Especificación de la Pila

Definir la Interfaz: Crea una `interface` cuyo nombre Pila

```
public interface Pila<E> {

}
```

Métodos Públicos: La interfaz debe contener la **firma** (nombre, parámetros y tipo de retorno) de todas las operaciones que el usuario puede realizar con el TAD. Estos son los únicos métodos visibles.

```
void push(E elemento);
E pop();
E top();
boolean estaVacia();
```

Parámetros Genéricos (`<E>`): Para hacer la Pila **reutilizable** con diferentes tipos de datos (como una lista de `Integer` o una pila de `String`), usa genéricos.

Manejo de Excepciones: Especifica qué excepciones arrojarán los métodos en caso de fallos (ej. `pop()` en una pila vacía debe lanzar una excepción como `PilaVaciaException`).

Fase de Implementación (La Clase Concreta)

La implementación define **cómo** se llevan a cabo las operaciones, utilizando una estructura de datos concreta. Esto se hace con una **`class`** que implementa la interfaz.

Implementar la Interfaz: Crea una clase que utilice la palabra clave `implements` para cumplir con el "contrato" definido por la interfaz.

```
public class PilaArray<E> implements Pila<E> {

}
```

Encapsulamiento de la Representación: La **estructura de datos subyacente** (la que almacena los datos, ej. un `Array` o una `Lista Enlazada`) debe ser declarada como **`private`**. Esto garantiza que el usuario solo interactúe con el TAD a través de los métodos públicos, protegiendo los detalles internos.

```
private E[] elementos; // Estructura de datos interna (Array)
private int cima;      // Índice de la cima
```

Lógica del Constructor: Inicializa la estructura de datos interna y los atributos necesarios (ej. tamaño del array, o el primer nodo).

Implementar está vacía

```
public boolean estaVacia() {
    return cima == 0;
}
```

Implementar Push:

```
@Override
public void push(E elemento) {
    if (cima < elementos.length - 1) {
        cima++;
        elementos[cima] = elemento;
    } else {
        // Indicar que está Llena
    }
}
```

Implementar Peek

```
public E peek() {
    if (estaVacia()) {
       System.out.println("Pila Vacía");
    }
    System.out.println("Conociendo el último de la pila");
    return (E) elementos[cima -1];
 }
```

Ocultar Métodos Auxiliares: Si necesitas métodos internos de soporte, decláralos como **`private`** (ej. un método para "redimensionarArray").

---

3. Principios de POO en la Implementación del TAD ✨

Principio	Aplicación en el TAD	Java (Mecanismo)
Abstracción	Separar el *qué* (las operaciones del TAD) del *cómo* (la implementación).	**`Interface`**

Principio	Aplicación en el TAD	Java (Mecanismo)
Encapsulamiento	Ocultar la estructura de datos interna del TAD.	Uso de **`private`** para los atributos y la estructura interna.
Polimorfismo	Una misma interfaz (`Pila`) puede tener múltiples implementaciones (`PilaArray`, `PilaListaEnlazada`), permitiendo cambiar la implementación sin afectar al código del cliente.	**`implements`**