

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

## INFORME DE LABORATORIO

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		
INTEGRANTE (s): Davila Flores Mathias Dario				NOTA:	
DOCENTE(s): <ul style="list-style-type: none"> <li>Mg. Ing. Rene Alonso Nieto Valencia.</li> <li>ENLACE GITHUB: <a href="https://github.com/mathiasddf/LabsEDA">https://github.com/mathiasddf/LabsEDA</a></li> </ul>					

SOLUCIÓN Y RESULTADOS
<p><b>I. SOLUCIÓN DE EJERCICIOS/PROBLEMAS</b></p> <p><b>a. Ejercicios Resueltos:</b></p> <p><b>i. Ejercicio 1:</b> 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>

```
Lab5 > Ej_resueltos > Ejercicio1 > Main.java > ...
1 package Lab5.Ej_resueltos.Ejercicio1;
2
3 public class Main {
4     Run | Debug
5     public static void main(String[] args) {
6         StackList<Integer> stack = new StackList<>();
7         // Apilamos los valores 1 a 8
8         for (int i = 1; i <= 8; i++) {
9             stack.push(i);
10        }
11        // Mostramos el comportamiento LIFO
12        System.out.println("Desapilando elementos de la Pila (LIFO):");
13        while (!stack.isEmpty()) {
14            System.out.print(stack.pop() + " ");
15        }
16        System.out.println("\nTamaño final de la pila: " + stack.size());
17    }
18 }
19
```

PROBLEMAS 1 SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA> & 'C:\Program Files\BellSoft\LibericaJDK-21\bin\java.exe' '-XX:+ShowCodeOptionsMessages' '-cp' 'C:\Users\AORUS\AppData\Roaming\Code\User\workspaceStorage\00e40baa2889328f3ccf141a5869a9cd\redhat\_java\jdk\_ws\LabsEDA\_4af3ebda\bin' 'Lab5.Ejercicio1.Main'

Desapilando elementos de la Pila (LIFO):  
8 7 6 5 4 3 2 1  
Tamaño final de la pila: 0  
PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA>

- ii. **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.

```
Lab5 > Ej_resueltos > Ejercicio2 > Main.java > ...
1 package Lab5.Ej_resueltos.Ejercicio2;
2
3 public class Main {
4     Run | Debug
5     public static void main(String[] args) {
6         QueueList<Integer> queue = new QueueList<>();
7         // Encolamos los valores 1 a 8
8         for (int i = 1; i <= 8; i++) {
9             queue.enqueue(i);
10        }
11        // Mostramos el comportamiento FIFO
12        System.out.println("Atendiendo elementos de la cola (FIFO):");
13        while (!queue.isEmpty()) {
14            System.out.print(queue.dequeue() + " ");
15        }
16        System.out.println("\nTamaño final de la cola: " + queue.size());
17    }
18 }
19
```

PROBLEMAS 1 SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA> & 'C:\Program Files\BellSoft\LibericaJDK-21\bin\java.exe' '-XX:+ShowCodeOptionsMessages' '-cp' 'C:\Users\AORUS\AppData\Roaming\Code\User\workspaceStorage\00e40baa2889328f3ccf141a5869a9cd\redhat\_java\jdk\_ws\LabsEDA\_4af3ebda\bin' 'Lab5.Ejercicio2.Main'

