

	UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA	
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
Aprobación: 2022/03/01	Código: GUIA-PRLE-001	Página: 1

INFORME DE LABORATORIO

(formato estudiante)

INFORMACIÓN BÁSICA					
ASIGNATURA:	ESTRUCTURA DE DATOS Y ALGORITMOS				
TÍTULO DE LA PRÁCTICA:	PILAS Y COLAS				
NÚMERO DE PRÁCTICA:	05	AÑO LECTIVO:	2025 – A	NRO. SEMESTRE:	Tercero III
FECHA DE PRESENTACIÓN	07/06/2025	HORA DE PRESENTACIÓN	23:59:59		
INTEGRANTE (s): Aragón Carpio Fredy José				NOTA:	
DOCENTE(s): • Mg. Ing. Rene Alonso Nieto Valencia.					

SOLUCIÓN Y RESULTADOS
<p>1. SOLUCIÓN DE EJERCICIOS/PROBLEMAS</p> <p>https://github.com/DARSEN12/Laboratorio_EDA_E.git</p> <p>a. Problemas Desarrollados</p> <p>1. Ejercicio: Implementar una Pila utilizando una clase StackList y una clase nodo e ingresar los elementos 1, 2, 3, 4, 5, 6, 7 y 8. De acuerdo a la implementación del marco teórico utilizando clases y métodos genéricos.</p> <p>Referencia:</p> <p>https://docs.oracle.com/javase/7/docs/api/java/util/Stack.html</p> <pre>package ED1; public class ED1 { public static void main(String[] args) { StackList<Integer> pila = new StackList<>(); for (int i = 1; i <= 8; i++) { pila.push(i); } } }</pre>

```
System.out.println("Contenido de la pila:");
pila.mostrar();

System.out.println("Elemento en la cima: " + pila.peak());

System.out.println("Eliminar elementos:");
while (!pila.isEmpty()) {
    System.out.println("Elemento eliminado: " + pila.pop());
}
}
```

En la clase principal ED1, primero inicializamos la pila utilizando una instancia de la clase StackList<Integer>. Luego, insertamos los números del 1 al 8 en la pila utilizando el método push. Posteriormente, mostramos el valor que se encuentra en la cima de la pila, sin eliminarlo, usando el método peek, lo que nos permite ver el último número ingresado en la pila. Finalmente, mostramos todos los elementos de la pila en orden de eliminación (siguiendo el principio LIFO: Last In, First Out) mediante el método pop, el cual elimina y retorna los elementos uno por uno, comenzando por el último ingresado hasta el primero.

```
package ED1;
```

```
class Nodo<T> {
    T valor;
    Nodo<T> siguiente;

    public Nodo(T valor) {
        this.valor = valor;
        this.siguiente = null;
    }
}

class StackList<T> {
    private Nodo<T> cima;

    public StackList() {
        this.cima = null;
    }
}
```

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

public void push(T valor) {
    Nodo<T> nuevoNodo = new Nodo<>(valor);
    nuevoNodo.siguiente = cima;
    cima = nuevoNodo;
}

public T pop() {
    if (isEmpty()) {
        throw new RuntimeException("La pila está vacía.");
    }
    T valor = cima.valor;
    cima = cima.siguiente;
    return valor;
}

public T peek() {
    if (isEmpty()) {
        throw new RuntimeException("La pila está vacía.");
    }
    return cima.valor;
}

public void mostrar() {
    Nodo<T> actual = cima;
    while (actual != null) {
        System.out.print(actual.valor + " ");
        actual = actual.siguiente;
    }
    System.out.println();
}
}
```

Esta clase implementa una pila (StackList) utilizando una lista enlazada simple. La clase Nodo define un nodo genérico que almacena un valor y un puntero al siguiente nodo. La clase StackList gestiona la pila con operaciones básicas: push para agregar un elemento a la cima de la pila, pop para eliminar y retorna

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
Aprobación: 2022/03/01	Código: GUIA-PRLE-001	Página: 4

el elemento en la cima, peek para obtener el valor de la cima sin eliminarlo, y isEmpty para verificar si la pila está vacía. Además, tiene el método mostrar que imprime los elementos de la pila desde la cima hasta el final. La pila se implementa mediante la manipulación de los punteros cima y siguiente de los nodos.

