



```
// ===== IMPORTACIONES =====
```

```
// Estas son clases de la biblioteca estándar de Java (java.util)
```

```
import java.util.Stack;    // Implementación clásica de Pila (LIFO)
```

```
import java.util.Queue;    // Interfaz de Cola (FIFO)
```

```
import java.util.LinkedList; // Implementación común de Queue
```

```
import java.util.EmptyStackException; // Excepción que lanza Stack si haces pop/peek en vacío
```

```
// Clase pública: el nombre del archivo debe coincidir con esta clase
```

```
public class DemoPilasColas {
```

```
    // ===== CLASE DE MODELO PARA LA PILA =====
```

```
    // Representa una "operación" (ej: deshacer/undo en un editor)
```

```
    static class Operacion {
```

```
        private final int id;    // campo inmutable
```

```
        private final String descripcion;
```

```
    // Constructor: crea la instancia asignando sus campos
```

```
    Operacion(int id, String descripcion) {
```

```
        this.id = id;
```


```
        this.descripcion = descripcion;
```

```
    }
```

```
    // toString(): cómo se imprime el objeto al mostrarlo por consola
```

```
    @Override
```

```
    public String toString() {
```



```
        return "Operacion{id=" + id + ", desc=" + descripcion + "}";
    }
}
```

```
// ===== CLASE DE MODELO PARA LA COLA =====
```

```
// Representa un cliente esperando atención
```

```
static class Cliente {
```

```
    private final String nombre;
```

```
    private final int turno;
```

```
    Cliente(String nombre, int turno) {
```

```
        this.nombre = nombre;
```

```
        this.turno = turno;
```

```
    }
```

```
@Override
```

```
public String toString() {
```

```
    return "Cliente{turno=" + turno + ", nombre=" + nombre + "}";
```

```
}
```

```
}
```

```
// Método auxiliar para imprimir contenido de la pila sin modificarla
```

```
// (Stack hereda de Vector, por eso su toString muestra los elementos)
```

```
static void imprimirPila(Stack<Operacion> pila) {
```

```
    System.out.println("Pila -> " + pila);
```

```
}
```



```
// Método auxiliar para imprimir contenido de la cola sin modificarla
```

```
static void imprimirCola(Queue<Cliente> cola) {  
    System.out.println("Cola -> " + cola);  
}
```

```
// ===== MÉTODO PRINCIPAL =====
```

```
public static void main(String[] args) {  
    // ----- DEMO: PILA (Stack) -----  
    // Creamos una pila de objetos Operacion (tipo genérico <Operacion>)  
    Stack<Operacion> pila = new Stack<>();
```

```
    System.out.println("=== DEMOSTRACIÓN DE PILA (LIFO) ===");
```

```
    // push(): apila elementos en la "cima"
```

```
    pila.push(new Operacion(1, "Escribir título"));
```

```
    pila.push(new Operacion(2, "Aplicar negrita"));
```

```
    pila.push(new Operacion(3, "Insertar imagen"));
```

```
    imprimirPila(pila); // Muestra los 3 elementos en orden de inserción
```

```
    // peek(): mira la cima sin extraerla
```

```
    Operacion cima = pila.peek();
```

```
    System.out.println("peek() -> cima actual: " + cima);
```

```
    // pop(): extrae y devuelve el objeto en la cima
```

```
    Operacion ultima = pila.pop();
```

```
    System.out.println("pop() -> se deshace: " + ultima);
```



```
imprimirPila(pila);
```

```
// size() e isEmpty(): tamaño y si está vacía
```

```
System.out.println("size() -> " + pila.size());
```

```
System.out.println("isEmpty() -> " + pila.isEmpty());
```

```
// search(obj): devuelve posición 1-based desde la cima (o -1 si no está)
```

```
int posDesdeCima = pila.search(cima);
```

```
System.out.println("search(cima) -> posición desde cima = " + posDesdeCima);
```

```
// Ejemplo de manejo de error al vaciar de más
```

```
try {
```

```
    pila.pop(); // quita "Aplicar negrita"
```

```
    pila.pop(); // quita "Escribir título"
```

```
    System.out.println("Intentando hacer pop() en pila vacía...");
```

```
    pila.pop(); // aquí lanza EmptyStackException
```

```
} catch (EmptyStackException e) {
```

```
    System.out.println(";Error!: EmptyStackException -> no puedes hacer pop() si la pila  
está vacía");
```

```
}
```

```
// ----- DEMO: COLA (Queue) -----
```

```
// LinkedList implementa la interfaz Queue, perfecta para FIFO
```

```
Queue<Cliente> cola = new LinkedList<>();
```

```
System.out.println("\n=== DEMOSTRACIÓN DE COLA (FIFO) ===");
```



```
// offer(): intenta encolar, devuelve true/false (no lanza excepción si no cabe)
```

```
cola.offer(new Cliente("Ana", 1));
```

```
cola.offer(new Cliente("Luis", 2));
```

```
// add(): similar a offer, pero lanza excepción si la inserción falla
```

```
cola.add(new Cliente("María", 3));
```

```
imprimirCola(cola);
```

```
// peek(): mira el frente sin extraer
```

```
Cliente frente = cola.peek();
```

```
System.out.println("peek() -> frente actual: " + frente);
```

```
// poll(): extrae el frente o devuelve null si está vacía (no lanza excepción)
```

```
Cliente atendido = cola.poll();
```

```
System.out.println("poll() -> atendiendo a: " + atendido);
```

```
imprimirCola(cola);
```

```
// element(): como peek, pero lanza excepción si está vacía
```

```
Cliente sigue = cola.element();
```

```
System.out.println("element() -> siguiente en fila: " + sigue);
```

```
// remove(): como poll, pero lanza excepción si la cola está vacía
```

```
Cliente atendido2 = cola.remove();
```

```
System.out.println("remove() -> atendiendo a: " + atendido2);
```

```
imprimirCola(cola);
```

```
// size() e isEmpty()
```



```
System.out.println("size() -> " + cola.size());
```

```
System.out.println("isEmpty() -> " + cola.isEmpty());
```

```
// Limpieza completa
```

```
cola.clear(); // borra todos los elementos
```

```
System.out.println("Después de clear(), isEmpty() -> " + cola.isEmpty());
```

```
}
```

```
}
```



