

	<p 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-PRLD-001	Página: 1

GUÍA DE LABORATORIO

GUÍA DE LABORATORIO

(formato docente)

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
TIPO DE PRÁCTICA:	INDIVIDUAL	Quispe Pauccar, Josué Claudio			
	GRUPAL	—	MÁXIMO DE ESTUDIANTES	01	
FECHA INICIO:	02/06/2025	FECHA FIN:	06/06/2025	DURACIÓN:	90 minutos.
RECURSOS A UTILIZAR: <ul style="list-style-type: none"> • REPOSITORIO GITHUB: https://github.com/JosueClaudioQP/EDA-Lab • Lenguaje de Programación Java. • Ide Java Eclipse/Visual Studio Code. 					
DOCENTE(s): <ul style="list-style-type: none"> • Mg. Ing. Rene Alonso Nieto Valencia. 					

OBJETIVOS/TEMAS Y COMPETENCIAS	
OBJETIVOS: <ul style="list-style-type: none"> • Aprenda Pilas y Colas. • Aplicar conceptos elementales de programación a resolver utilizando POO en problemas de algoritmos. • Desarrollar pruebas. 	
TEMAS: <ul style="list-style-type: none"> • Introducción. • Pilas y operaciones. • Colas y operaciones. 	
COMPETENCIAS	C.a
	C.b
	C.c
	C.d

 INGENIERIA SISTEMAS	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-PRLD-001	Página: 2

CONTENIDO DE LA GUÍA

II. EJERCICIO/PROBLEMA RESUELTO POR EL DOCENTE

En un editor Java, realizar la integración de los siguientes ejercicios, revisar y mostrar los resultados obtenidos y realizar una explicación del funcionamiento de forma concreta y clara.

1. **Ejercicio 1:** 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.

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

Para la resolución de este ejercicio se usaron 4 clases, primero creamos la clase que siempre usaremos, Node.java:

```
1 package Ejercicio1;
2
3 public class Nodo<T> {
4     T dato;
5     Nodo<T> next;
6
7     public Nodo(T dato){
8         this.dato = dato;
9         this.next = null;
10    }
11 }
12
```

Esta clase nos ayudará a manejar de manera correcta nuestras listas genéricas.

Luego se implementa la interfaz genérica Pila.java:

```
1 package Ejercicio1;
2
3 public interface Pila<T> {
4     void push(T elem);
5     T pop();
6     void top();
7     void destroyStack();
8     boolean isEmpty();
9     boolean isFull();
10 }
11
```

Esta interfaz será usada en la clase principal StackList.java, donde se implementarán todos los métodos necesarios para el funcionamiento de nuestra lista.

ProblemasResueltos > Ejercicio1 > StackList.java > {} Ejercicio1

```
1 package Ejercicio1;
2
3 public class StackList<T> implements Pila<T>{
4     private Nodo<T> cima;
5
6     public StackList(){
7         cima = null;
8     }
9
10    public void push(T dato) {
11        Nodo<T> nuevoNodo = new Nodo<>(dato);
12        nuevoNodo.next = cima;
13        cima = nuevoNodo;
14    }
15
16    public T pop(){
17        if (isEmpty()) {
18            System.out.println(x:"La pila está vacía, no se puede hacer pop.");
19            return null;
20        }
21        T dato = cima.dato;
22        cima = cima.next;
23        return dato;
24    }
25
26    public void top(){
27        if (isEmpty()) {
28            System.out.println(x:"La pila está vacía, no se puede hacer pop.");
29        } else {
30            System.out.println("Cima: " + cima.dato);
31        }
32    }
33
34    public void destroyStack() {
35        cima = null;
36        System.out.println(x:"Pila destruida (vacía).");
37    }
38
39    public boolean isEmpty() {
40        return cima == null;
41    }
42
```

```
43    public boolean isFull() {
44        return false;
45    }
46
47    public void mostrar() {
48        if (isEmpty()) {
49            System.out.println(x:"La pila está vacía.");
50            return;
51        }
52        Nodo<T> actual = cima;
53        System.out.print(s:"Pila: ");
54        while (actual != null) {
55            System.out.print(actual.dato + " ");
56            actual = actual.next;
57        }
58        System.out.println();
59    }
60 }
61
```