```
PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 5> & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.7-hotspot\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Usuario\AppData\Roaming\Code\User\workspaceStorage\e5bf5cdb02fe313a58247499688b53a3\redhat.java\jdt_ws\Sesión 5_c6ea4efb\bin' 'ED1.EDI'
Contenido de la pila:
8 7 6 5 4 3 2 1
Elemento en la cima: 8
Eliminar elementos:
Elemento eliminado: 8
Elemento eliminado: 7
Elemento eliminado: 6
Elemento eliminado: 5
Elemento eliminado: 4
Elemento eliminado: 3
Elemento eliminado: 2
Elemento eliminado: 1
PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 5>
```

2. Ejercicio: Implementar una Cola utilizando una clase QueueList y una clase nodo e ingresar los elementos 1, 2, 3, 4, 5, 6, 7 y 8. De acuerdo a la implementación del marco teórico utilizando clases y métodos genéricos.

Referencia:

<https://docs.oracle.com/javase/7/docs/api/java/util/AbstractQueue.html>

```
package ED2;

public class ED2 {
    public static void main(String[] args) {
        QueueList<Integer> cola = new QueueList<>();

        for (int i = 1; i <= 8; i++) {
            cola.enqueue(i);
        }

        System.out.println("Contenido de la cola:");
        cola.mostrar();

        System.out.println("Elemento en el frente: " + cola.peek());

        System.out.println("Eliminar elementos:");
        while (!cola.isEmpty()) {
```

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 5</p>

```

                                System.out.println("Elemento eliminado: " +
cola.dequeue());
        }
    }
}

```

Este código crea una instancia de una cola (QueueList) que almacena enteros (Integer). Luego, agrega ocho elementos (del 1 al 8) a la cola utilizando el método enqueue. Después, muestra el contenido actual de la cola con el método mostrar. A continuación, imprime el elemento que está al frente de la cola usando peek. Finalmente, elimina los elementos de la cola uno por uno con el método dequeue y los imprime hasta que la cola esté vacía, utilizando un bucle while que verifica si la cola está vacía con el método isEmpty.

```

package ED2;

class Nodo<T> {
    T valor;
    Nodo<T> siguiente;

    public Nodo(T valor) {
        this.valor = valor;
        this.siguiente = null;
    }
}

class QueueList<T> {
    private Nodo<T> frente;
    private Nodo<T> finalCola;

    public QueueList() {
        this.frente = null;
        this.finalCola = null;
    }

    public boolean isEmpty() {
        return frente == null;
    }
}



```

```
public void enqueue(T valor) {
    Nodo<T> nuevoNodo = new Nodo<>(valor);
    if (finalCola == null) {
        frente = nuevoNodo;
        finalCola = nuevoNodo;
    } else {
        finalCola.siguiente = nuevoNodo;
        finalCola = nuevoNodo;
    }
}

public T dequeue() {
    if (isEmpty()) {
        throw new RuntimeException("La cola está vacía.");
    }
    T valor = frente.valor;
    frente = frente.siguiente;
    if (frente == null) {
        finalCola = null;
    }
    return valor;
}

public T peek() {
    if (isEmpty()) {
        throw new RuntimeException("La cola está vacía.");
    }
    return frente.valor;
}

public void mostrar() {
    Nodo<T> actual = frente;
    while (actual != null) {
        System.out.print(actual.valor + " ");
        actual = actual.siguiente;
    }
    System.out.println();
}
```

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
Aprobación: 2022/03/01	Código: GUIA-PRLE-001	Página: 7

```
}
```

Este código implementa una cola (QueueList) utilizando una lista enlazada simple. La clase Nodo define un nodo genérico que almacena un valor y un puntero al siguiente nodo. La clase QueueList gestiona la cola con operaciones básicas: enqueue para agregar un elemento al final de la cola, dequeue para eliminar y retorna el elemento en el frente de la cola, peek para obtener el valor en el frente sin eliminarlo, y isEmpty para verificar si la cola está vacía. La cola se maneja mediante los punteros frente y finalCola. El método mostrar imprime los elementos de la cola desde el frente hasta el final. Además, si la cola está vacía y se intenta hacer un dequeue o peek, se lanza una excepción.

```
PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 5> & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.7-hotspot\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Usuario\AppData\Roaming\Code\User\workspaceStorage\e5bf5cdb02fe313a58247499688b53a3\redhat.java\jdt_ws\Sesión_5_c6ea4efb\bin' 'ED2.ED2'
Contenido de la cola:
1 2 3 4 5 6 7 8
Elemento en el frente: 1
Eliminar elementos:
Elemento eliminado: 1
Elemento eliminado: 2
Elemento eliminado: 3
Elemento eliminado: 4
Elemento eliminado: 5
Elemento eliminado: 6
Elemento eliminado: 7
Elemento eliminado: 8
PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 5>
```

b. Problemas Propuestos

1. Implementar una Pila que tenga los elementos del 1 al 10, usando la clase nodo en java.