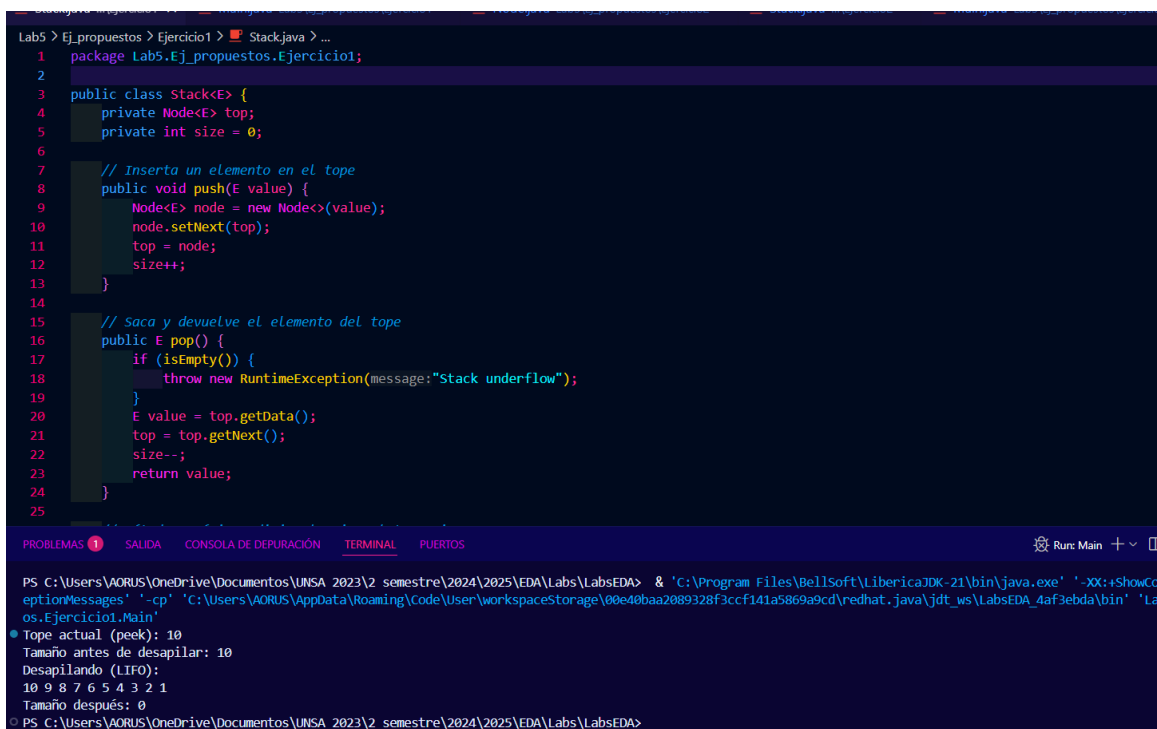
Atendiendo elementos de la Cola (FIFO):  
1 2 3 4 5 6 7 8  
Tamaño final de la cola: 0  
PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA>

	<p style="text-align: center;"><b>UNIVERSIDAD NACIONAL DE SAN AGUSTIN</b>  <b>FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS</b>  <b>ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</b></p>	
<p style="text-align: center;"><b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p><b>Aprobación:</b> 2022/03/01</p>	<p><b>Código:</b> GUIA-PRLE-001</p>	<p><b>Página:</b> 3</p>

## b. Ejercicios Propuestos:

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

Se crea una clase genérica Node<E> que enlaza cada elemento con el siguiente, y una clase Stack<E> que mantiene un puntero al “tope” y un contador de tamaño. Al insertar (push) se crea un nuevo nodo apuntando al antiguo tope; al desapilar (pop) se retira el nodo del tope y se actualiza el puntero. Con un bucle se apilan los enteros del 1 al 10 y luego se desapilan mostrando LIFO.



```

Lab5 > Ej_propuestos > Ejercicio1 > Stack.java > ...
1 package Lab5.Ej_propuestos.Ejercicio1;
2
3 public class Stack<E> {
4     private Node<E> top;
5     private int size = 0;
6
7     // Inserta un elemento en el tope
8     public void push(E value) {
9         Node<E> node = new Node<>(value);
10        node.setNext(top);
11        top = node;
12        size++;
13    }
14
15    // Saca y devuelve el elemento del tope
16    public E pop() {
17        if (isEmpty()) {
18            throw new RuntimeException(message:"Stack underflow");
19        }
20        E value = top.getData();
21        top = top.getNext();
22        size--;
23        return value;
24    }
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

PROBLEMAS 1 SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA> & 'C:\Program Files\BellSoft\LibericaJDK-21\bin\java.exe' '-XX:+ShowCodeDetails' '-cp' 'C:\Users\AORUS\AppData\Roaming\Code\User\workspaceStorage\00e40baa2089328f3ccf141a5869a9cd\redhat.java\jdt\_ws\LabsEDA\_4af3ebda\bin' 'Lab5.Ejercicio1.Main'

● Tope actual (peek): 10  
Tamaño antes de desapilar: 10  
Desapilando (LIFO):  
10 9 8 7 6 5 4 3 2 1  
Tamaño después: 0  
PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA>

- ii. **EJERCICIO 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.



A la misma estructura de nodos se le añaden los métodos clásicos: push, pop, top (peek), destroyStack (vacía toda la pila), isEmpty, isFull (comparando tamaño vs. capacidad) y printStack (recorre desde el tope imprimiendo cada dato). En la clase Principal se inicializa la pila con 1–10 y se presenta un menú de consola para invocar cada método y observar sus efectos en tiempo real.