Se usa la clase genérica Node para guardar los datos y el enlace al siguiente nodo el cual permanece vacío. La clase genérica Stack contiene una referencia llamada "cima", el cual al hacer push crea un nuevo nodo con el dato, hace que el nuevo nodo apunte al nodo que estaba en la cima y actualiza la cima con el nuevo nodo. Al hacer pop se realiza una verificación si la pila está vacía, si no es así, guarda el dato actual de la cima, mueve la cima del siguiente nodo y devuelve el dato eliminado. El método top muestra el dato que está en la cima sin eliminarlo. El método destroyStack borra todos los elementos de la pila. isEmpty verifica si la pila está vacía. isFull retorna vacío ya que al ser listas no tienen un límite (solo el espacio de memoria).

Finalmente, el método mostrar imprime todos los elementos desde la cima hacia abajo.

Estos métodos serán usados también en los problemas propuestos.

RESULTADOS:

```
1 package Ejercicio1;
2
3 public class Main {
4     Run | Debug
5     public static void main(String[] args) {
6         StackList<Object> pila = new StackList<>();
7         for (int i = 1; i <= 8; i++) {
8             pila.push(i);
9         }
10        pila.mostrar();
11        pila.push(dato:9);
12        pila.mostrar();
13        pila.top();
14        System.out.println("Pop: " + pila.pop());
15        pila.top();
16        System.out.println("¿Está vacía? " + pila.isEmpty());
17        System.out.println("¿Está llena? " + pila.isFull());
18        pila.destroyStack();
19        System.out.println("¿Está vacía? " + pila.isEmpty());
20        pila.mostrar();
21    }
```


```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab\Laboratorio5>
ptionMessages' '-cp' 'C:\Users\JOSUE\AppData\Roaming\Code\User\workspace
Pila: 8 7 6 5 4 3 2 1
Pila: 9 8 7 6 5 4 3 2 1
Cima: 9
Pop: 9
Cima: 8
¿Está vacía? false
¿Está llena? false
Pila destruida (vacía).
¿Está vacía? true
La pila está vacía.
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab\Laboratorio5>
```

2. **Ejercicio 2:** 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>

Para la resolución de este ejercicio se usaron 4 clases, primero creamos la clase que siempre usaremos, Node.java:


```

ProblemasResueltos > Ejercicio2 >  Nodo.java > ...
1  package Ejercicio2;
2
3  public class Nodo<T> {
4      T dato;
5      Nodo<T> next;
6
7      public Nodo(T dato){
8          this.dato = dato;
9          this.next = null;
10     }
11
12 }

```

Se implementa la interfaz genérica Cola.java:

```

ProblemasResueltos > Ejercicio2 >  Cola.java > ...
1  package Ejercicio2;
2
3  public interface Cola<T> {
4      T enqueue(T elem);
5      void dequeue();
6      void destroyQueue();
7      boolean isEmpty();
8      boolean isFull();
9      void front();
10     void back();
11 }
12

```

Esta interfaz será usada en la clase principal QueueList.java, donde se implementarán todos los métodos necesarios para el funcionamiento de nuestra lista.

```

ProblemasResueltos > Ejercicio2 >  QueueList.java > ...
1  package Ejercicio2;
2
3  public class QueueList<T> implements Cola<T>{
4      private Nodo<T> frente;
5      private Nodo<T> finCola;
6      private int tamaño;
7      private int capacidad;
8
9      public QueueList() {
10         this.frente = null;
11         this.finCola = null;
12         this.tamaño = 0;
13         this.capacidad = Integer.MAX_VALUE;
14     }
15
16     public QueueList(int capacidad){
17         this();
18         this.capacidad = capacidad;
19     }
20
21     public T enqueue(T elem) {
22         if (isFull()) {
23             System.out.println(x:"La cola está llena, no se puede encolar.");
24             return null;
25         }
26         Nodo<T> nuevoNodo = new Nodo<>(elem);
27         if (isEmpty()) {
28             frente = finCola = nuevoNodo;
29         } else {
30             finCola.next = nuevoNodo;
31             finCola = nuevoNodo;
32         }
33         tamaño++;
34         return elem;
35     }