```
package EP1;

public class EP1 {
    public static void main(String[] args) {
        Pila<Integer> pila = new PilaImpl<>();

        // Apilar los elementos del 1 al 10
        for (int i = 1; i <= 10; i++) {
            pila.apilar(i);
            System.out.println("Elemento apilado: " + i);
        }

        // Desapilar los elementos y mostrarlos
        while (!pila.esVacia()) {
            System.out.println("Elemento desapilado: " +
pila.desapilar());
        }
    }
}
```

```
    }  
}  
}
```

```
package EP1;  
  
public class Nodo<T> {  
    T valor;  
    Nodo<T> siguiente;  
  
    Nodo(T valor) {  
        this.valor = valor;  
        this.siguiente = null;  
    }  
}
```

```
package EP1;  
  
public interface Pila<T> {  
    void apilar(T elemento);  
    T desapilar();  
    T cima();  
    boolean esVacia();  
}
```

```
package EP1;  
  
public class PilaImpl<T> implements Pila<T> {  
    private Nodo<T> cima;  
  
    public PilaImpl() {  
        cima = null;  
    }  
  
    @Override  
    public void apilar(T elemento) {
```



```
Nodo<T> nuevoNodo = new Nodo<>(elemento);
nuevoNodo.siguiente = cima;
cima = nuevoNodo;
}

@Override
public T desapilar() {
    if (esVacia()) {
        throw new RuntimeException("La pila está vacía");
    }
    T valor = cima.valor;
    cima = cima.siguiente;
    return valor;
}

@Override
public T cima() {
    if (esVacia()) {
        throw new RuntimeException("La pila está vacía");
    }
    return cima.valor;
}

@Override
public boolean esVacia() {
    return cima == null;
}
}
```

Explicación:

- Pila.java: Define la interfaz que cualquier pila debe cumplir.
- Nodo.java: Define la estructura básica del nodo que almacena un valor y un puntero al siguiente nodo.

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 10</p>

- PilaImpl.java: Implementa la interfaz Pila usando nodos. Mantiene una referencia a la cima de la pila y define cómo agregar (apilar), quitar (desapilar), y ver (cima) elementos.
- EP1.java: Es la clase principal que contiene el método main(), en el cual se crean instancias de la pila, se apilan los números del 1 al 10, y luego se desapilan y se muestran en orden inverso.

```
PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 5> c:: cd 'c:\Program Files\Eclipse Adoptium\jdk-17.0.7-hotspot\bin\java.exe' '-XX:+ShowCodeDuringGC' -Xmx1024m -Djava.class.path=. -Duser.dir=. -Duser.home=C:\Users\Usuario\workspaceStorage\e5bf5cdb02fe313a58247499688b53a3\redhat.java\jdt_ws\Sesión 5_
Elemento apilado: 1
Elemento apilado: 2
Elemento apilado: 3
Elemento apilado: 4
Elemento apilado: 5
Elemento apilado: 6
Elemento apilado: 7
Elemento apilado: 8
Elemento apilado: 9
Elemento apilado: 10
Elemento desapilado: 10
Elemento desapilado: 9
Elemento desapilado: 8
Elemento desapilado: 7
Elemento desapilado: 6
Elemento desapilado: 5
Elemento desapilado: 4
Elemento desapilado: 3
Elemento desapilado: 2
Elemento desapilado: 1
PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 5>
```

2. Implementar una Pila que tenga los elementos del 1 al 10, usando la clase nodo en java y los métodos vistos en el marco teórico (push, pop, top, destroyStack, isEmpty, isFull, printStack) y probar una clase Principal con un menú de opciones para probar los métodos.

```
package EP2;

import java.util.*;

public class ED2 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Crear la pila con un tamaño máximo de 10
        Pila<Integer> pila = new PilaImpl<>(10);
```

```
// Apilar los elementos del 1 al 10
for (int i = 1; i <= 10; i++) {
    pila.push(i);
}

int opcion;
do {
    System.out.println("\nMenú de Opciones:");
    System.out.println("1. Apilar un elemento");
    System.out.println("2. Desapilar un elemento");
    System.out.println("3. Ver el elemento en la cima");
    System.out.println("4. Destruir la pila");
    System.out.println("5. Verificar si la pila está vacía");
    System.out.println("6. Verificar si la pila está llena");
    System.out.println("7. Imprimir la pila");
    System.out.println("0. Salir");
    System.out.print("Seleccione una opción: ");
    opcion = scanner.nextInt();

    switch (opcion) {
        case 1:
            System.out.print("Ingrese el número a apilar: ");

            int numero = scanner.nextInt();
            pila.push(numero);
            break;
        case 2:
            Integer elementoDesapilado = pila.pop();
            if (elementoDesapilado != null) {
                System.out.println("Elemento desapilado: " + elementoDesapilado);
            }
            break;
        case 3:
            Integer cima = pila.top();
            if (cima != null) {
```

```
                System.out.println("Elemento en la cima: " +
cima);
            }
            break;
        case 4:
            pila.destroyStack();
            break;
        case 5:
            if (pila.isEmpty()) {
                System.out.println("La pila está vacía.");
            } else {
                System.out.println("La pila no está
vacía.");
            }
            break;
        case 6:
            if (pila.isFull()) {
                System.out.println("La pila está llena.");
            } else {
                System.out.println("La pila no está
llena.");
            }
            break;
        case 7:
            pila.printStack();
            break;
        case 0:
            System.out.println("Saliendo...");
            break;
        default:
            System.out.println("Opción no válida.");
            break;
    }
} while (opcion != 0);

scanner.close();
}
```

```
package EP2;

public class Nodo<T> {
    T valor;
    Nodo<T> siguiente;

