



| | | |
|--|--|---|
|  | <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 | | HORA DE PRESENTACIÓN | | | |
| INTEGRANTE (s): Yauli Merma Diego Raul | | | | NOTA: | |
| DOCENTE(s): Mg. Ing. Rene Alonso Nieto Valencia. | | | | | |

| SOLUCIÓN Y RESULTADOS |
|---|
| <p>LINK DE GITHUB:</p> <p>https://github.com/diegoyauli/LAB-EDA-DiegoYauli.git</p> |
| <p>I. SOLUCIÓN DE EJERCICIOS/PROBLEMAS</p> <p>Ejercicios resueltos:</p> <p>1. ArrayList:</p> <pre style="background-color: #2e3436; color: #eeeeec; padding: 10px; font-family: monospace;"> 2 import java.util.ArrayList; 3 public class Resuelto1 { 4 5 Run Debug Run main Debug main 6 public static void main(String[] args) { 7 ArrayList<String> alumnos = new ArrayList<String>(); 8 ArrayList<Integer> notas = new ArrayList<Integer>(); 9 alumnos.add(e:"MARIA"); 10 alumnos.add(e:"DIEGO"); 11 alumnos.add(e:"RENE"); 12 alumnos.add(e:"ALONSO"); 13 System.out.println(alumnos.hashCode()); 14 System.out.println(alumnos.isEmpty()); 15 System.out.println(alumnos.size()); 16 } </pre> |

| | | |
|--|--|---|
|  | <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: 2</p> |

Resultados:

```
PS D:\UNSA\5º Semestre\EDA\Laboratoriooss\LAB-EDA-DiegoYauli>
.0.2\bin\java.exe' '-cp' 'C:\Users\Usuario\AppData\Roaming\Cod
d11\bin' 'Laboratorio3.Resuelto1'
1121431919
false
4
```

Explicacion:

Crea dos listas: una para nombres y otra para notas.

Llena la lista de nombres con 4 alumnos.



Luego imprime tres cosas sobre esa lista:

- Su código hash.
- Si está vacía.
- Su tamaño (cantidad de elementos).

2. Iterador:

```
2  import java.util.ArrayList;
3  import java.util.Iterator;
4
5  public class Resuelto2 {
6      Run | Debug | Run main | Debug main
7      public static void main(String[] args){
8          ArrayList<String> alumnos = new ArrayList<String>();
9          alumnos.add(e:"MARIA");
10         alumnos.add(e:"DIEGO");
11         alumnos.add(e:"RENE");
12         alumnos.add(e:"ALONSO");
13         Iterator<String> itA = alumnos.iterator();
14         while (itA.hasNext()) {
15             System.out.println(itA.next());
16         }
17     }
```

Resultado:

| | | |
|--|--|---|
|  | <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: 3</p> |

```
PS D:\UNSA\5° Semestre\EDA\Laboratorioooss\LAB-EDA-DiegoYauli>
Files\Java\jdk-13.0.2\bin\java.exe' '-cp' 'C:\Users\Usuario\AppData\Local\Temp\jdt_ws\LAB-EDA-DiegoYauli_17c9cd11\bin' 'Laboratorio3.Resue
MARIA
DIEGO
RENE
ALONSO
```

Explicación:

- El Iterator permite recorrer la lista alumnos paso a paso.
- Con hasNext() se verifica si hay más elementos.
- Con next() se obtiene cada nombre y se imprime.
- Es una forma segura y ordenada de recorrer una colección en Java.

3. Clase Animal en Java:

```
3 import java.util.ArrayList;
4 import java.util.List;
5 public class Animal {
6     String nombre;
7     boolean genero;
8     public Animal(String nombre, boolean genero) {
9         this.nombre = nombre;
10        this.genero = genero;
11    }
12    public String getNombre() {
13        return nombre;
14    }
15    public void setNombre(String nombre) {
16        this.nombre = nombre;
17    }
18    public boolean isGenero() {
19        return genero;
20    }
21    public void setGenero(boolean genero) {
22        this.genero = genero;
23    }
24    Run | Debug | Run main | Debug main
25    public static void main(String[] args) {
26        ArrayList<Animal> mascotas = new ArrayList<Animal>();
27        // List<Animal> mascotas2 = new List<Animal>(); //No se puede instanciar directamente a una interfaz
28        List<Animal> mascotas3 = new ArrayList<Animal>();
29        mascotas.add(new Animal(nombre:"Toby", genero:true)); // ArrayList
30        System.out.println("ArrayList -> " + mascotas.get(index:0).getNombre());
31
32        mascotas3.add(new Animal(nombre:"Luna", genero:false)); // List (interfaz)
33        System.out.println("List -> " + mascotas3.get(index:0).getNombre());
34    }
35 }
```

| | | |
|--|--|---|
|  | <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: 4</p> |

