

	<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: 1</p>

INFORME DE LABORATORIO

(formato estudiante)

INFORMACIÓN BÁSICA					
ASIGNATURA:	ESTRUCTURA DE DATOS Y ALGORITMOS				
TÍTULO DE LA PRÁCTICA:	POO, HERENCIA, INTERFACES Y GENERICIDAD				
NÚMERO DE PRÁCTICA:	03	AÑO LECTIVO:	2025 – A	NRO. SEMESTRE:	Tercero III
FECHA DE PRESENTACIÓN	24/05/2025	HORA DE PRESENTACIÓN	23:59:59		
INTEGRANTE (s): Aragón Carpio Fredy José				NOTA:	
DOCENTE(s): <ul style="list-style-type: none"> Mg. Ing. Rene Alonso Nieto Valencia. 					

SOLUCIÓN Y RESULTADOS
<p>I. SOLUCIÓN DE EJERCICIOS/PROBLEMAS https://github.com/DARSEN12/Laboratorio_EDA_E.git</p> <h3>a. Problemas Desarrollados</h3> <h4>1. ArrayList:</h4> <pre>import java.util.ArrayList; public class ED1 { public static void main(String[] args) { ArrayList<String> alumnos = new ArrayList<String>(); ArrayList<Integer> notas = new ArrayList<Integer>(); alumnos.add("MARIA"); alumnos.add("DIEGO"); alumnos.add("RENE"); alumnos.add("ALONSO"); System.out.println(alumnos.hashCode()); System.out.println(alumnos.isEmpty()); System.out.println(alumnos.size()); } }</pre> <p>❖ Se crean dos listas vacías:</p>

- Una para almacenar nombres de alumnos (`ArrayList<String>`).
- Otra para posibles notas (`ArrayList<Integer>`).

- ❖ Se agregan 4 nombres a la lista `alumnos`: "MARIA", "DIEGO", "RENE", "ALONSO".
- ❖ Se imprime el hash code del objeto `alumnos` (valor numérico único según su contenido).
- ❖ Se verifica si la lista está vacía con `isEmpty()` → devuelve `false`.
- ❖ Se obtiene el tamaño de la lista con `size()` → devuelve 4.

2. Iterador:

```
import java.util.*;
public class ED2 {
    public static void main(String[] args) {
        ArrayList<String> alumnos = new ArrayList<String>();
        ArrayList<Integer> notas = new ArrayList<Integer>();
        alumnos.add("MARIA");
        alumnos.add("DIEGO");
        alumnos.add("RENE");
        alumnos.add("ALONSO");
        System.out.println(alumnos.hashCode());
        System.out.println(alumnos.isEmpty());
        System.out.println(alumnos.size());

        Iterator<String> itA = alumnos.iterator();
        while (itA.hasNext()) {
            System.out.println(itA.next());
        }
    }
}
```

- Se obtiene un iterador a partir de la lista `alumnos` con `alumnos.iterator()`.
- Se declara una variable `itA` de tipo `Iterator<String>` que permitirá recorrer la lista elemento por elemento.

- Se inicia un bucle `while` que se ejecuta mientras el iterador tenga un siguiente elemento (`itA.hasNext()` devuelve `true`).
- Dentro del bucle, se obtiene el siguiente elemento con `itA.next()`.
- Se imprime el valor obtenido en consola usando `System.out.println()`.
- El ciclo continúa hasta que se hayan recorrido todos los elementos de la lista.

3. Clase Animal en Java:

```
import java.util.*;

public class ED3 {
    public static void main(String[] args) {

        ArrayList<String> alumnos = new ArrayList<String>();
        ArrayList<Integer> notas = new ArrayList<Integer>();

        alumnos.add("MARIA");
        alumnos.add("DIEGO");
        alumnos.add("RENE");
        alumnos.add("ALONSO");

        System.out.println(alumnos.hashCode());
        System.out.println(alumnos.isEmpty());
        System.out.println(alumnos.size());

        Iterator<String> itA = alumnos.iterator();
        while (itA.hasNext()) {
            System.out.println(itA.next());
        }

        ArrayList<Animal> mascotas = new ArrayList<Animal>();
        ArrayList<Animal> mascotas1 = new ArrayList<Animal>();
        List<Animal> mascotas2 = new ArrayList<Animal>(); // ← corrección
aplicada aquí

        mascotas1.add(new Animal("FIRULAIS", true));
        mascotas1.add(new Animal("LUNA", false));
```

```
mascotas2.add(new Animal("TOM", true));
mascotas2.add(new Animal("MIAU", false));


System.out.println("\nMascotas1:");
for (Animal a : mascotas1) {
    System.out.println("Nombre: " + a.getNombre() + ", Género: " +
(a.isGenero() ? "Macho" : "Hembra"));
}

System.out.println("\nMascotas2:");
for (Animal a : mascotas2) {
    System.out.println("Nombre: " + a.getNombre() + ", Género: " +
(a.isGenero() ? "Macho" : "Hembra"));
}
}
```