    Nodo(T valor) {
        this.valor = valor;
        this.siguiente = null;
    }
}
```

```
package EP2;

public interface Pila<T> {
    void push(T elemento);
    T pop();
    T top();
    void destroyStack();
    boolean isEmpty();
    boolean isFull();
    void printStack();
}
```

```
package EP2;

public class PilaImpl<T> implements Pila<T> {
    private Nodo<T> cima;
    private int tamañoMaximo;
    private int tamañoActual;

    public PilaImpl(int tamañoMaximo) {
        cima = null;
        this.tamañoMaximo = tamañoMaximo;
        tamañoActual = 0;
    }
}
```

```
// Método para apilar un elemento
@Override
public void push(T elemento) {
    if (isFull()) {
        System.out.println("La pila está llena. No se puede
apilar el elemento.");
        return;
    }
    Nodo<T> nuevoNodo = new Nodo<>(elemento);
    nuevoNodo.siguiente = cima;
    cima = nuevoNodo;
    tamañoActual++;
}

// Método para desapilar un elemento
@Override
public T pop() {
    if (isEmpty()) {
        System.out.println("La pila está vacía. No se puede
desapilar.");
        return null;
    }
    T valor = cima.valor;
    cima = cima.siguiente;
    tamañoActual--;
    return valor;
}

// Método para obtener el elemento en la cima sin eliminarlo
@Override
public T top() {
    if (isEmpty()) {
        System.out.println("La pila está vacía. No hay elementos
en la cima.");
        return null;
    }
    return cima.valor;
}
```

```
// Método para destruir la pila
@Override
public void destroyStack() {
    cima = null;
    tamañoActual = 0;
    System.out.println("La pila ha sido destruida.");
}

// Método para verificar si la pila está vacía
@Override
public boolean isEmpty() {
    return cima == null;
}

// Método para verificar si la pila está llena
@Override
public boolean isFull() {
    return tamañoActual >= tamañoMaximo;
}

// Método para imprimir los elementos de la pila
@Override
public void printStack() {
    if (isEmpty()) {
        System.out.println("La pila está vacía.");
        return;
    }
    Nodo<T> actual = cima;
    System.out.print("Elementos en la pila: ");
    while (actual != null) {
        System.out.print(actual.valor + " ");
        actual = actual.siguiente;
    }
    System.out.println();
}
}
```

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
Aprobación: 2022/03/01	Código: GUIA-PRLE-001	Página: 16

Explicación:

- A. Pila.java: Define los métodos que debe implementar cualquier pila, como push, pop, top, destroyStack, isEmpty, isFull, y printStack.
- B. Nodo.java: Define un nodo genérico que almacena un valor y una referencia al siguiente nodo en la pila.
- C. PilaImpl.java: Implementa la interfaz Pila. Esta clase maneja la lógica de la pila:
 - a. push apila un nuevo elemento en la cima.
 - b. pop elimina y devuelve el elemento de la cima.
 - c. top devuelve el elemento de la cima sin eliminarlo.
 - d. destroyStack vacía la pila.
 - e. isEmpty verifica si la pila está vacía.
 - f. isFull verifica si la pila ha alcanzado su capacidad máxima.
 - g. printStack imprime todos los elementos en la pila.
- D. Principal.java: Clase que contiene un menú interactivo para probar los métodos de la pila. Permite al usuario:
 - a. Apilar un número.
 - b. Desapilar un número.
 - c. Ver el número en la cima.
 - d. Destruir la pila.
 - e. Comprobar si la pila está vacía o llena.
 - f. Imprimir todos los elementos de la pila.


```
PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Lab  
:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Usuario\Ap  
ción 5_c6ea4efb\bin' 'EP2.ED2'
```

Menú de Opciones:

1. Apilar un elemento
2. Desapilar un elemento
3. Ver el elemento en la cima
4. Destruir la pila
5. Verificar si la pila está vacía
6. Verificar si la pila está llena
7. Imprimir la pila
0. Salir

Seleccione una opción: 1

Ingrese el número a apilar: 5



La pila está llena. No se puede apilar el elemento.

Menú de Opciones:

1. Apilar un elemento
2. Desapilar un elemento
3. Ver el elemento en la cima
4. Destruir la pila
5. Verificar si la pila está vacía
6. Verificar si la pila está llena
7. Imprimir la pila
0. Salir

Seleccione una opción: 2

Elemento desapilado: 10

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
Aprobación: 2022/03/01	Código: GUIA-PRLE-001	Página: 18

Menú de Opciones:

1. Apilar un elemento
2. Desapilar un elemento
3. Ver el elemento en la cima
4. Destruir la pila
5. Verificar si la pila está vacía
6. Verificar si la pila está llena
7. Imprimir la pila
0. Salir

Seleccione una opción: 0

Saliendo...

PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio

3. Implementar una Cola que tenga los elementos del 1 al 10, usando la clase nodo en java.