```

```

37     public void dequeue() {
38         if (isEmpty()) {
39             System.out.println(x:"La cola está vacía, no se puede desencolar.");
40             return;
41         }
42         frente = frente.next;
43         tamaño--;
44         if (frente == null) {
45             finCola = null;
46         }
47     }
48
49     public void front() {
50         if (isEmpty()) {
51             System.out.println(x:"La cola está vacía, no hay frente.");
52         } else {
53             System.out.println("Frente: " + frente.dato);
54         }
55     }
56
57     public void back() {
58         if (isEmpty()) {
59             System.out.println(x:"La cola está vacía, no hay final.");
60         } else {
61             System.out.println("Final: " + finCola.dato);
62         }
63     }
64
65     public void destroyQueue() {
66         frente = null;
67         finCola = null;
68         tamaño = 0;
69         System.out.println(x:"Cola destruida (vacía).");
70     }
71
72     public boolean isEmpty() {
73         return frente == null;
74     }
75
76     public boolean isFull() {
77         return tamaño >= capacidad;
78     }
79
80     public void mostrar() {
81         if (isEmpty()) {
82             System.out.println(x:"La cola está vacía.");
83             return;
84         }
85         Nodo<T> actual = frente;
86         System.out.print(s:"Cola: ");
87         while (actual != null) {
88             System.out.print(actual.dato + " ");
89             actual = actual.next;
90         }
91         System.out.println();
92     }
93 }
94

```

La clase genérica QueueList implementa una cola utilizando una lista enlazada, donde cada nodo contiene un dato de tipo T y un puntero al siguiente nodo. La cola mantiene referencias al primer elemento (frente) y al último (finCola), así como su tamaño actual y una capacidad máxima (por defecto ilimitada). El método enqueue agrega elementos al final, mientras que dequeue elimina el elemento del frente. También incluye métodos para ver el elemento al frente (front) y al final (back), verificar si la cola está vacía o llena, destruirla (destroyQueue), e imprimir su contenido (mostrar).

RESULTADOS:

```

ProblemasResueltos > Ejercicio2 > Main.java > Main > main(String[])
1  package Ejercicio2;
2
3  public class Main {
4      Run | Debug
    public static void main(String[] args) {
5          QueueList<Integer> cola = new QueueList<>(capacidad:8);
6
7          for (int i = 1; i <= 8; i++) {
8              cola.enqueue(i);
9          }
10
11         cola.mostrar();
12         System.out.println("¿Está llena? " + cola.isFull());
13
14         cola.front();
15         cola.back();
16
17         cola.enqueue(elem:9);
18         cola.mostrar();
19
20         cola.dequeue();
21         cola.dequeue();
22
23         cola.mostrar();
24         System.out.println("¿Está llena? " + cola.isFull());
25
26         System.out.println("¿Está vacía? " + cola.isEmpty());
27
28         cola.destroyQueue();
29         cola.mostrar();
30
31     }
32 }
33

```

```

PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab\Laboratorio5>
ptionMessages' '-cp' 'C:\Users\JOSUE\AppData\Roaming\Code\User\workspace'
Cola: 1 2 3 4 5 6 7 8
¿Está llena? true
Frente: 1
Final: 8
La cola está llena, no se puede encolar.
Cola: 1 2 3 4 5 6 7 8
Cola: 3 4 5 6 7 8
¿Está llena? false
¿Está vacía? false
Cola destruida (vacía).
La cola está vacía.
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab\Laboratorio5>

```

III. EJERCICIOS/PROBLEMAS PROPUESTOS

De acuerdo a los ejercicios propuestos desarrollar los algoritmos y mostrar las siguientes indicaciones:

- Enunciado del ejercicio.
- Código en java desarrollado.
- Resultados obtenidos.
- Explicación breve y concreta del código implementado.