El código ED3 . java demuestra el uso de colecciones genéricas en Java, trabajando con listas de tipo `ArrayList` para almacenar nombres de alumnos y objetos personalizados de tipo `Animal`. Primero, se agregan nombres a la lista `alumnos` y se realizan operaciones como impresión del `hashCode`, verificación de si está vacía y obtención del tamaño. Luego, se recorre la lista utilizando un `Iterator`. Posteriormente, se crean tres listas para manejar animales: `mascotas`, `mascotas1` y `mascotas2`. En este punto se aplica una corrección importante: se reemplaza la instanciación incorrecta `new List<Animal>()`, ya que `List` es una interfaz y no puede ser instanciada directamente, por `new ArrayList<Animal>()`, lo cual es correcto. Finalmente, se agregan objetos a las listas `mascotas1` y `mascotas2`, y se recorren con un bucle `for-each` para mostrar el nombre y género de cada animal. El código integra conceptos fundamentales de la programación orientada a objetos, el uso adecuado de interfaces y clases en colecciones, así como el manejo básico de iteradores.

b. Problemas Propuestos

1. Listas, Implementar una Lista usando POO con clases y métodos genéricos siguiendo los estándares de Java. (Los métodos para una lista)

	<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>

EP1: Clase principal que prueba operaciones básicas de la lista genérica.

```
package EP1;

public class EP1 {
    public static void main(String[] args) {
        Lista<String> lista = new Lista<>();

        lista.add("UNO");
        lista.add("DOS");
        lista.add("TRES");

        System.out.println("Tamaño: " + lista.size());
        System.out.println("Contiene 'DOS': " + lista.contains("DOS"));

        lista.remove("DOS");
        System.out.println("Tamaño luego de eliminar 'DOS': " + lista.size());
        System.out.println("Contiene 'DOS': " + lista.contains("DOS"));

        System.out.println("Elemento en posición 1: " + lista.get(1));

        lista.clear();
        System.out.println("Lista vacía: " + lista.isEmpty());
    }
}
```

Lista<T>: Implementa una lista enlazada genérica con métodos para agregar, eliminar y consultar elementos.

```
package EP1;

class Lista<T> {
    private Node<T> root;

    public Lista() {
        this.root = null;
    }

    public void add(T item) {
        Node<T> newNode = new Node<>(item);
        if (root == null) {
```

```
        root = newNode;
    } else {
        Node<T> current = root;
        while (current.nextNode != null) {
            current = current.nextNode;
        }
        current.nextNode = newNode;
    }
}

public void remove(T item) {
    if (root == null) return;

    if (root.data.equals(item)) {
        root = root.nextNode;
        return;
    }

    Node<T> current = root;
    while (current.nextNode != null) {
        if (current.nextNode.data.equals(item)) {
            current.nextNode = current.nextNode.nextNode;
            return;
        }
        current = current.nextNode;
    }
}

public boolean contains(T item) {
    Node<T> current = root;
    while (current != null) {
        if (current.data.equals(item)) {
            return true;
        }
        current = current.nextNode;
    }
    return false;
}
```

```
public int size() {
    int count = 0;
    Node<T> current = root;
    while (current != null) {
        count++;
        current = current.nextNode;
    }
    return count;
}

public T get(int index) {
    int count = 0;
    Node<T> current = root;
    while (current != null) {
        if (count == index) return current.data;
        count++;
        current = current.nextNode;
    }
    throw new IndexOutOfBoundsException("Index: " + index);
}

public void clear() {
    root = null;
}

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

Node<T>: Nodo genérico que almacena un dato y referencia al siguiente nodo.