```
package EP3;

import java.util.Scanner;

public class EP3 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Crear la cola con un tamaño máximo de 10
        Cola<Integer> cola = new ColaImpl<>(10);

        // Encolar los elementos del 1 al 10
        for (int i = 1; i <= 10; i++) {
            cola.enqueue(i);
        }

        int opcion;
        do {
            System.out.println("\nMenú de Opciones:");
            System.out.println("1. Agregar un elemento a la cola");
            System.out.println("2. Eliminar un elemento de la cola");
        } while (opcion != 0);
    }
}
```

```
        System.out.println("3. Ver el primer elemento de la cola");  
  
        System.out.println("4. Verificar si la cola está vacía");  
  
        System.out.println("5. Verificar si la cola está llena");  
  
        System.out.println("6. Imprimir la cola");  
        System.out.println("0. Salir");  
        System.out.print("Seleccione una opción: ");  
        opcion = scanner.nextInt();  
  
        switch (opcion) {  
            case 1:  
                System.out.print("Ingrese el número a agregar a la cola: ");  
  
                int numero = scanner.nextInt();  
                cola.enqueue(numero);  
                break;  
            case 2:  
                Integer elementoEliminado = cola.dequeue();  
                if (elementoEliminado != null) {  
                    System.out.println("Elemento eliminado: " + elementoEliminado);  
                }  
                break;  
            case 3:  
                Integer frente = cola.front();  
                if (frente != null) {  
                    System.out.println("Elemento en la parte frontal: " + frente);  
                }  
                break;  
            case 4:  
                if (cola.isEmpty()) {  
                    System.out.println("La cola está vacía.");  
                } else {  
                    System.out.println("La cola no está vacía.");  
                }  
            }  
        }  
    }  
}
```

```
        }
        break;
    case 5:
        if (cola.isFull()) {
            System.out.println("La cola está llena.");
        } else {
            System.out.println("La cola no está
llenena.");
        }
        break;
    case 6:
        cola.printQueue();
        break;
    case 0:
        System.out.println("Saliendo...");
        break;
    default:
        System.out.println("Opción no válida.");
        break;
    }
} while (opcion != 0);

scanner.close();
}
}
```

```
package EP3;

public interface Cola<T> {
    void enqueue(T elemento);
    T dequeue();
    T front();
    boolean isEmpty();
    boolean isFull();
    void printQueue();
}
```

```
package EP3;

public class ColaImpl<T> implements Cola<T> {
    private Nodo<T> frente;
    private Nodo<T> finalCola;
    private int tamañoMaximo;
    private int tamañoActual;

    public ColaImpl(int tamañoMaximo) {
        frente = null;
        finalCola = null;
        this.tamañoMaximo = tamañoMaximo;
        tamañoActual = 0;
    }

    // Método para agregar un elemento a la cola
    @Override
    public void enqueue(T elemento) {
        if (isFull()) {
            System.out.println("La cola está llena. No se puede
agregar el elemento.");
            return;
        }
        Nodo<T> nuevoNodo = new Nodo<>(elemento);
        if (finalCola == null) {
            frente = nuevoNodo;
        } else {
            finalCola.siguiente = nuevoNodo;
        }
        finalCola = nuevoNodo;
        tamañoActual++;
    }

    // Método para eliminar y devolver el primer elemento de la cola
    @Override
    public T dequeue() {
        if (isEmpty()) {
```

```
        System.out.println("La cola está vacía. No se puede
eliminar un elemento.");
        return null;
    }
    T valor = frente.valor;
    frente = frente.siguiente;
    if (frente == null) {
        finalCola = null; // La cola se vacía
    }
    tamañoActual--;
    return valor;
}

// Método para obtener el primer elemento sin eliminarlo
@Override
public T front() {
    if (isEmpty()) {
        System.out.println("La cola está vacía. No hay elementos
en la parte frontal.");
        return null;
    }
    return frente.valor;
}

// Método para verificar si la cola está vacía
@Override
public boolean isEmpty() {
    return frente == null;
}

// Método para verificar si la cola está llena
@Override
public boolean isFull() {
    return tamañoActual >= tamañoMaximo;
}

// Método para imprimir los elementos de la cola
@Override
```

```
public void printQueue() {  
    if (isEmpty()) {  
        System.out.println("La cola está vacía.");  
        return;  
    }  
    Nodo<T> actual = frente;  
    System.out.print("Elementos en la cola: ");  
    while (actual != null) {  
        System.out.print(actual.valor + " ");  
        actual = actual.siguiente;  
    }  
    System.out.println();  
}
```