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

Usando la clase genérica: `public class Stack<E>` usar métodos genéricos `public E Metodo()`

Antes de empezar, cada ejercicio consta de una clase **Nodo**, la cual es igual en todos los ejercicios:

```
3 public class Nodo<E> {
4     E dato;
5     Nodo<E> siguiente;
6
7     public Nodo(E dato) {
8         this.dato = dato;
9         this.siguiente = null;
10    }
11 }
```

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

```
ProblemasPropuestos > Ejercicio1 > Stack.java > Stack<E> > printStack()
1 package ProblemasPropuestos.Ejercicio1;
2
3 public class Stack<E> {
4     private Nodo<E> cima;
5
6     public Stack(){
7         cima = null;
8     }
9
10    public E push(E dato) {
11        Nodo<E> nuevo = new Nodo<>(dato);
12        nuevo.siguiente = cima;
13        cima = nuevo;
14        return dato;
15    }
16
17    public void printStack() {
18        Nodo<E> actual = cima;
19        System.out.print(s:"Pila: ");
20        while (actual != null) {
21            System.out.print(actual.dato + " ");
22            actual = actual.siguiente;
23        }
24        System.out.println();
25    }
26 }
```

Se usa la clase genérica **Nodo** para guardar el dato y el enlace al siguiente nodo. La clase **stack** mantiene una referencia "cima", la cual al hacer **push**, se apila de nuevo sobre el anterior.

RESULTADOS:

```
ProblemasPropuestos > Ejercicio1 > Test.java > ...
1 package ProblemasPropuestos.Ejercicio1;
2
3 public class Test {
4     public static void main(String[] args) {
5         Stack<Integer> pila = new Stack<>();
6
7         for (int i = 1; i <= 10; i++) {
8             pila.push(i);
9         }
10
11        pila.printStack();
12    }
13 }
```

```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab\Laboratorio5>
ptionMessages' '-cp' 'C:\Users\JOSUE\AppData\Roaming\Code\User\workspace
Pila: 10 9 8 7 6 5 4 3 2 1
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab\Laboratorio5>
```

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.

```

ProblemasPropuestos > Ejercicio2 > Stack.java > Stack<E> > Stack(int)
1  package ProblemasPropuestos.Ejercicio2;
2
3  public class Stack<E> {
4      private Nodo<E> cima;
5      private int tamaño;
6      private final int capacidad;
7
8      public Stack() {
9          this.capacidad = Integer.MAX_VALUE;
10         this.cima = null;
11         this.tamaño = 0;
12     }
13
14     public Stack(int capacidad) {
15         this.capacidad = capacidad;
16         this.cima = null;
17         this.tamaño = 0;
18     }
19
20     public E push(E dato) {
21         if (isFull()) {
22             System.out.println("La pila está llena. No se puede insertar " + dato);
23             return null;
24         }
25         Nodo<E> nuevo = new Nodo<>(dato);
26         nuevo.siguiente = cima;
27         cima = nuevo;
28         tamaño++;
29         return dato;
30     }
31
32     public E pop() {
33         if (isEmpty()) {
34             System.out.println(x:"La pila está vacía. No se puede desapilar.");
35             return null;
36         }
37         E dato = cima.dato;
38         cima = cima.siguiente;
39         tamaño--;
40         return dato;
41     }
42
43     public void top() {
44         if (isEmpty()) {
45             System.out.println(x:"La pila está vacía.");
46         } else {
47             System.out.println("Cima: " + cima.dato);
48         }
49     }
50
51     public void destroyStack() {
52         cima = null;
53         tamaño = 0;
54         System.out.println(x:"Pila destruida (vacía).");
55     }
56
57     public boolean isEmpty() {
58         return cima == null;
59     }
60
61     public boolean isFull() {
62         return tamaño >= capacidad;
63     }
64
65     public void printStack() {
66         if (isEmpty()) {
67             System.out.println(x:"La pila está vacía.");
68             return;
69         }
70         Nodo<E> actual = cima;
71         System.out.print(s:"Pila: ");
72         while (actual != null) {
73             System.out.print(actual.dato + " ");
74             actual = actual.siguiente;
75         }
76         System.out.println();
77     }
78 }
79