```
package EP1;
```

```
class Node<T> {

    T data;
```

```
Node<T> nextNode;

public Node(T data) {

    this.data = data;

    this.nextNode = null;

}

public T getData() {

    return data;

}

public void setData(T data) {

    this.data = data;

}

public Node<T> getNextNode() {

    return nextNode;

}

public void setNextNode(Node<T> nextNode) {

    this.nextNode = nextNode;

}
```


	<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: 9</p>

```
}
```

Resultado:

- El programa crea una lista genérica, añade y elimina elementos, y verifica su contenido y tamaño.
- Muestra en consola el estado de la lista y los elementos en posiciones específicas.

```
PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 3>
c::; cd 'c:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 3';
& 'C:\Program Files\Eclipse Adoptium\jdk-17.0.7-hotspot\bin\java.exe'
'-XX:+ShowCodeDetailsInExceptionMessages' '-cp'
'C:\Users\Usuario\AppData\Roaming\Code\User\workspaceStorage\8d0757327a5b51a2d
7b25d70f16e79e7\redhat.java\jdt_ws\Sesión 3_c6ea4ef9\bin' 'EP1.EP1'
```

Tamaño: 3

Contiene 'DOS': true

Tamaño luego de eliminar 'DOS': 2

Contiene 'DOS': false

Elemento en posición 1: TRES

Lista vacía: true

```
PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 3>
```

Este proyecto implementa una lista enlazada genérica en Java utilizando programación orientada a objetos (POO). Se divide en tres clases: **Node<T>**, que representa cada nodo con un dato y una referencia al siguiente; **Lista<T>**, que maneja la lógica de la lista (agregar, eliminar, buscar, obtener por índice, limpiar, verificar si está vacía); y **EP1**, que actúa como clase principal para probar su funcionamiento. Todo se basa en el uso de tipos genéricos (<T>), permitiendo reutilizar la lista con cualquier tipo de dato. Es un ejemplo claro de estructuras dinámicas, encapsulamiento y reutilización de código.

	<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: 10</p>

2. Calculadora Genérica: Cree un nuevo proyecto en Java con el nombre **CalculadoraGenerica**. Implemente una clase genérica llamada **Operador<T extends Number>**, que contenga los atributos **valor1** y **valor2**, además de un constructor público **Operador(T valor1, T valor2)** para inicializar dichos valores. Dentro de esta clase, implemente métodos genéricos de instancia que realicen las siguientes operaciones: **suma**, **resta**, **producto**, **división** (validando división por cero) y **potencia**, retornando todos los resultados como **double**. Adicionalmente, defina dos métodos estáticos genéricos: **raizCuadrada(T valor)** y **raizCubica(T valor)**, los cuales calculen la raíz cuadrada y cúbica, respectivamente, validando si corresponde. En la clase principal (**Main** o **EP2**), implemente un menú de opciones por consola que permita al usuario seleccionar entre las siguientes operaciones: 1. Suma, 2. Resta, 3. Producto, 4. División, 5. Potencia, 6. Raíz Cuadrada, 7. Raíz Cúbica, 8. Salir del Programa. El programa debe solicitar al usuario el ingreso de valores (validando el tipo de dato entre **Integer** y **Double**), instanciar objetos de tipo **Operador** y mostrar los resultados de las operaciones elegidas. El sistema debe continuar funcionando hasta que se seleccione explícitamente la opción 8 (salir). Se espera que el desarrollo demuestre el uso correcto de clases genéricas, entrada por consola y manejo de excepciones en Java.

Clase EP2 (principal):

Controla la interacción con el usuario mediante un menú, lee datos de entrada, crea instancias de **Operador<Double>** para operaciones binarias y llama a métodos estáticos para operaciones unarias. Gestiona el flujo, validaciones y manejo de excepciones.

```
package EP2;

import java.util.*;

public class EP2 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
```

```
int opcion;

do {
    System.out.println("\nMenú de Operaciones Clases Genéricas:");
    System.out.println("1. Suma");
    System.out.println("2. Resta");
    System.out.println("3. Producto");
    System.out.println("4. División");
    System.out.println("5. Potencia");
    System.out.println("6. Raíz Cuadrada");
    System.out.println("7. Raíz Cúbica");
    System.out.println("8. Salir del Programa");
    System.out.print("Seleccione una opción: ");

    opcion = scanner.nextInt();

    switch (opcion) {
        case 1 -> {
            System.out.print("Valor 1: ");
            double a = scanner.nextDouble();
            System.out.print("Valor 2: ");
            double b = scanner.nextDouble();
            Operador<Double> op = new Operador<>(a, b);
            System.out.println("Resultado: " + op.suma());
        }
        case 2 -> {
            System.out.print("Valor 1: ");
            double a = scanner.nextDouble();
            System.out.print("Valor 2: ");
            double b = scanner.nextDouble();
            Operador<Double> op = new Operador<>(a, b);
            System.out.println("Resultado: " + op.rest());
        }
        case 3 -> {
            System.out.print("Valor 1: ");
            double a = scanner.nextDouble();
            System.out.print("Valor 2: ");
            double b = scanner.nextDouble();
```