```
Desapilado: 10

---- MENÚ DE PRUEBA DE MÉTODOS ----
1. push (apilar un valor)
2. pop (desapilar)
3. top (ver tope)
4. destroyStack (vaciar pila)
5. isEmpty
6. isFull
7. printStack
8. Salir
seleccione opción [1 - 8]: 7
Contenido de la pila: [ 9 8 7 6 5 4 3 2 1 ]

---- MENÚ DE PRUEBA DE MÉTODOS ----
1. push (apilar un valor)
2. pop (desapilar)
3. top (ver tope)
4. destroyStack (vaciar pila)
5. isEmpty
6. isFull
7. printStack
8. Salir
seleccione opción [1 - 8]: 1
Valor a apilar: 11

---- MENÚ DE PRUEBA DE MÉTODOS ----
1. push (apilar un valor)
2. pop (desapilar)
3. top (ver tope)
4. destroyStack (vaciar pila)
5. isEmpty
6. isFull
7. printStack
8. Salir
seleccione opción [1 - 8]: 7
contenido de la pila: [ 11 9 8 7 6 5 4 3 2 1 ]
```

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

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

Se define la misma clase Node<E> y una clase Queue<E> que lleva referencias a cabeza y cola. Al encolar (enqueue) se añade un nodo al final y se actualiza el puntero tail; al desencolar (dequeue) se retira el nodo de la cabeza y se avanza el puntero head. Se cargan los valores del 1 al 10 y luego se vacía la cola imprimiendo FIFO.



```

Lab5 > Ej_propuestos > Ejercicio3 > Queue.java > Queue<E>
1 public class Queue<E> {
2     // encolar al final
3
4     public void enqueue(E value) {
5         Node<E> n = new Node<>(value);
6         if (tail != null) {
7             tail.setNext(n);
8         } else {
9             head = n;
10        }
11        tail = n;
12    }
13
14    // desencolar del frente
15    public E dequeue() {
16        if (head == null) {
17            return null;
18        }
19        E val = head.getData();
20        head = head.getNext();
21        if (head == null) {
22            tail = null;
23        }
24        return val;
25    }
26
27 }
28
29
30

```

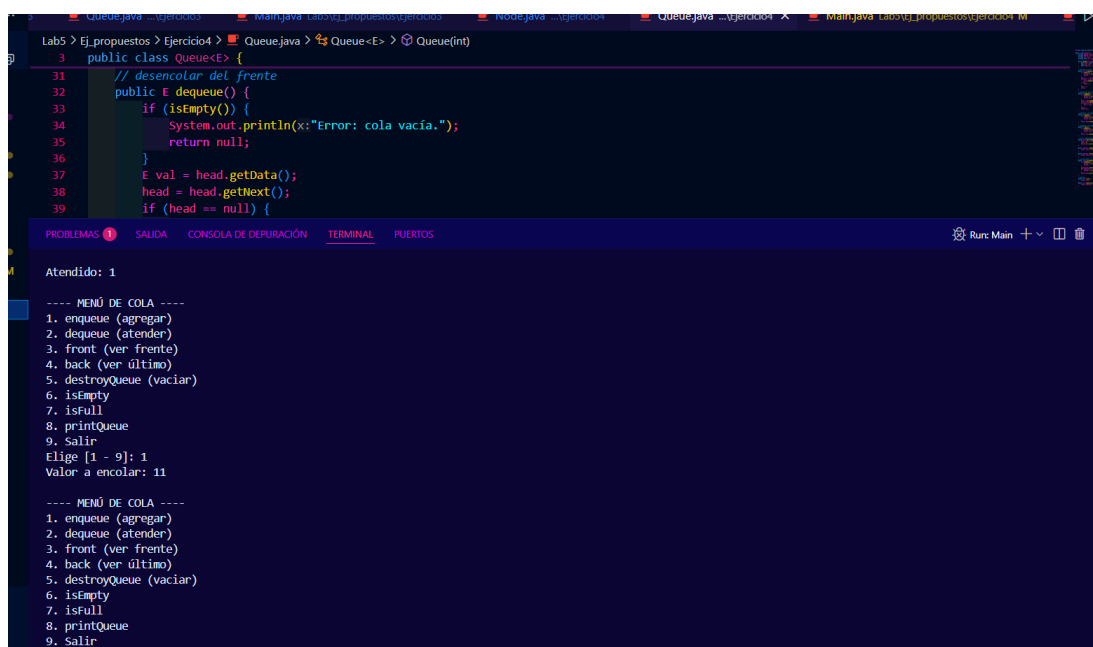
```

PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA> & 'C:\Program Files\BellSoft\LibericaJDK-21\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\AORUS\AppData\Roaming\Code\User\workspaceStorage\00e40baa2089328f3ccf141a5869a9cd\redhat_java\jdt_ws\LabsEDA_4af3ebda\bin' 'Lab5.Ej_propuestos.rc1c103.Main'
Frente (Metodo): 1
Desencolando todos los elementos: 1 2 3 4 5 6 7 8 9 10
PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA>

```

**iv. EJERCICIO 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

Sobre la cola enlazada se implementan: enqueue, dequeue, destroyQueue (vacía toda la cola), isEmpty, isFull (según tamaño y capacidad), front (ver cabeza), back (ver cola) y printQueue (recorrer e imprimir). La clase Principal carga 1-10 y ofrece un menú interactivo para probar cada operación y verificar visualmente el estado de la estructura.



```
Lab5 > Ej_propuestos > Ejercicio4 > Queue.java > Queue<E> > Queue(int)
3 public class Queue<E> {
31 // desencolar del frente
32 public E dequeue() {
33     if (isEmpty()) {
34         System.out.println(x:"Error: cola vacía.");
35         return null;
36     }
37     E val = head.getData();
38     head = head.getNext();
39     if (head == null) {
Atendido: 1
---- MENÚ DE COLA ----
1. enqueue (agregar)
2. dequeue (atender)
3. front (ver frente)
4. back (ver último)
5. destroyQueue (vaciar)
6. isEmpty
7. isFull
8. printQueue
9. Salir
Elige [1 - 9]: 1
Valor a encolar: 11
---- MENÚ DE COLA ----
1. enqueue (agregar)
2. dequeue (atender)
3. front (ver frente)
4. back (ver último)
5. destroyQueue (vaciar)
6. isEmpty
7. isFull
8. printQueue
9. Salir
Elige [1 - 9]:
```

	<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;"><b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p><b>Aprobación:</b> 2022/03/01</p>	<p><b>Código:</b> GUIA-PRLE-001</p>	<p><b>Página:</b> 7</p>

```

---- MENÚ DE COLA ----
1. enqueue (agregar)
2. dequeue (atender)
3. front (ver frente)
4. back (ver último)
5. destroyQueue (vaciar)
6. isEmpty
7. isFull
8. printQueue
9. Salir
Elige [1 - 9]: 1
Valor a encolar: 11

---- MENÚ DE COLA ----
1. enqueue (agregar)
2. dequeue (atender)
3. front (ver frente)
4. back (ver último)
5. destroyQueue (vaciar)
6. isEmpty
7. isFull
8. printQueue
9. Salir
Elige [1 - 9]: 8
Contenido: [ 2 3 4 5 6 7 8 9 10 11 ]