```

Se usa el nodo genérico en la pila. Se controla el tamaño con una variable tamaño, y se puede establecer una capacidad máxima.

RESULTADOS:

```
ProblemasPropuestos > Ejercicio2 > Test.java > Test > main(String[])
1 package ProblemasPropuestos.Ejercicio2;
2 import java.util.*;
3
4 public class Test {
5     Run | Debug
6     public static void main(String[] args) {
7         Stack<Integer> pila = new Stack<>(capacidad:10);
8         Scanner sc = new Scanner(System.in);
9         int opcion;
10
11         do {
12             System.out.println(x:"\n=== Menú de Pila ===");
13             System.out.println(x:"1. Apilar (push)");
14             System.out.println(x:"2. Desapilar (pop)");
15             System.out.println(x:"3. Ver cima (top)");
16             System.out.println(x:"4. ¿Está vacía?");
17             System.out.println(x:"5. ¿Está llena?");
18             System.out.println(x:"6. Mostrar pila");
19             System.out.println(x:"7. Vaciar pila");
20             System.out.println(x:"0. Salir");
21             System.out.print(s:"Seleccione una opción: ");
22             opcion = sc.nextInt();
23
24             switch (opcion) {
25                 case 1:
26                     System.out.print(s:"Ingrese número a apilar: ");
27                     int dato = sc.nextInt();
28                     pila.push(dato);
29                     break;
30                 case 2:
31                     pila.pop();
32                     break;
33                 case 3:
34                     pila.top();
35                     break;
36                 case 4:
37                     System.out.println("¿Vacía?: " + pila.isEmpty());
38                     break;
39                 case 5:
40                     System.out.println("¿Llena?: " + pila.isFull());
41                     break;
42                 case 6:
43                     pila.printStack();
44                     break;
45                 case 7:
46                     pila.destroyStack();
47                     break;
48                 case 0:
49                     System.out.println(x:"Saliendo...");
50                     break;
51                 default:
52                     System.out.println(x:"Opción inválida.");
53             }
54             while (opcion != 0);
55             sc.close();
56         }
57     }
58 }
```

