

	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: 1


GUÍA DE LABORATORIO

(formato docente)

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
TIPO DE PRÁCTICA:	INDIVIDUAL	Josue Claudio Quispe Pauccar			
	GRUPAL	—	MÁXIMO DE ESTUDIANTES	01	
FECHA INICIO:	19/05/2025	FECHA FIN:	23/05/2025	DURACIÓN:	90 minutos.
RECURSOS A UTILIZAR:					
<ul style="list-style-type: none"> 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 Herencia, Interfaces y Genericidad. Aplicar conceptos elementales de programación a resolver utilizando POO en problemas de algoritmos. Desarrollar pruebas. 	
TEMAS:	
<ul style="list-style-type: none"> Introducción. TAD. POO, Herencia, Interfaces y Genericidad. 	
COMPETENCIAS	C.a
	C.b
	C.c
	C.d

	<p align="center">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
Forma :o: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
Aprobación: 2022/03/01	Código: GUIA-PRLD-001	Página: 2

CONTENIDO DE LA GUÍA

I. MARCO CONCEPTUAL

- <https://www.w3schools.com/java/>
- <https://docs.oracle.com/javase/7/docs/api/java/util/List.html>
- <https://docs.oracle.com/javase/tutorial/java/generics/types.html>

II. EJERCICIO/PROBLEMA RESUELTO POR EL DOCENTE

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

1. ArrayList:

```
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());
```

2. Iterador:

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

3. Clase Animal en Java:

Animal

```

public class Animal {
    String nombre;
    boolean genero;

    public Animal(String nombre, boolean genero) {
        super(nombre, genero);
        this.nombre = nombre;
        this.genero = genero;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}

```

```

public boolean isGenero() { return genero; }
public void setGenero(boolean genero) {
    this.genero = genero;
}
}

```

```

ArrayList<Animal> mascotas = new ArrayList<Animal>();
ArrayList<Animal> mascotas2 = new ArrayList<Animal>();
// Ver error */

```

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.

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

- Referencia: <https://docs.oracle.com/javase/7/docs/api/java/util/List.html>
- Puede **ignorar** los siguientes métodos:

hashCode()

iterator()

listIterator()

listIterator(int index)

retainAll(Collection<?> c)

toArray()

toArray(T[] a)

- Implemente una clase **Node<T>** donde **T** es un **tipo genérico**, esta clase debe contener al menos dos propiedades.

<https://docs.oracle.com/javase/tutorial/java/generics/types.html>

T data: la información almacenada en el nodo Node<T> nextNode: una referencia al siguiente nodo