```
        Operador<Double> op = new Operador<>(a, b);
        System.out.println("Resultado: " + op.producto());
    }
    case 4 -> {
        try {
            System.out.print("Valor 1: ");
            double a = scanner.nextDouble();
            System.out.print("Valor 2: ");
            double b = scanner.nextDouble();
            Operador<Double> op = new Operador<>(a, b);
            System.out.println("Resultado: " + op.division());
        } catch (ArithmeticException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
    case 5 -> {
        System.out.print("Base: ");
        double a = scanner.nextDouble();
        System.out.print("Exponente: ");
        double b = scanner.nextDouble();
        Operador<Double> op = new Operador<>(a, b);
        System.out.println("Resultado: " + op.potencia());
    }



    case 6 -> {
        try {
            System.out.print("Valor: ");
            double a = scanner.nextDouble();
            System.out.println("Resultado: " +
Operador.raizCuadrada(a));
        } catch (ArithmeticException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
    case 7 -> {
        System.out.print("Valor: ");
        double a = scanner.nextDouble();
        System.out.println("Resultado: " + Operador.raizCubica(a));
    }
}
```

```
    }  
    case 8 -> System.out.println("Saliendo del programa...");  
    default -> System.out.println("Opción no válida. Intente  
nuevamente.");  
    }  
    } while (opcion != 8);  
  
    scanner.close();  
}  
}
```

Clase genérica `Operador<T extends Number>`:

Representa un operador matemático que almacena dos valores genéricos (`valor1`, `valor2`) y proporciona métodos para realizar operaciones matemáticas básicas (suma, resta, producto, división, potencia) como métodos de instancia. Además, define métodos estáticos para cálculo de raíz cuadrada y cúbica.

```
package EP2;  
  
public class Operador<T extends Number> {  
    private T valor1;  
    private T valor2;  
  
    public Operador(T valor1, T valor2) {  
        this.valor1 = valor1;  
        this.valor2 = valor2;  
    }  
  
    public T getValor1() {  
        return valor1;  
    }  
  
    public T getValor2() {  
        return valor2;  
    }  
  
    public double suma() {  
        return valor1.doubleValue() + valor2.doubleValue();  
    }  
}
```

	<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: 14</p>

```

public double resta() {
    return valor1.doubleValue() - valor2.doubleValue();
}

public double producto() {
    return valor1.doubleValue() * valor2.doubleValue();
}

public double division() {
    if (valor2.doubleValue() == 0) throw new ArithmeticException("División
por cero");
    return valor1.doubleValue() / valor2.doubleValue();
}

public double potencia() {
    return Math.pow(valor1.doubleValue(), valor2.doubleValue());
}

public static <T extends Number> double raizCuadrada(T valor) {
    if (valor.doubleValue() < 0) throw new ArithmeticException("Raíz de
negativo");
    return Math.sqrt(valor.doubleValue());
}

public static <T extends Number> double raizCubica(T valor) {
    return Math.cbrt(valor.doubleValue());
}
}

```

Resultado:

- El programa muestra el resultado de la operación matemática solicitada con precisión **double** en consola.
- En caso de errores como división por cero o raíz negativa, muestra mensajes claros sin terminar abruptamente.
- Permite realizar múltiples operaciones hasta que el usuario elige salir.

PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 3>
c.; cd 'c:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 3';

	<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: 15