---- MENÚ DE COLA ----
1. enqueue (agregar)
2. dequeue (atender)
3. front (ver frente)
4. back (ver último)
5. destroyQueue (vaciar)
6. isEmpty
7. isFull
8. printQueue
9. Salir
Elige [1 - 9]: █

```

## II. SOLUCIÓN DEL CUESTIONARIO

- a. ¿Cuáles fueron las dificultades que encontraste al desarrollar los ejercicios propuestos? por ejemplo, poca documentación, complejidad del lenguaje, etc.
- **Comprender y manejar genéricos**, especialmente el *type erasure* y los límites de tipo, puede ser complejo al inicio, pues buena parte de la información en la documentación oficial asume ya familiaridad con estos conceptos
  - **Escasez de ejemplos sencillos** en la documentación de Oracle para implementaciones “desde cero” de listas enlazadas simples (la API estándar solo muestra la clase `LinkedList`, que es doblemente enlazada y muy completa)

	<p style="text-align: center;"><b>UNIVERSIDAD NACIONAL DE SAN AGUSTIN</b>  <b>FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS</b>  <b>ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</b></p>	
<p style="text-align: center;"><b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p><b>Aprobación:</b> 2022/03/01</p>	<p><b>Código:</b> GUIA-PRLE-001</p>	<p><b>Página:</b> 8</p>

- **NullPointerException** frecuentes al manipular nodos sin inicializar correctamente, lo que obliga a vigilar siempre referencias **null** en cada operación.

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

Una clase genérica `Node<E>` es, en esencia, un contenedor de datos y referencias que puede adaptarse a multitud de estructuras. Por ejemplo, en un árbol binario basta con extender la definición básica para que cada nodo tenga dos punteros (`left` y `right`). Así, el mismo `Node<T>` utilizado en la pila o la cola se convierte en y sirve como base para insertar, buscar o recorrer nodos en orden, preorden o postorden

De igual forma, en la representación de un grafo por listas de adyacencia, cada entrada del mapa puede apuntar a una lista enlazada de `Node<E>` que representen vecinos. No es necesario duplicar la lógica de la clase nodo: solo cambia el significado de “siguiente” (ahora varios vecinos en lugar de un único sucesor) y el código de recorrido (BFS/DFS) trabaja sobre la misma estructura

Más allá de árboles y grafos, `Node<E>` se emplea en estructuras como skip-lists, donde un nodo puede tener varios enlaces a distintos “niveles”, o en tablas hash con encadenamiento, donde cada cubeta contiene una lista de nodos. Esta versatilidad maximiza la reutilización de código, simplifica el mantenimiento y aprovecha al máximo la capacidad de los genéricos para trabajar con cualquier tipo de dato sin cambios de implementación.

**c. ¿Qué tipo de dato es NULL en java?**

En Java, `null` no es un objeto ni un tipo de dato primitivo, sino el único valor del llamado `null type`, un tipo especial definido por la especificación del lenguaje que no tiene nombre explícito. El literal `null` se forma con las tres letras en minúscula y puede asignarse a cualquier variable de tipo referencia; internamente, el compilador lo trata como una conversión de referencia ancha que siempre es válida, aunque carece de una representación `Class<?>` propia

Debido a que “el `null type` tiene un único valor—la referencia nula” y “no tiene nombre”, no es posible declarar variables de ese tipo ni hacerle cast directo, sino únicamente usar el literal `null` para indicar “ausencia de objeto”

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



	<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;"><b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p><b>Aprobación:</b> 2022/03/01</p>	<p><b>Código:</b> GUIA-PRLE-001</p>	<p><b>Página:</b> 9</p>

- **Seguridad de tipos en tiempo de compilación:** El compilador comprueba que solo se apilen o encolen objetos del tipo declarado, evitando así ClassCastException en ejecución.
- **Reutilización de código:** Una misma implementación genérica (Stack<T>, Queue<T>) sirve para cualquiera clase de objeto sin duplicar lógica, lo que reduce el mantenimiento y mejora la cohesión.
- **Eliminación de casteos explícitos:** Al recuperar elementos ya no es necesario convertir manualmente de Object al tipo deseado, lo que simplifica el código y evita errores de conversión.

### III. CONCLUSIONES

- Al implementar pilas y colas “desde cero” con nodos genéricos, afianzamos el entendimiento de cómo funcionan internamente estas colecciones dinámicas, más allá de usar directamente las clases de la biblioteca estándar.
- El diseño de una única clase Node<E> y de estructuras parametrizadas (Stack<T>, Queue<T>) demuestra cómo los genéricos promueven la reutilización de código, la seguridad de tipos en compilación y la claridad en la intención de cada estructura.
- Manejar correctamente null (ausencia de nodo) es esencial para evitar errores en tiempo de ejecución como NullPointerException. Cada operación debe considerar si la lista está vacía antes de acceder a punteros.
- La misma clase nodo puede adaptarse fácilmente a árboles, grafos o tablas hash con encadenamiento, lo que muestra la versatilidad del patrón “nodo enlazado” para representar cualquier red de datos conectados.

### REFERENCIAS Y BIBLIOGRAFÍA

- Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). *Data Structures and Algorithms in Java* (6th ed.). Wiley.
- Gosling, J., Joy, B., Steele, G., & Bracha, G. (2014). *The Java® Language Specification, Java SE 8 Edition*. Addison-Wesley.

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

- Horstmann, C. S., & Cornell, G. (2013). *Core Java Volume I—Fundamentals* (9th ed.). Prentice Hall.
- Oracle. (2014). *Generic methods*. En *The Java™ Tutorials*. Recuperado de <https://docs.oracle.com/javase/tutorial/extra/generics/methods.html>
- Oracle. (2014). *Generic types*. En *The Java™ Tutorials*. Recuperado de <https://docs.oracle.com/javase/tutorial/extra/generics/types.html>
- Oracle. (2014). *The Java® language specification, Java SE 8 edition: Chapter 4, Types, values, and variables (null type)*. Recuperado de <https://docs.oracle.com/javase/specs/jls/se8/html/jls-4.html#jls-4.1>