Resultado:

```
PS D:\UNSA\5º Semestre\EDA\Laboratoriooss\LAB-EDA-DiegoYauli> d:; c
Files\Java\jdk-13.0.2\bin\java.exe' '-cp' 'C:\Users\Usuario\AppData\
va\jdt_ws\LAB-EDA-DiegoYauli_17c9cd11\bin' 'Laboratorio3.Animal'
ArrayList -> Toby
List -> Luna
```

Explicación:

Se definió la clase Animal con dos atributos: nombre (cadena de texto) y genero (booleano).

En el método main, se instanciaron dos listas:

- mascotas con ArrayList, usando la clase directamente.
- mascotas3 como List<Animal> pero asignándole una instancia de ArrayList (buena práctica de programación orientada a interfaces).

Se agregaron objetos a ambas listas y se imprimieron los nombres usando get(0).

Ejercicios propuestos:

Ejercicios propuestos:

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

Código:

Node.java

```
4 public class Node<T> {
5     T data; // Datos almacenados en el nodo
6     Node<T> nextNode; // Referencia al siguiente nodo
7     // Constructor con referencia al siguiente nodo
8     public Node(T data, Node<T> nextNode) {
9         this.data = data;
10        this.nextNode = nextNode;
11    }
12    // Constructor sin siguiente nodo (último)
13    public Node(T data) {
14        this(data, nextNode:null);
15    }
16    // Representación en texto del nodo
17    public String toString() {
18        return data.toString();
19    }
20 }
```

List

```
3 // Clase genérica para implementar una lista enlazada simple
4 public class List<T> {
5     private Node<T> root; // Nodo inicial de la lista
6     // Constructor: crea lista vacía
7     public List() {
8         this.root = null;
9     }
10    // Verifica si la lista está vacía
11    public boolean isEmpty() {
12        return root == null;
13    }
14    // Devuelve el tamaño (número de nodos)
15    public int size() {
16        int count = 0;
17        Node<T> current = root;
18        while (current != null) {
```

```
18         while (current != null) {
19             count++;
20             current = current.nextNode;
21         }
22         return count;
23     }
24     // Inserta un nuevo nodo al inicio
25     public void addFirst(T data) {
26         root = new Node<>(data, root);
27     }
28     // Inserta un nuevo nodo al final
29     public void addLast(T data) {
30         if (isEmpty()) {
31             addFirst(data);
32         } else {
33             Node<T> current = root;
34             while (current.nextNode != null) {
35                 current = current.nextNode;
36             }
37             current.nextNode = new Node<>(data);
38         }
39     }
40     // Busca si un valor existe en la lista
41     public boolean contains(T data) {
42         Node<T> current = root;
43         while (current != null) {
44             if (current.data.equals(data)) return true;
45             current = current.nextNode;
46         }
47         return false;
48     }
```

```
49     // Elimina la primera aparición de un valor
50     public void remove(T data) {
51         if (isEmpty()) return;
52
53         // Si el valor está en el primer nodo
54         if (root.data.equals(data)) {
55             root = root.nextNode;
56             return;
57         }
58         Node<T> prev = root;
59         Node<T> current = root.nextNode;
60         // Buscar el nodo que contiene el valor
61         while (current != null && !current.data.equals(data)) {
62             prev = current;
63             current = current.nextNode;
```

```
64     }
65     // Si se encontró el valor, eliminar el nodo
66     if (current != null) {
67         prev.nextNode = current.nextNode;
68     }
69 }
70 // Representación de la lista como texto
71 public String toString() {
72     StringBuilder sb = new StringBuilder(str:"[");
73     Node<T> current = root;
74     while (current != null) {
75         sb.append(current.data);
76         if (current.nextNode != null) sb.append(str:", ");
77         current = current.nextNode;
78     }
79     sb.append(str:"]");
80     return sb.toString();
81 }
82 }
```