```
package EP3;  
  
public class Nodo<T> {  
    T valor;  
    Nodo<T> siguiente;  
  
    Nodo(T valor) {  
        this.valor = valor;  
        this.siguiente = null;  
    }  
}
```

Explicación:

A. Cola.java: Define los métodos básicos que deben implementar las colas:

- a. enqueue: Agrega un nuevo elemento a la cola.
- b. dequeue: Elimina y devuelve el primer elemento de la cola.
- c. front: Devuelve el primer elemento sin eliminarlo.
- d. isEmpty: Verifica si la cola está vacía.
- e. isFull: Verifica si la cola está llena.
- f. printQueue: Imprime los elementos de la cola.

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 24</p>

- B. Nodo.java: Representa un nodo que contiene un valor y un puntero al siguiente nodo.
- C. ColaImpl.java: Implementa la interfaz Cola usando nodos para manejar los elementos de la cola:
- a. enqueue: Agrega un elemento al final de la cola.
 - b. dequeue: Elimina el primer elemento de la cola.
 - c. front: Devuelve el primer elemento sin eliminarlo.
 - d. isEmpty: Verifica si la cola está vacía.
 - e. isFull: Verifica si la cola está llena.
 - f. printQueue: Imprime todos los elementos de la cola.
- D. EP3.java: Clase principal que proporciona un menú interactivo para realizar las operaciones sobre la cola. Permite al usuario agregar, eliminar, ver el primer elemento, verificar si la cola está vacía o llena, y imprimir los elementos de la cola.


```
PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 5>
.\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Usuario\AppData
88b53a3\redhat.java\jdt_ws\Sesión 5_c6ea4efb\bin' 'EP3.EP3'
```

Menú de Opciones:

1. Agregar un elemento a la cola
2. Eliminar un elemento de la cola
3. Ver el primer elemento de la cola
4. Verificar si la cola está vacía
5. Verificar si la cola está llena
6. Imprimir la cola
0. Salir

Seleccione una opción: 2

Elemento eliminado: 1

Menú de Opciones:



1. Agregar un elemento a la cola
2. Eliminar un elemento de la cola
3. Ver el primer elemento de la cola
4. Verificar si la cola está vacía
5. Verificar si la cola está llena
6. Imprimir la cola
0. Salir

Seleccione una opción: 1

Ingrese el número a agregar a la cola: 4

Menú de Opciones:

1. Agregar un elemento a la cola
2. Eliminar un elemento de la cola
3. Ver el primer elemento de la cola
4. Verificar si la cola está vacía
5. Verificar si la cola está llena
6. Imprimir la cola

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 26</p>

```

0. Salir
Seleccione una opción: 6
Elementos en la cola: 2 3 4 5 6 7 8 9 10 4

Menú de Opciones:
1. Agregar un elemento a la cola
2. Eliminar un elemento de la cola
3. Ver el primer elemento de la cola
4. Verificar si la cola está vacía
5. Verificar si la cola está llena
6. Imprimir la cola
0. Salir
Seleccione una opción: 0
Saliendo...
PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 5>

```

4. Implementar una Cola que tenga los elementos del 1 al 10, usando la clase nodo en java y los métodos vistos en el marco teórico (encolar, desencolar, destroyQueue, isEmpty, isFull, front, back, printQueue) y probar una clase Principal con un menú de opciones para probar los métodos.

```

package EP4;

import java.util.Scanner;

public class EP4 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Crear la cola con un tamaño máximo de 10
        Cola<Integer> cola = new ColaImpl<>(10);

        // Encolar los elementos del 1 al 10
        for (int i = 1; i <= 10; i++) {
            cola.encolar(i);
        }

        int opcion;
        do {
            System.out.println("\nMenú de Opciones:");
            System.out.println("1. Agregar un elemento a la cola");

```

```
        System.out.println("2. Eliminar un elemento de la
cola");

        System.out.println("3. Ver el primer elemento de la
cola");

        System.out.println("4. Ver el último elemento de la
cola");

        System.out.println("5. Verificar si la cola está
vacía");

        System.out.println("6. Verificar si la cola está
llena");

        System.out.println("7. Imprimir la cola");
        System.out.println("8. Destruir la cola");
        System.out.println("0. Salir");
        System.out.print("Seleccione una opción: ");
        opcion = scanner.nextInt();

        switch (opcion) {
            case 1:
                System.out.print("Ingrese el número a agregar a
la cola: ");

                int numero = scanner.nextInt();
                cola.encolar(numero);
                break;
            case 2:
                Integer elementoEliminado = cola.desencolar();
                if (elementoEliminado != null) {
                    System.out.println("Elemento eliminado: " +
elementoEliminado);
                }
                break;
            case 3:
                Integer primerElemento = cola.front();
                if (primerElemento != null) {
                    System.out.println("Elemento en la parte
frontal: " + primerElemento);
                }
                break;
            case 4:
```