- Implementar una clase List<T> esta clase debe contener al menos esta propiedad
Node<T> root: la referencia sobre el nodo inicial

```
1 package Laboratorio3;
2
3 public class Node<T> {
4     T data;
5     Node<T> nextNode;
6
7     public Node(T data) {
8         this.data = data;
9         this.nextNode = null;
10    }
11
12    public T getData() {
13        return data;
14    }
15
16    public void setData(T data) {
17        this.data = data;
18    }
19
20    public Node<T> getNextNode() {
21        return nextNode;
22    }
23
24    public void setNextNode(Node<T> nextNode) {
25        this.nextNode = nextNode;
26    }
27
28 }
```

```
Laboratorio3 > MyList.java > MyList<T> > add(T)
1 package Laboratorio3;
2
3 public class MyList<T> {
4     private Node<T> root;
5     private int size;
6
7     public MyList() {
8         root = null;
9         size = 0;
10    }
11
12    public int size() {
13        return size;
14    }
15
16    public boolean isEmpty() {
17        return size == 0;
18    }
19
20    public boolean add(T element) {
21        Node<T> newNode = new Node<>(element);
22        if (root == null) {
23            root = newNode;
24        } else {
25            Node<T> current = root;
26            while (current.getNextNode() != null) {
27                current = current.getNextNode();
28            }
29            current.setNextNode(newNode);
30        }
31        size++;
32        return true;
33    }
}
```

```
35     public void add(int index, T element) {
36         checkIndexForAdd(index);
37         Node<T> newNode = new Node<>(element);
38         if (index == 0) {
39             newNode.setNextNode(root);
40             root = newNode;
41         } else {
42             Node<T> prev = getNode(index - 1);
43             newNode.setNextNode(prev.getNextNode());
44             prev.setNextNode(newNode);
45         }
46         size++;
47     }
48
49     public T get(int index) {
50         checkIndex(index);
51         return getNode(index).getData();
52     }
53
54     public T set(int index, T element) {
55         checkIndex(index);
56         Node<T> node = getNode(index);
57         T old = node.getData();
58         node.setData(element);
59         return old;
60     }
61
62     public T remove(int index) {
63         checkIndex(index);
64         T removed;
65         if (index == 0) {
66             removed = root.getData();
67             root = root.getNextNode();
68         } else {
69             Node<T> prev = getNode(index - 1);
70             removed = prev.getNextNode().getData();
71             prev.setNextNode(prev.getNextNode().getNextNode());
72         }
73         size--;
74         return removed;
75     }
76
77     public boolean contains(T element) {
78         return indexOf(element) != -1;
79     }
80
81     public int indexOf(T element) {
82         Node<T> current = root;
83         int index = 0;
84         while (current != null) {
85             if ((element == null && current.getData() == null) ||
86                 (element != null && element.equals(current.getData()))) {
87                 return index;
88             }
89             current = current.getNextNode();
90             index++;
91         }
92         return -1;
93     }
94 }
```

```
95     public void clear() {
96         root = null;
97         size = 0;
98     }
99
100     private Node<T> getNode(int index) {
101         Node<T> current = root;
102         for (int i = 0; i < index; i++) {
103             current = current.getNextNode();
104         }
105         return current;
106     }
107
108     private void checkIndex(int index) {
109         if (index < 0 || index >= size)
110             throw new IndexOutOfBoundsException("Index: " + index + ", Size: " + size);
111     }
112
113     private void checkIndexForAdd(int index) {
114         if (index < 0 || index > size)
115             throw new IndexOutOfBoundsException("Index: " + index + ", Size: " + size);
116     }
```

```
117
118     @Override
119     public String toString() {
120         StringBuilder sb = new StringBuilder(str:"[");
121         Node<T> current = root;
122         while (current != null) {
123             sb.append(current.getData());
124             if (current.getNextNode() != null)
125                 sb.append(str:", ");
126             current = current.getNextNode();
127         }
128         sb.append(str:"]");
129         return sb.toString();
130     }
131 }
132
```

```
6     public class Test {
7         public static void main(String[] args) {
8
23             MyList<String> lista = new MyList<>();
24
25             // Agregar elementos
26             lista.add(element:"Uno");
27             lista.add(element:"Dos");
28             lista.add(element:"Tres");
29             System.out.println("Lista después de agregar elementos: " + lista);
30
31             // Obtener elementos
32             System.out.println("Elemento en la posición 1: " + lista.get(index:1)); // Debe ser "Dos"
33
34             // Insertar en una posición específica
35             lista.add(index:1, element:"Insertado");
36             System.out.println("Lista después de insertar en índice 1: " + lista);
37
38             // Eliminar un elemento
39             String eliminado = lista.remove(index:2);
40             System.out.println("Elemento eliminado en índice 2: " + eliminado);
41             System.out.println("Lista después de eliminar: " + lista);
42
43             // Buscar elementos
44             System.out.println("Contiene 'Uno': " + lista.contains(element:"Uno")); // true
45             System.out.println("Índice de 'Tres': " + lista.indexOf(element:"Tres")); // 2
46
47             // Reemplazar un elemento
48             lista.set(index:0, element:"Primero");
49             System.out.println("Lista después de reemplazar índice 0: " + lista);
50
51             // Tamaño de la lista
52             System.out.println("Tamaño actual: " + lista.size());
53
54             // Vaciar lista
55             lista.clear();
56             System.out.println("Lista después de limpiar: " + lista);
57         }
```




2. Calculadora Genérica, Cree un nuevo proyecto en Java: **CalculadoraGenerica**.

- Implementar las clases Genérica **Operador<T>**, para declarar sus **atributos** (valor1 y valor2), **constructor** (Operador).




```
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 class Main {  
  
    static <T extends Number> double suma(T valor1, T valor2) {  
        return (valor1.doubleValue() + valor2.doubleValue());  
    }  
  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
        int opcion;
```