¿Qué hace cada cosa?

Librerías (import ...)

- `java.util.Stack`: clase de pila **LIFO**; métodos clave: `push`, `pop`, `peek`, `isEmpty`, `size`, `search`.
- `java.util.Queue`: **interfaz** que define el contrato de una cola **FIFO**; no se instancia directamente.
- `java.util.LinkedList`: clase que **implementa** `Queue`; inserciones y extracciones $O(1)$ en extremos.
- `java.util.EmptyStackException`: excepción lanzada por `Stack` si haces `pop/peek` cuando está vacía.

Todas pertenecen a la **biblioteca estándar** (no necesitas dependencias externas).

Clases de modelo

- `Operacion` y `Cliente` son **clases propias** (POJOs).
 - **Constructor**: inicializa campos.
 - **`toString()`**: define cómo se imprimen los objetos (útil para `System.out.println`).

Métodos auxiliares

- `imprimirPila` / `imprimirCola`: muestran el contenido **sin modificar** las estructuras (son de solo lectura).

Flujo de la demo (Pila)

1. `new Stack<Operacion>()`: crea una pila vacía (capacidad crece automáticamente).
2. `push(...)`: apila 3 operaciones (cima queda “Insertar imagen”).
3. `peek()`: mira la cima → no elimina.
4. `pop()`: saca y devuelve la cima → ahora la cima pasa a ser “Aplicar negrita”.
5. `size()` / `isEmpty()`: info de estado.
6. `search(obj)`: posición 1-based desde la cima (útil para localizar).
7. `try { pop() ... } catch (EmptyStackException)`: demostración de **manejo de errores**.



Flujo de la demo (Cola)

1. `Queue<Cliente> cola = new LinkedList<>();` cola vacía.
2. `offer(...)` / `add(...)`: encolan al **final** (frente queda “Ana”).
3. `peek()`: mira el frente (no elimina).
4. `poll()`: saca el frente o devuelve **null** si está vacía (seguro).
5. `element()`: como peek pero **lanza excepción** si está vacía.
6. `remove()`: como poll pero **lanza excepción** si está vacía.
7. `size()` / `isEmpty()` / `clear()`: estado y limpieza.

3) Diferencias finas de métodos

Estructura	Ver sin quitar	Quitar y devolver	¿Qué pasa si está vacía?
Stack	<code>peek()</code>	<code>pop()</code>	Lanza <code>EmptyStackException</code>
Queue	<code>peek()</code> o <code>element()</code>	<code>poll()</code> o <code>remove()</code>	<code>peek()</code> → null; <code>element()</code> → excepción ; <code>poll()</code> → null; <code>remove()</code> → excepción
Insertar en Queue	<code>offer(e)</code> vs <code>add(e)</code>	—	<code>offer</code> devuelve false si falla; <code>add</code> lanza excepción

En `LinkedList` normalmente ambas insertan sin problema (no hay capacidad fija), pero la diferencia es **semántica** y es buena práctica conocerla.

4) Complejidad (Big-O)

- **Pila (Stack)**
`push`, `pop`, `peek`, `isEmpty`, `size` → **O(1)** amortizado.
- **Cola (LinkedList)**
`offer/add` (al final), `poll/remove` (al frente), `peek/element` → **O(1)**.
Búsquedas arbitrarias (`contains`) → **O(n)**.