Main

```
3 public class Main {
4     Run | Debug | Run main | Debug main
5     public static void main(String[] args) {
6         List<Integer> myList = new List<>();
7         System.out.println("¿Está vacía?: " + myList.isEmpty());
8         myList.addLast(data:10);
9         myList.addLast(data:20);
10        myList.addFirst(data:5);
11        System.out.println("Lista actual: " + myList);
12        System.out.println("Contiene 20?: " + myList.contains(data:20));
13        System.out.println("Contiene 100?: " + myList.contains(data:100));
14        myList.remove(data:10);
15        System.out.println("Después de eliminar 10: " + myList);
16        System.out.println("Tamaño: " + myList.size());
17    }
```

Resultados:

```
PS D:\UNSA\5º Semestre\EDA\Laboratorioos\LAB-EDA-DiegoYauli> d:; cd ..\
Files\Java\jdk-13.0.2\bin\java.exe' '-cp' 'C:\Users\Usuario\AppData\Roa
va\jdt_ws\LAB-EDA-DiegoYauli_17c9cd11\bin' 'Laboratorio3.Propuesto1.Ma
¿Está vacía?: true
Lista actual: [5, 10, 20]
Contiene 20?: true
Contiene 100?: false
Después de eliminar 10: [5, 20]
Tamaño: 2
```

Explicacion:

1. Clase Node<T>

Representa un nodo individual dentro de la lista enlazada.

Tiene dos atributos fundamentales:

data: almacena el valor o dato de tipo genérico T.

nextNode: referencia al siguiente nodo en la lista, permitiendo que los nodos estén enlazados entre sí.

Posee dos constructores:

Uno que recibe el dato y una referencia al siguiente nodo.

Otro que recibe solo el dato, asumiendo que es el último nodo (con nextNode = null).

Implementa el método toString() para representar el contenido del nodo en forma de texto, lo cual facilita la impresión de la lista.

2. Clase List<T>

Implementa la estructura lista enlazada simple usando nodos de tipo genérico Node<T>.

Cuenta con la propiedad root que apunta al primer nodo de la lista. Si root es null, la lista está vacía.

Métodos clave:

isEmpty(): verifica si la lista está vacía (cuando root es null).

size(): recorre todos los nodos para contar cuántos elementos hay en la lista.

| | | |
|---|--|---|
|  | <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> |

addFirst(T data): inserta un nodo al inicio de la lista, actualizando el root para que apunte a este nuevo nodo.

addLast(T data): agrega un nodo al final. Si la lista está vacía, usa addFirst. De lo contrario, recorre hasta el último nodo y enlaza el nuevo nodo allí.

contains(T data): recorre la lista para determinar si un dato específico está presente.

remove(T data): elimina la primera aparición de un nodo que contenga el dato. Se maneja el caso en que el nodo a eliminar es el primero, y el caso en que está en medio o al final, ajustando correctamente los enlaces para mantener la lista consistente.

toString(): construye una representación textual de toda la lista, mostrando los datos en orden y separados por comas, encerrados entre corchetes.

3. Clase Main

Contiene el método main para realizar pruebas sobre la lista enlazada.

Crea una lista de enteros y realiza operaciones básicas:

Verifica si la lista está vacía.

Inserta elementos tanto al inicio como al final.

Imprime la lista para mostrar su contenido actual.

Busca elementos para comprobar si existen o no.

Elimina un elemento y vuelve a imprimir la lista para mostrar los cambios.

Muestra el tamaño actual de la lista.

Esto permite validar que todos los métodos funcionan correctamente y que la lista se comporta según lo esperado.

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).

-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.

| | | |
|--|--|---|
|  | <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> |

-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.



Codigo:

Operador

```

3  public class Operador<T extends Number> {
4      private T valor1;
5      private T valor2;
6
7      public Operador(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 }

```

| | | |
|--|--|---|
|  | <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: 11</p> |

Calculadora:

```

3 public class Calculadora {
4
5     public static <T extends Number> double suma(Operador<T> op) {
6         return op.getValor1().doubleValue() + op.getValor2().doubleValue();
7     }
8
9     public static <T extends Number> double resta(Operador<T> op) {
10        return op.getValor1().doubleValue() - op.getValor2().doubleValue();
11    }
12
13    public static <T extends Number> double producto(Operador<T> op) {
14        return op.getValor1().doubleValue() * op.getValor2().doubleValue();
15    }
16
17    public static <T extends Number> double division(Operador<T> op) {
18        if (op.getValor2().doubleValue() == 0)
19            throw new ArithmeticException(s:"No se puede dividir entre 0.");
20        return op.getValor1().doubleValue() / op.getValor2().doubleValue();
21    }
22
23    public static <T extends Number> double potencia(Operador<T> op) {
24        return Math.pow(op.getValor1().doubleValue(), op.getValor2().doubleValue());
25    }
26
27    public static <T extends Number> double raizCuadrada(T valor) {
28        return Math.sqrt(valor.doubleValue());
29    }
30
31    public static <T extends Number> double raizCubica(T valor) {
32        return Math.cbrt(valor.doubleValue());
33    }
34 }