- Escribir la clase **Main**, e implementar el método genérico suma con la siguiente estructura.
- Escribir los métodos genéricos: **resta, producto, división, potencia, raíz cuadrada y raíz cubica**.
- Para poder probar las clases y métodos genéricos implementar mediante un menú de opciones las operaciones, mostrando los resultados:
Menú de Operaciones Clases Genéricas:
 1. Suma.
 2. Resta.
 3. Producto.
 4. División.
 5. Potencia.
 6. Raíz Cuadrada.
 7. Raíz Cubica.
 8. Salir del Programa.
- **Nota:** El programa debe permitir validar entre valores o tipo de dato (**integer o double**) para poder utilizar los métodos genéricos, el programa no termina hasta escoger la opción **SALIR**.

Laboratorio3 >  Operator.java >  Operator<T extends Number> >  getValor2()

```
1 package Laboratorio3;
2
3 public class Operator<T extends Number>{
4     private T valor1;
5     private T valor2;
6
7     public Operator(T valor1, T valor2) {
8         this.valor1 = valor1;
9         this.valor2 = valor2;
10    }
11
12    public T getValor1() {
13        return valor1;
14    }
15
16    public T getValor2() {
17        return valor2;
18    }
19 }
20
```

Laboratorio3 >  TestOperator.java >  TestOperator >  ejecutarRaiz(int, T) <T extends Number>

```
1 package Laboratorio3;
2
3 import java.util.Scanner;
4
5 public class TestOperator {
6     static <T extends Number> double suma(T valor1, T valor2) {
7         return valor1.doubleValue() + valor2.doubleValue();
8     }
9
10    static <T extends Number> double resta(T valor1, T valor2) {
11        return valor1.doubleValue() - valor2.doubleValue();
12    }
13
14    static <T extends Number> double producto(T valor1, T valor2) {
15        return valor1.doubleValue() * valor2.doubleValue();
16    }
17
18    static <T extends Number> double division(T valor1, T valor2) {
19        if (valor2.doubleValue() == 0) {
20            System.out.println(x:"Error: División por cero.");
21            return Double.NaN;
22        }
23        return valor1.doubleValue() / valor2.doubleValue();
24    }
25
26    static <T extends Number> double potencia(T valor1, T valor2) {
27        return Math.pow(valor1.doubleValue(), valor2.doubleValue());
28    }
29
30    static <T extends Number> double raizCuadrada(T valor) {
31        if (valor.doubleValue() < 0) {
32            System.out.println(x:"Error: Raíz cuadrada de número negativo.");
33            return Double.NaN;
34        }
35        return Math.sqrt(valor.doubleValue());
36    }
37 }
```

```
37
38 static <T extends Number> double raizCubica(T valor) {
39     return Math.cbrt(valor.doubleValue());
40 }
41
42 Run | Debug
43 public static void main(String[] args) {
44     Scanner scanner = new Scanner(System.in);
45     int opcion;
46
47     do {
48         System.out.println(x: "\nMenú de Operaciones Clases Genéricas:");
49         System.out.println(x: "1. Suma");
50         System.out.println(x: "2. Resta");
51         System.out.println(x: "3. Producto");
52         System.out.println(x: "4. División");
53         System.out.println(x: "5. Potencia");
54         System.out.println(x: "6. Raíz Cuadrada");
55         System.out.println(x: "7. Raíz Cúbica");
56         System.out.println(x: "8. Salir del Programa");
57         System.out.print(s: "Seleccione una opción: ");
58         opcion = scanner.nextInt();
59
60         if (opcion >= 1 && opcion <= 7) {
61             System.out.print(s: "¿Tipo de dato? (1: Integer, 2: Double): ");
62             int tipo = scanner.nextInt();
```

```
63         switch (opcion) {
64             case 1, 2, 3, 4, 5 -> {
65                 if (tipo == 1) {
66                     System.out.print(s: "Ingrese valor 1: ");
67                     Integer i1 = scanner.nextInt();
68                     System.out.print(s: "Ingrese valor 2: ");
69                     Integer i2 = scanner.nextInt();
70                     ejecutarOperacion(opcion, i1, i2);
71                 } else {
72                     System.out.print(s: "Ingrese valor 1: ");
73                     Double d1 = scanner.nextDouble();
74                     System.out.print(s: "Ingrese valor 2: ");
75                     Double d2 = scanner.nextDouble();
76                     ejecutarOperacion(opcion, d1, d2);
77                 }
78             }
79             case 6, 7 -> {
80                 if (tipo == 1) {
81                     System.out.print(s: "Ingrese el valor: ");
82                     Integer i = scanner.nextInt();
83                     ejecutarRaiz(opcion, i);
84                 } else {
85                     System.out.print(s: "Ingrese el valor: ");
86                     Double d = scanner.nextDouble();
87                     ejecutarRaiz(opcion, d);
88                 }
89             }
90         }
91     }
```

```
91         } else if (opcion == 8) {
92             System.out.println(x:"Saliendo del programa...");
93         } else {
94             System.out.println(x:"Opción no válida.");
95         }
96     }
97     while (opcion != 8);
98
99     scanner.close();
100 }
101
102 // Método auxiliar para operaciones con dos operandos
103 static <T extends Number> void ejecutarOperacion(int opcion, T valor1, T valor2) {
104     double resultado = switch (opcion) {
105         case 1 -> suma(valor1, valor2);
106         case 2 -> resta(valor1, valor2);
107         case 3 -> producto(valor1, valor2);
108         case 4 -> division(valor1, valor2);
109         case 5 -> potencia(valor1, valor2);
110         default -> 0;
111     };
112     System.out.println("Resultado: " + resultado);
113 }
114
115 // Método auxiliar para operaciones con un solo operando
116 static <T extends Number> void ejecutarRaiz(int opcion, T valor) {
117     double resultado = (opcion == 6) ? raizCuadrada(valor) : raizCubica(valor);
118     System.out.println("Resultado: " + resultado);
119 }
120 }
```