```
& 'C:\Program Files\Eclipse Adoptium\jdk-17.0.7-hotspot\bin\java.exe'
'-XX:+ShowCodeDetailsInExceptionMessages' '-cp'
'C:\Users\Usuario\AppData\Roaming\Code\User\workspaceStorage\8d0757327a5b51a2d7b25d70f16e79e7\redhat.java\jdt_ws\Sesión 3_c6ea4ef9\bin' 'EP2.EP2'
```

Menú de Operaciones Clases Genéricas:

1. Suma
2. Resta
3. Producto
4. División
5. Potencia
6. Raíz Cuadrada
7. Raíz Cúbica
8. Salir del Programa

Seleccione una opción: 3

Valor 1: 4

Valor 2: 2

Resultado: 8.0

Menú de Operaciones Clases Genéricas:

1. Suma
2. Resta
3. Producto
4. División
5. Potencia
6. Raíz Cuadrada
7. Raíz Cúbica
8. Salir del Programa

Seleccione una opción: 8

Saliendo del programa...

PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 3>

Este programa implementa una calculadora genérica en Java, utilizando clases y métodos genéricos para operar con números del tipo `Integer` o `Double`. Se compone de la clase `Operador<T extends Number>`, que almacena dos valores numéricos y

permite realizar operaciones básicas como suma, resta, multiplicación, división, potencia, y además incluye métodos estáticos para calcular la raíz cuadrada y cúbica. En la clase principal EP2, se despliega un menú interactivo por consola que permite al usuario seleccionar la operación deseada, ingresar valores numéricos y ver el resultado. El programa valida errores comunes como división por cero o raíz de número negativo y solo finaliza cuando se selecciona la opción "Salir". Esta solución demuestra el uso práctico de POO con genéricos, manejo de excepciones y entrada de datos en Java.

II. SOLUCIÓN DEL CUESTIONARIO

1. Dificultades al desarrollar los ejercicios propuestos

Las principales dificultades encontradas fueron:

- **Poca documentación oficial en español:** gran parte del contenido relevante está en inglés, lo que puede dificultar la comprensión para algunos estudiantes.
- **Complejidad del lenguaje Java en temas avanzados:** al trabajar con estructuras como ArrayList, HashMap o clases genéricas, se requiere entender conceptos como referencias, tipos parametrizados o el uso de interfaces.
- **Errores por falta de práctica:** errores comunes como `IndexOutOfBoundsException`, `NullPointerException` o confusión entre tipos primitivos y objetos (`int` vs `Integer`).

2. Diferencia entre `List` y `ArrayList` en Java

Aspecto	<code>List</code>	<code>ArrayList</code>
Tipo	Es una interfaz (abstracta)	Es una clase concreta que implementa <code>List</code>

Uso	Define un contrato: operaciones como <code>add()</code> , <code>get()</code> , <code>remove()</code>	Implementa esas operaciones con un arreglo dinámico
------------	--	--

Ventaja	Flexibilidad: permite cambiar la implementación sin afectar el código (polimorfismo)	Uso directo con comportamiento definido
----------------	--	---

Ejemplo	<code>List<String> lista = new ArrayList<>();</code>	<code>ArrayList<String> lista = new ArrayList<>();</code>
----------------	--	---

3. Beneficios y oportunidades de las clases genéricas en Java

Las **clases genéricas** (generics) permiten definir clases, interfaces y métodos con **tipos de datos paramétricos**, lo que ofrece:

Beneficios:

- III. **Reutilización de código:** se puede usar una sola clase para múltiples tipos (`Stack<T>`, `List<T>`, etc.).
- IV. **Seguridad de tipos en tiempo de compilación:** evita errores por conversiones de tipo inadecuadas (elimina necesidad de `cast`).
- V. **Legibilidad y mantenimiento:** se sabe exactamente qué tipo de dato se está usando (`List<String>` es más claro que `List`).
- VI. **Mejor integración con colecciones de Java (Collections Framework).**

II. CONCLUSIONES

Durante el desarrollo de los ejercicios en Java se identificaron diversas dificultades, principalmente relacionadas con la comprensión del lenguaje, la escasa documentación en español y la complejidad al implementar estructuras como listas genéricas. No obstante, esto permitió reconocer la importancia de entender las diferencias entre `List` y

	<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: 18</p>

ArrayList, siendo clave para aplicar correctamente el principio de programación orientada a interfaces. Además, se valoró el uso de clases genéricas por los beneficios que ofrecen en cuanto a reutilización, seguridad de tipos y claridad del código. En conjunto, estos aprendizajes fortalecen las competencias en diseño y desarrollo de soluciones eficientes y escalables dentro del enfoque de ingeniería de software.

RETROALIMENTACIÓN GENERAL

REFERENCIAS Y BIBLIOGRAFÍA