=== Menú de Pila ===	=== Menú de Pila ===	=== Menú de Pila ===	=== Menú de Pila ===
1. Apilar (push)	1. Apilar (push)	1. Apilar (push)	1. Apilar (push)
2. Desapilar (pop)	2. Desapilar (pop)	2. Desapilar (pop)	2. Desapilar (pop)
3. Ver cima (top)	3. Ver cima (top)	3. Ver cima (top)	3. Ver cima (top)
4. ¿Está vacía?	4. ¿Está vacía?	4. ¿Está vacía?	4. ¿Está vacía?
5. ¿Está llena?	5. ¿Está llena?	5. ¿Está llena?	5. ¿Está llena?
6. Mostrar pila	6. Mostrar pila	6. Mostrar pila	6. Mostrar pila
7. Vaciar pila	7. Vaciar pila	7. Vaciar pila	7. Vaciar pila
0. Salir	0. Salir	0. Salir	0. Salir
Seleccione una opción: 6	Seleccione una opción: 1	Seleccione una opción: 6	Seleccione una opción: 3
Pila: 5 4 3 2 1	Ingrese número a apilar: 6	Pila: 6 5 4 3 2 1	Cima: 6

```

=== Menú de Pila ===
1. Apilar (push)
2. Desapilar (pop)
3. Ver cima (top)
4. ¿Está vacía?
5. ¿Está llena?
6. Mostrar pila
7. Vaciar pila
0. Salir
Seleccione una opción: 5
¿Llena?: false
    
```

```

=== Menú de Pila ===
1. Apilar (push)
2. Desapilar (pop)
3. Ver cima (top)
4. ¿Está vacía?
5. ¿Está llena?
6. Mostrar pila
7. Vaciar pila
0. Salir
Seleccione una opción: 4
¿Vacía?: false
    
```

```

=== Menú de Pila ===
1. Apilar (push)
2. Desapilar (pop)
3. Ver cima (top)
4. ¿Está vacía?
5. ¿Está llena?
6. Mostrar pila
7. Vaciar pila
0. Salir
Seleccione una opción: 7
Pila destruida (vacía).
    
```

```

=== Menú de Pila ===
1. Apilar (push)
2. Desapilar (pop)
3. Ver cima (top)
4. ¿Está vacía?
5. ¿Está llena?
6. Mostrar pila
7. Vaciar pila
0. Salir
Seleccione una opción: 4
¿Vacía?: true
    
```

Usando la clase genérica: `public class Queue<E>` usar métodos genéricos `public E Metodo()`

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

```

ProblemasPropuestos > Ejercicio3 > Queue.java > Queue<E>
1  package ProblemasPropuestos.Ejercicio3;
2
3  public class Queue<E> {
4      private Nodo<E> frente;
5      private Nodo<E> fin;
6
7      public Queue() {
8          frente = null;
9          fin = null;
10     }
11
12     public E enqueue(E dato) {
13         Nodo<E> nuevo = new Nodo<>(dato);
14         if (isEmpty()) {
15             frente = nuevo;
16             fin = nuevo;
17         } else {
18             fin.siguiente = nuevo;
19             fin = nuevo;
20         }
21         return dato;
22     }
23
24     public void printQueue() {
25         Nodo<E> actual = frente;
26         System.out.print(s:"Cola: ");
27         while (actual != null) {
28             System.out.print(actual.dato + " ");
29             actual = actual.siguiente;
30         }
31         System.out.println();
32     }
33
34     public boolean isEmpty() {
35         return frente == null;
36     }
37 }
    
```

La cola tiene 2 referencias clave: frente y fin. Cada elemento se inserta al final (FIFO). Si la cola está vacía, frente y fin apuntan al mismo nodo. Luego, cada inserción actualiza el puntero fin.

RESULTADOS:

```

ProblemasPropuestos > Ejercicio3 > Test.java > Test > main(String[])
1  package ProblemasPropuestos.Ejercicio3;
2
3  public class Test {
4      public static void main(String[] args) {
5          Queue<Integer> cola = new Queue<>();
6
7          for (int i = 1; i <= 10; i++) {
8              cola.enqueue(i);
9          }
10
11         cola.printQueue();
12     }
13 }
14

```

```

PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab\Laboratorio5>
e:\10b26baafd4a20a8ab7b2d7976498cae\redhat.java\jdt_ws\Laboratorio5_f903f
Cola: 1 2 3 4 5 6 7 8 9 10
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab\Laboratorio5>