```
        Integer ultimoElemento = cola.back();
        if (ultimoElemento != null) {
            System.out.println("Elemento en la parte
final: " + ultimoElemento);
        }
        break;
    case 5:
        if (cola.isEmpty()) {
            System.out.println("La cola está vacía.");
        } else {
            System.out.println("La cola no está
vacía.");
        }
        break;
    case 6:
        if (cola.isFull()) {
            System.out.println("La cola está llena.");
        } else {
            System.out.println("La cola no está
llena.");
        }
        break;
    case 7:
        cola.printQueue();
        break;
    case 8:
        cola.destroyQueue();
        break;
    case 0:
        System.out.println("Saliendo...");
        break;
    default:
        System.out.println("Opción no válida.");
        break;
    }
} while (opcion != 0);

scanner.close();
```

```
}  
}  
  
package EP4;  
  
public class Nodo<T> {  
    T valor;  
    Nodo<T> siguiente;  
  
    Nodo(T valor) {  
        this.valor = valor;  
        this.siguiente = null;  
    }  
}
```

```
package EP4;  
  
public interface Cola<T> {  
    void encolar(T elemento);  
    T desencolar();  
    void destroyQueue();  
    boolean isEmpty();  
    boolean isFull();  
    T front();  
    T back();  
    void printQueue();  
}
```

```
package EP4;  
  
public class ColaImpl<T> implements Cola<T> {  
    private Nodo<T> frente;  
    private Nodo<T> finalCola;  
    private int tamañoMaximo;  
    private int tamañoActual;  
  
    public ColaImpl(int tamañoMaximo) {  
        frente = null;
```

```
finalCola = null;
this.tamañoMaximo = tamañoMaximo;
tamañoActual = 0;
}

// Método para agregar un elemento a la cola
@Override
public void encolar(T elemento) {
    if (isFull()) {
        System.out.println("La cola está llena. No se puede
agregar el elemento.");
        return;
    }
    Nodo<T> nuevoNodo = new Nodo<>(elemento);
    if (finalCola == null) {
        frente = nuevoNodo;
    } else {
        finalCola.siguiente = nuevoNodo;
    }
    finalCola = nuevoNodo;
    tamañoActual++;
}

// Método para eliminar y devolver el primer elemento de la cola
@Override
public T desencolar() {
    if (isEmpty()) {
        System.out.println("La cola está vacía. No se puede
eliminar un elemento.");
        return null;
    }
    T valor = frente.valor;
    frente = frente.siguiente;
    if (frente == null) {
        finalCola = null; // La cola se vacía
    }
    tamañoActual--;
    return valor;
}
```

```
}

// Método para destruir la cola
@Override
public void destroyQueue() {
    frente = null;
    finalCola = null;
    tamañoActual = 0;
    System.out.println("La cola ha sido destruida.");
}

// Método para verificar si la cola está vacía
@Override
public boolean isEmpty() {
    return frente == null;
}

// Método para verificar si la cola está llena
@Override
public boolean isFull() {
    return tamañoActual >= tamañoMaximo;
}

// Método para obtener el primer elemento sin eliminarlo
@Override
public T front() {
    if (isEmpty()) {
        System.out.println("La cola está vacía. No hay elementos en la parte frontal.");
        return null;
    }
    return frente.valor;
}

// Método para obtener el último elemento sin eliminarlo
@Override
public T back() {
    if (isEmpty()) {
```

```
        System.out.println("La cola está vacía. No hay elementos  
en la parte final.");  
        return null;  
    }  
    return finalCola.valor;  
}  
  
// Método para imprimir los elementos de la cola  
@Override  
public void printQueue() {  
    if (isEmpty()) {  
        System.out.println("La cola está vacía.");  
        return;  
    }  
    Nodo<T> actual = frente;  
    System.out.print("Elementos en la cola: ");  
    while (actual != null) {  
        System.out.print(actual.valor + " ");  
        actual = actual.siguiente;  
    }  
    System.out.println();  
}
```

Explicación:

Cola.java: Define las operaciones que deben ser implementadas para una cola:

encolar: Agrega un nuevo elemento al final de la cola.

desencolar: Elimina y devuelve el primer elemento de la cola.

destroyQueue: Destruye la cola (vacía todos los elementos).

isEmpty: Verifica si la cola está vacía.

isFull: Verifica si la cola está llena.

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 33</p>

front: Devuelve el primer elemento de la cola sin eliminarlo.

back: Devuelve el último elemento de la cola sin eliminarlo.

printQueue: Imprime los elementos de la cola.

Nodo.java: Representa un nodo en la cola, que contiene el valor y una referencia al siguiente nodo.

ColaImpl.java: Implementa la interfaz Cola. Mantiene referencias al primer (frente) y último (finalCola) nodo de la cola. Además, maneja la lógica para agregar, eliminar, verificar si está vacía o llena, y obtener el primer y último elemento de la cola.

EP4.java: Contiene un menú interactivo para probar las operaciones de la cola. Permite al usuario:

Agregar un elemento.

Eliminar un elemento.

Ver el primer y último elemento de la cola.

Verificar si la cola está vacía o llena.

Imprimir todos los elementos de la cola.

Destruir la cola.

