenta la estructura de datos **Pila** (*Stack*), utilizada para manejar elementos con el principio **LIFO (Last In, First Out)**. Se estudian sus operaciones fundamentales, implementaciones y aplicaciones prácticas.

Descripción del tema
Una pila es una colección ordenada de elementos donde el último en entrar es el primero en salir. Se asemeja a una torre de platos: solo se puede acceder al elemento en la parte superior.

Desarrollo del tema:

1.1 Concepto de pila

- Modelo **LIFO**: último en entrar, primero en salir.
- Solo permite inserción y eliminación en un extremo: **la cima (top)**.

1.2 Operaciones principales

Operación	Descripción
`push(x)`	Inserta un elemento en la cima
`pop()`	Elimina y retorna el elemento en la cima
`peek()`	Consulta el elemento en la cima sin eliminarlo
`isEmpty()`	Verifica si la pila está vacía

1.3 Implementación con Arrays

```
class PilaArray {
    private int[] datos;
    private int tope;
    private int capacidad;

    public PilaArray(int capacidad) {
        this.capacidad = capacidad;
        datos = new int[capacidad];
        tope = -1;
    }

    public void push(int valor) {
        if (tope == capacidad - 1) {
            throw new RuntimeException("Desbordamiento: pila llena");
        }
        datos[++tope] = valor;
    }

    public int pop() {
        if (isEmpty()) {
            throw new RuntimeException("Subdesbordamiento: pila vacía");
        }
        return datos[tope--];
    }

    public int peek() {
        if (isEmpty()) throw new RuntimeException("Pila vacía");
        return datos[tope];
    }

    public boolean isEmpty() {
        return tope == -1;
    }
}

// Uso
PilaArray pila = new PilaArray(5);
pila.push(10);
pila.push(20);
```

```
System.out.println(pila.peek()); // 20
System.out.println(pila.pop());  // 20
```

1.4 Implementación con Listas Enlazadas

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

class PilaLista {
    private Nodo cima;

    public void push(int valor) {
        Nodo nuevo = new Nodo(valor);
        nuevo.siguiente = cima;
        cima = nuevo;
    }

    public int pop() {
        if (isEmpty()) throw new RuntimeException("Pila vacía");
        int valor = cima.dato;
        cima = cima.siguiente;
        return valor;
    }

    public int peek() {
        if (isEmpty()) throw new RuntimeException("Pila vacía");
        return cima.dato;
    }

    public boolean isEmpty() {
        return cima == null;
    }
}
```

1.5 Aplicaciones de las pilas

- Evaluación de expresiones aritméticas (infija, postfija, prefija).
- Control de llamadas a funciones (pila de ejecución).
- Deshacer/rehacer en editores de texto.
- Navegadores web (historial).

1.6 Errores comunes

> **Errores frecuentes**
> - Subdesbordamiento: intentar hacer `pop()` en pila vacía. - Desbordamiento: insertar en pila llena (en implementación con arrays). - No controlar excepciones → el programa se detiene abruptamente.

📚 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.