```

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.

```

ProblemasPropuestos > Ejercicio4 > Queue.java > Queue<E> > dequeue()
1  package ProblemasPropuestos.Ejercicio4;
2
3  public class Queue<E> {
4      private Nodo<E> frente;
5      private Nodo<E> fin;
6      private int tamaño;
7      private final int capacidad;
8
9      public Queue() {
10         this.capacidad = Integer.MAX_VALUE;
11         this.frente = null;
12         this.fin = null;
13         this.tamaño = 0;
14     }
15
16     public Queue(int capacidad) {
17         this.capacidad = capacidad;
18         this.frente = null;
19         this.fin = null;
20         this.tamaño = 0;
21     }
22
23     public E enqueue(E dato) {
24         if (isFull()) {
25             System.out.println("La cola está llena. No se puede insertar " + dato);
26             return null;
27         }
28         Nodo<E> nuevo = new Nodo<>(dato);
29         if (isEmpty()) {
30             frente = nuevo;
31             fin = nuevo;
32         } else {
33             fin.siguiente = nuevo;
34             fin = nuevo;
35         }
36         tamaño++;
37         return dato;
38     }
39
40     public E dequeue() {
41         if (isEmpty()) {
42             System.out.println("La cola está vacía. No se puede eliminar.");
43             return null;
44         }
45         E dato = frente.dato;
46         frente = frente.siguiente;
47         if (frente == null) fin = null;
48         tamaño--;
49         return dato;
50     }
51
52     public void destroyQueue() {
53         frente = null;
54         fin = null;
55         tamaño = 0;
56         System.out.println("Cola destruida (vacía).");
57     }
58
59     public boolean isEmpty() {
60         return frente == null;
61     }
62
63     public boolean isFull() {
64         return tamaño >= capacidad;
65     }
66 }

```

```

66
67 public void front() {
68     if (isEmpty()) {
69         System.out.println(x:"La cola está vacía.");
70     } else {
71         System.out.println("Frente: " + frente.dato);
72     }
73 }
74
75 public void back() {
76     if (isEmpty()) {
77         System.out.println(x:"La cola está vacía.");
78     } else {
79         System.out.println("Último: " + fin.dato);
80     }
81 }
82
83 public void printQueue() {
84     if (isEmpty()) {
85         System.out.println(x:"La cola está vacía.");
86         return;
87     }
88     Nodo<E> actual = frente;
89     System.out.print(s:"Cola: ");
90     while (actual != null) {
91         System.out.print(actual.dato + " ");
92         actual = actual.siguiente;
93     }
94     System.out.println();
95 }
96 }
97

```

Se mantiene el orden FIFO con dos referencias: frente y fin. enqueue() agrega al final; dequeue() elimina al frente. Se controla la capacidad máxima si se desea simular una cola limitada. front() y back() permiten inspeccionar los extremos sin eliminarlos. printQueue() recorre la cola mostrando todos los elementos en orden de llegada.

RESULTADOS:

```

ProblemasPropuestos > Ejercicio4 > Test.java > Test > main(String[])
1 package ProblemasPropuestos.Ejercicio4;
2 import java.util.Scanner;
3
4 public class Test {
5     Run | Debug
6     public static void main(String[] args) {
7         Queue<Integer> cola = new Queue<>(capacidad:10);
8         Scanner sc = new Scanner(System.in);
9         int opcion;
10
11         do {
12             System.out.println(x:"\n=== Menú de Cola ===");
13             System.out.println(x:"1. Encolar (enqueue)");
14             System.out.println(x:"2. Desencolar (dequeue)");
15             System.out.println(x:"3. Ver frente");
16             System.out.println(x:"4. Ver último");
17             System.out.println(x:"5. ¿Está vacía?");
18             System.out.println(x:"6. ¿Está llena?");
19             System.out.println(x:"7. Mostrar cola");
20             System.out.println(x:"8. Vaciar cola");
21             System.out.println(x:"0. Salir");
22             System.out.print(s:"Seleccione una opción: ");
23             opcion = sc.nextInt();
24