```


Main:

```

3 import java.util.Scanner;
4
5 public class Main {
6     Run | Debug | Run main | Debug main
7     public static void main(String[] args) {
8         Scanner sc = new Scanner(System.in);
9         boolean salir = false;
10
11         while (!salir) {
12             System.out.println(x:"\nMenú de Operaciones Clases Genéricas:");
13             System.out.println(x:"1. Suma");
14             System.out.println(x:"2. Resta");
15             System.out.println(x:"3. Producto");
16             System.out.println(x:"4. División");
17             System.out.println(x:"5. Potencia");
18             System.out.println(x:"6. Raíz Cuadrada");
19             System.out.println(x:"7. Raíz Cubica");
20             System.out.println(x:"8. Salir");
21             int opcion = Integer.parseInt(sc.nextLine());
22             switch (opcion) {
23                 case 1:
24                     suma(sc);
25                     break;
26                 case 2:
27                     resta(sc);
28                     break;
29                 case 3:
30                     producto(sc);
31                     break;
32                 case 4:
33                     division(sc);
34                     break;
35                 case 5:
36                     potencia(sc);
37                     break;
38                 case 6:
39                     raizCuadrada(sc);
40                     break;
41                 case 7:
42                     raizCubica(sc);
43                     break;
44                 case 8:
45                     salir = true;
46                     break;
47                 default:
48                     System.out.println(x:"Opción no válida");
49             }
50         }
51     }
52 }

```

```
18 System.out.println(x:"7. Raíz Cúbica");
19 System.out.println(x:"8. Salir del Programa");
20 System.out.print(s:"Seleccione una opción: ");
21 int opcion = sc.nextInt();
22
23 if (opcion >= 1 && opcion <= 5) {
24     System.out.print(s:"¿Tipo de dato? (int/double): ");
25     String tipo = sc.next();
26     System.out.print(s:"Ingrese el primer valor: ");
27     double v1 = sc.nextDouble();
28     System.out.print(s:"Ingrese el segundo valor: ");
29     double v2 = sc.nextDouble();
30
31     if (tipo.equalsIgnoreCase(anotherString:"int")) {
32         Operador<Integer> op = new Operador<>((int)v1, (int)v2);
33         ejecutarOperacion(opcion, op);
34     } else {
35         Operador<Double> op = new Operador<>(v1, v2);
36         ejecutarOperacion(opcion, op);
37     }
38
39     } else if (opcion == 6 || opcion == 7) {
40         System.out.print(s:"¿Tipo de dato? (int/double): ");
41         String tipo = sc.next();
42         System.out.print(s:"Ingrese el valor: ");
43         double v = sc.nextDouble();
44
45         if (tipo.equalsIgnoreCase(anotherString:"int")) {
46             if (opcion == 6)
47                 System.out.println("Raíz Cuadrada: " + Calculadora.raizCuadrada((int)v));
48             else
49                 System.out.println("Raíz Cúbica: " + Calculadora.raizCubica((int)v));
50         } else {
51             if (opcion == 6)
52                 System.out.println("Raíz Cuadrada: " + Calculadora.raizCuadrada(v));
53             else
54                 System.out.println("Raíz Cúbica: " + Calculadora.raizCubica(v));
55         }
56
57     } else if (opcion == 8) {
58         salir = true;
59         System.out.println(x:"Saliendo del programa...");
60     } else {
61         System.out.println(x:"Opción inválida.");
62     }
63 }
64 sc.close();
65 }
66
67 // Método auxiliar para ejecutar operaciones básicas
68 public static <T extends Number> void ejecutarOperacion(int opcion, Operador<T> op) {
69     switch (opcion) {
70         case 1:
```

| | | |
|---|--|---|
|  | <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: 13</p> |

```

71         System.out.println("Resultado de la suma: " + Calculadora.suma(op));
72         break;
73     case 2:
74         System.out.println("Resultado de la resta: " + Calculadora.resta(op));
75         break;
76     case 3:
77         System.out.println("Resultado del producto: " + Calculadora.producto(op));
78         break;
79     case 4:
80         try {
81             System.out.println("Resultado de la división: " + Calculadora.division(op));
82         } catch (ArithmeticException e) {
83             System.out.println("Error: " + e.getMessage());
84         }
85         break;
86     case 5:
87         System.out.println("Resultado de la potencia: " + Calculadora.potencia(op));
88         break;
89     }
90 }
91 }