```
PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 5>
.\java.exe' -XX:+ShowCodeDetailsInExceptionMessages' -cp' 'C:\Users\Usuario\AppData
88b53a3\redhat.java\jdt_ws\Sesión 5_c6ea4efb\bin' 'EP4.EP4'
```

Menú de Opciones:

1. Agregar un elemento a la cola
2. Eliminar un elemento de la cola
3. Ver el primer elemento de la cola
4. Ver el último elemento de la cola
5. Verificar si la cola está vacía
6. Verificar si la cola está llena
7. Imprimir la cola
8. Destruir la cola
0. Salir

Seleccione una opción: 7

Elementos en la cola: 1 2 3 4 5 6 7 8 9 10

Menú de Opciones:

1. Agregar un elemento a la cola
2. Eliminar un elemento de la cola
3. Ver el primer elemento de la cola
4. Ver el último elemento de la cola
5. Verificar si la cola está vacía
6. Verificar si la cola está llena
7. Imprimir la cola
8. Destruir la cola
0. Salir

Seleccione una opción: 8

La cola ha sido destruida.

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 35</p>

Menú de Opciones:

1. Agregar un elemento a la cola
2. Eliminar un elemento de la cola
3. Ver el primer elemento de la cola
4. Ver el último elemento de la cola
5. Verificar si la cola está vacía
6. Verificar si la cola está llena
7. Imprimir la cola
8. Destruir la cola
0. Salir

Seleccione una opción: 7

La cola está vacía.

Menú de Opciones:

1. Agregar un elemento a la cola
2. Eliminar un elemento de la cola
3. Ver el primer elemento de la cola
4. Ver el último elemento de la cola
5. Verificar si la cola está vacía
6. Verificar si la cola está llena
7. Imprimir la cola
8. Destruir la cola
0. Salir

Seleccione una opción: 0

Saliendo...

PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 5>

2. SOLUCIÓN DEL CUESTIONARIO

¿Cuáles fueron las dificultades que encontraste al desarrollar los ejercicios propuestos?

Las principales dificultades fueron manejar correctamente las referencias en las estructuras de datos como pilas y colas, especialmente al realizar operaciones de agregar o eliminar elementos. Además, el uso de generics y las interfaces puede ser confuso al principio si no se tiene experiencia previa con estos conceptos. La falta de ejemplos prácticos también complicó un poco la implementación y la depuración de las operaciones.

¿Es posible reutilizar la clase nodo para otras estructuras de datos, además de listas enlazadas, pilas y colas?

Sí, la clase Nodo puede reutilizarse en otras estructuras de datos como listas dobles enlazadas, árboles o grafos. Cualquier estructura de datos que requiera almacenar elementos de manera secuencial o jerárquica puede aprovechar los nodos, ya que se encargan de almacenar un valor y mantener una referencia al siguiente nodo.

¿Qué tipo de dato es NULL en Java?

En Java, null no es un tipo de dato, sino un valor especial que puede ser asignado a cualquier variable

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 36</p>

de tipo objeto para indicar que no apunta a ningún objeto o valor. Es usado para representar la ausencia de un valor, pero no puede ser utilizado con tipos primitivos como int o char.

¿Cuáles son los beneficios de utilizar tipos genéricos en las pilas y colas?

Usar tipos genéricos en pilas y colas permite que la estructura de datos sea más flexible, ya que puede almacenar cualquier tipo de dato sin necesidad de escribir múltiples versiones de la misma clase. Además, mejora la seguridad de tipos y evita errores de casting, haciendo el código más limpio y fácil de mantener.

3. CONCLUSIONES

En conclusión, el uso de estructuras de datos como pilas y colas en Java, junto con la implementación de nodos y tipos genéricos, ofrece flexibilidad y seguridad de tipos, permitiendo reutilizar el código para diferentes tipos de datos. A pesar de las dificultades iniciales al trabajar con generics y las referencias en los nodos, una vez comprendidos, estos conceptos facilitan la creación de soluciones más limpias y eficientes. Además, la reutilización de la clase Nodo en otras estructuras de datos aumenta la versatilidad del código, mientras que el valor null en Java juega un papel importante en representar la ausencia de datos.

RETROALIMENTACIÓN GENERAL

--

REFERENCIAS Y BIBLIOGRAFÍA

--