```

```

24         switch (opcion) {
25             case 1:
26                 System.out.print(s:"Ingrese número a encolar: ");
27                 int dato = sc.nextInt();
28                 cola.enqueue(dato);
29                 break;
30             case 2:
31                 cola.dequeue();
32                 break;
33             case 3:
34                 cola.front();
35                 break;
36             case 4:
37                 cola.back();
38                 break;
39             case 5:
40                 System.out.println("¿Vacía?: " + cola.isEmpty());
41                 break;
42             case 6:
43                 System.out.println("¿Llena?: " + cola.isFull());
44                 break;
45             case 7:
46                 cola.printQueue();
47                 break;
48             case 8:
49                 cola.destroyQueue();
50                 break;
51             case 0:
52                 System.out.println(x:"Saliendo...");
53                 break;
54             default:
55                 System.out.println(x:"Opción inválida.");
56         }
57     } while (opcion != 0);
58     sc.close();
59 }
60 }

```

```

=== Menú de Cola ===
1. Encolar (enqueue)
2. Desencolar (dequeue)
3. Ver frente
4. Ver último
5. ¿Está vacía?
6. ¿Está llena?
7. Mostrar cola
8. Vaciar cola
0. Salir
Seleccione una opción: 

```

Opción múltiple

IV. CUESTIONARIO

1. ¿Cuáles fueron las dificultades que encontraste al desarrollar los ejercicios propuestos? por ejemplo, poca documentación, complejidad del lenguaje, etc.
Una de las principales dificultades fue comprender y aplicar correctamente el uso de clases genéricas combinadas con estructuras dinámicas como los nodos enlazados. Aunque Java ofrece una sintaxis clara, al trabajar desde cero sin utilizar las clases de colecciones del lenguaje, surgen desafíos como manejar correctamente referencias nulas, evitar errores comunes como el NullPointerException y mantener el control de las referencias en estructuras enlazadas.
2. ¿Es posible reutilizar la clase nodo para otras estructuras de datos, además de listas enlazadas, pilas y colas?
Sí, la clase nodo es altamente reutilizable y adaptable para implementar otras estructuras de datos más complejas.
3. ¿Qué tipo de dato es NULL en java?
En Java, null no es un tipo de dato, sino un valor literal especial que representa la ausencia de una referencia a un objeto. Se utiliza para indicar que una variable de tipo referencia (como una clase o un array) no apunta a ninguna instancia en memoria.
4. ¿Cuáles son los beneficios de utilizar tipos genéricos en las pilas y colas?

	UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA	
Forma :o: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
Aprobación: 2022/03/01	Código: GUIA-PRLD-001	Página: 16
	UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA	
Forma :o: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
Aprobación: 2022/03/01	Código: GUIA-PRLD-001	Página: 7
<p>El uso de tipos genéricos en pilas y colas permite escribir código más flexible, reutilizable y seguro. Al implementar una estructura genérica como Stack<E> o Queue<E>, se evita tener que crear una nueva clase para cada tipo de dato, ya que con una sola implementación se puede trabajar con enteros, cadenas, objetos, etc.</p>		
V. REFERENCIAS Y BIBLIOGRAFÍA RECOMENDADAS: <ul style="list-style-type: none"> • Weiss M., Data Structures & Problem Solving Using Java, 2010, Addison-Wesley. • Weiss M., Data Structures and Algorithms Analysis in Java, 2012, Addison-Wesley. • Cormen T., Leiserson C., Rivest R., Stein C., Introduction to Algorithms, 2022, The MIT Press • The Java™ Tutorials - https://docs.oracle.com/javase/tutorial/ • Sedgewick, R., Algorithms in Java, Parts 1-4: Fundamentals, Data Structures, Sorting, Searching, Part 5: Graph Algorithms, Addison-Wesley. • Malik D., Data Structures Usign C++, 2003, Thomson Learning. • Knuth D., The Art of Computer Programming, Vol. 1 y 3, Addison - Wesley. 		

CRITERIOS DE EVALUACIÓN

Los criterios de evaluación se encuentran en el silabo DUFA ANEXO en la sección EVOLUCIÓN CONTINUA

TÉCNICAS E INSTRUMENTOS DE EVALUACIÓN	
TÉCNICAS: Actividades Resueltas Ejercicios Propuestos	INSTRUMENTOS: Rubricas