```

Resultado:

```

PS D:\UNSA\5º Semestre\EDA\Laboratoriooss\LAB-EDA-DiegoYauli> &
Roaming\Code\User\workspaceStorage\5b3bd710e35cd683adc03f0079f3e
Main'

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
¿Tipo de dato? (int/double): int
Ingrese el primer valor: 4
Ingrese el segundo valor: 5
Resultado del producto: 20.0

```

| | | |
|---|--|---|
|  | <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: 14</p> |

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: 5
¿Tipo de dato? (int/double): int
Ingrese el primer valor: 4
Ingrese el segundo valor: 2
Resultado de la potencia: 16.0
```

Explicación:

- La clase Operador<T> permite trabajar con diferentes tipos de datos numéricos.

Con el uso de <T extends Number>, se asegura que solo se puedan usar tipos numéricos como Integer, Double, entre otros.

Tiene dos atributos: valor1 y valor2, que representan los números a operar.

Su constructor permite inicializar estos valores al crear un nuevo objeto.

Esta clase es flexible porque se puede reutilizar con distintos tipos numéricos sin repetir código.

Es una forma práctica de representar valores numéricos con generics en Java.

- La clase Calculadora implementa los métodos para realizar operaciones matemáticas básicas y avanzadas.

Con suma(), resta(), producto() y division(), se hacen operaciones aritméticas entre dos valores.

Con potencia(), raizCuadrada() y raizCubica(), se realizan operaciones más complejas.

Todos los métodos son genéricos y estáticos, lo que permite llamarlos sin necesidad de crear una instancia.

Internamente convierten los valores a double para obtener resultados más precisos.

Incluyen validaciones, como evitar dividir entre cero.

| | | |
|---|--|---|
|  | <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: 15</p> |

Es una forma segura y reutilizable de aplicar operaciones matemáticas usando programación genérica.

- La clase Main contiene el menú de opciones para interactuar con el usuario.

Con Scanner, se permite al usuario elegir entre trabajar con Integer o Double, ingresar los datos y seleccionar una operación del menú.

El menú muestra 7 operaciones disponibles más la opción de salir del programa.

Con un switch, se ejecuta el método correspondiente según la opción elegida.

El menú se repite en un bucle while hasta que se elige la opción 8 (salir del programa).

Es una forma ordenada, clara y dinámica de probar el funcionamiento de las clases y métodos genéricos.

II. SOLUCIÓN DEL CUESTIONARIO

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

La principal dificultad fue entender y aplicar correctamente los genéricos en Java, especialmente para hacer operaciones matemáticas con distintos tipos numéricos. También fue un reto validar entradas y separar el código en clases ordenadas.

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

List es una interfaz que define el comportamiento de una lista, mientras que ArrayList es una clase concreta que implementa esa interfaz usando un arreglo dinámico para almacenar elementos.

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

Las clases genéricas permiten reutilizar código con distintos tipos, mejorar la seguridad de tipos en tiempo de compilación y facilitar el mantenimiento y la flexibilidad del software.

III. CONCLUSIONES

El uso de **clases genéricas en Java** facilita la creación de estructuras y métodos que pueden trabajar con diferentes tipos de datos, aumentando la reutilización y la flexibilidad del código.

Implementar una **lista genérica** permite manejar colecciones de objetos de manera segura y eficiente, adaptándose a distintos tipos sin necesidad de duplicar código.

La creación de una **calculadora genérica** demuestra cómo aplicar genéricos para realizar operaciones matemáticas con distintos tipos numéricos, garantizando precisión y seguridad en la ejecución.

| | | |
|---|---|---|
|  | <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: 16</p> |

El manejo adecuado de validaciones y la estructuración del código en clases claras mejora la legibilidad, el mantenimiento y la escalabilidad del programa.

Trabajar con genéricos es un reto inicial, pero aporta grandes beneficios a largo plazo, especialmente en proyectos que requieren adaptabilidad y robustez.

RETROALIMENTACIÓN GENERAL

REFERENCIAS Y BIBLIOGRAFÍA

- 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
- Malik D., *Data Structures Usign C++*, 2003, Thomson Learning.
- Knuth D., *The Art of Computer Programming*, Vol. 1 y 3, Addison - Wesley.