	<p align="center">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
Forma :o: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
Aprobación: 2022/03/01	Código: GUIA-PRLD-001	Página: 13

IV. CUESTIONARIO

1. ¿Cuáles fueron las dificultades que encontraste al desarrollar los ejercicios propuestos? por ejemplo, poca documentación, complejidad del lenguaje, etc.

Durante el desarrollo de este ejercicio, una de las principales dificultades fue comprender y aplicar correctamente los conceptos de clases y métodos genéricos en Java, especialmente al trabajar con tipos numéricos. Fue necesario convertir los valores genéricos a tipos concretos como `double` para realizar cálculos, lo cual puede ser confuso al principio.

2. ¿Qué diferencia hay entre un List y un ArrayList en Java?

La diferencia entre List y ArrayList en Java radica en que List es una interfaz que define un conjunto de operaciones para estructuras de tipo lista, mientras que ArrayList es una clase concreta que implementa esa interfaz utilizando un arreglo dinámico como estructura interna. Usar List como tipo de referencia permite cambiar la implementación sin modificar mucho el código, por ejemplo, intercambiar ArrayList por LinkedList.

3. ¿Qué beneficios y oportunidades ofrecen las clases genéricas en Java?

Las clases genéricas en Java ofrecen numerosos beneficios, como la reutilización del código, permitiendo escribir clases o métodos que funcionan con distintos tipos de datos sin necesidad de duplicar la lógica. También mejoran la seguridad en tiempo de compilación al garantizar que solo se utilicen los tipos esperados, evitando errores comunes de casteo. Además, eliminan la necesidad de conversiones explícitas de tipos, hacen que el código sea más limpio y fácil de mantener, y permiten construir estructuras y algoritmos altamente flexibles y reutilizables para una gran variedad de casos.

V. REFERENCIAS Y BIBLIOGRAFÍA RECOMENDADAS:

- 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.

TÉCNICAS E INSTRUMENTOS DE EVALUACIÓN

TÉCNICAS: <i>Actividades Resueltas</i> <i>Ejercicios Propuestos</i>	INSTRUMENTOS: <i>Rubricas</i>
----------------------------------------------------------------------------------	-----------------------------------------

	<p align="center">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p align="center">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLD-001</p>	<p>Página: 14</p>

CRITERIOS DE EVALUACIÓN

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