
	<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-PRLE-001	Página: 1

INFORME DE LABORATORIO

(formato estudiante)

INFORMACIÓN BÁSICA					
ASIGNATURA:	ESTRUCTURA DE DATOS Y ALGORITMOS				
TÍTULO DE LA PRÁCTICA:					
NÚMERO DE PRÁCTICA:	2	AÑO LECTIVO:	2025 – A	NRO. SEMESTRE:	Tercero III
FECHA DE PRESENTACIÓN	17/05/2025	HORA DE PRESENTACIÓN			
INTEGRANTE (s): Santa Cruz Villa Diego Sebastian				NOTA:	
DOCENTE(s): <ul style="list-style-type: none"> Mg. Ing. Rene Alonso Nieto Valencia. 					

REPOSTIORIO GITHUB: <https://github.com/DiegoSantaC/LaboratoriosEDA>

SOLUCIÓN Y RESULTADOS	
I. SOLUCIÓN DE EJERCICIOS/PROBLEMAS	
II. EJERCICIOS RESUELTOS	
<ul style="list-style-type: none"> 1.- Ejercicio 1: Implementación de un método recursivo. 	
 <pre> 5 public class EjerciciosResueltos { 6 static class Recursividad { 7 void repetir() { 8 System.out.println("Hola"); 9 repetir(); 10 } 11 } 12 public static void main(String[] ar) { 13 Recursividad re = new Recursividad(); 14 re.repetir(); 15 } 16 } </pre> <p>Hola Hola Hola Hola ueltos.java:9) at com.mycompany.Laboratorio2EDA.EjerciciosResueltos\$Recursividad.repetir(EjerciciosResueltos.java:9) at com.mycompany.Laboratorio2EDA.EjerciciosResueltos\$Recursividad.repetir(EjerciciosResueltos.java:9) at com.mycompany.Laboratorio2EDA.EjerciciosResueltos\$Recursividad.repetir(EjerciciosResueltos.java:9) at com.mycompany.Laboratorio2EDA.EjerciciosResueltos\$Recursividad.repetir(EjerciciosResueltos.java:9) at com.mycompany.Laboratorio2EDA.EjerciciosResueltos\$Recursividad.repetir(EjerciciosResueltos.java:9)</p>	
En este ejercicio se implementa una clase Recursividad con un método repetir el cual por cada llamado imprime la palabra Hola indicando que se ejecutó la función, además de volver a llamar el mismo método.	

```

18 static class Recursividad3 {
19     void imprimir(int n) {
20         if(n>0) {
21             System.out.println(n);
22             imprimir(n-1);
23         }
24     }
25 }
26
27 public static void main(String[] ar) {
28     Recursividad3 re = new Recursividad3();
29     re.imprimir(5);
30 }

```

```

--- exec:3.1.0:exec (default-cli) @ Laboratorio1EDA ---
5
4
3
2
1
-----
BUILD SUCCESS
-----

```

Siguiendo el ejercicio 2, esta vez ponemos un limite para parar el llamado de la función usando el condicional `if(n>0)` indicándonos que cuando nuestro `n` llegue a 0 el `if` no será ejecutado y como el llamado de nuestra función se encuentra dentro la recursividad parada

- 4.- Ejercicio 4: Imprimir los números de 1 a 5 en pantalla utilizando recursividad.

```

26 static class Recursividad4 {
27     void imprimir(int n) {
28         if(n>0) {
29             imprimir(n-1);
30             System.out.println(n);
31         }
32     }
33 }
34
35 public static void main(String[] ar) {
36     Recursividad4 re = new Recursividad4();
37     re.imprimir(5);
38 }
--- exec:3.1.0:exec (default-cli) @ Laboratorio1EDA ---
1
2
3
4
5
-----

```

Al igual que el ejercicio anterior, pero esta vez cambiaremos el orden de la ejecución de la recursividad y la imprimida del valor actual de `n`, esto ocasionada que la recursividad valla desde dentro hacia fuera, en este caso la recursividad ira llamándose sin imprimir nada hasta encontrar el caso base que seria nuestro `n=0` y luego de encontrar su fin recién empezara a ejecutar la segunda línea de impresión de `n` haciendo que se imprima desde el menor valor hasta el `n` actual.

- 5.- Ejercicio 5: Obtener el factorial de un número. Recordar que el factorial de un número es el resultado que se obtiene de multiplicar dicho número por el anterior y así sucesivamente hasta llegar a uno. Ej. el factorial de 4 es $4 * 3 * 2 * 1$ es decir 24.

```

35 static class Recursividad5 {
36     int factorial(int n) {
37         if(n==1)
38             return 1;
39         else
40             return n*factorial(n-1);
41     }
42 }
43
44 public static void main(String[] ar) {
45     Recursividad5 re = new Recursividad5();
46     int n=8;
47     System.out.print("El facotrial de: "+n+" es: "+re.factorial(n));
48 }

```

--- exec:3.1.0:exec (default-cli) @ LaboratoriolEDA ---

El facotrial de: 8 es: 40320

BUILD SUCCESS

En este ejercicio implementamos la lógica de factorial, pero de manera recursiva poniendo un caso base que será donde la recursividad pare que es en $n=1$ para que retorne 1 cuando la función factorial llegue a este número en su argumento, luego incluimos la recursividad que sería que la función retorne el número actual del argumento multiplicado por la función con argumento restado 1, así seguirá hasta llegar al caso base y ya no se ejecute la función sino que retornada el 1 del caso base.

• **6.- Ejercicio 6. Implementar un método recursivo para ordenar los elementos de un vector.**

```

44 static class Recursividad6 {
45     void ordenar(int[] v,int cantidad) {
46         if(cantidad > 1){
47             for(int f=0;f<cantidad-1;f++){
48                 if (v[f] > v[f + 1]) {
49                     int aux = v[f];
50                     v[f] = v[f + 1];
51                     v[f + 1] = aux;
52                 }
53                 ordenar(v,cantidad-1);
54             }
55         }
56     }
57     void imprimir(int[] vec) {
58         for (int f = 0; f < vec.length; f++)
59             System.out.print(vec[f] + " ");
60         System.out.println("\n");
61     }
62 }
63
64 public static void main(String[] ar) {
65     int[] vec = { 312, 614, 88, 22, 54 };
66     Recursividad6 re = new Recursividad6();
67     re.imprimir(vec);
68     re.ordenar(vec,5);
69     re.imprimir(vec);
70 }

```

```

--- exec:3.1.0:exec (default-cli) @ Laboratorio1EDA ---
312 614 88 22 54

22 54 88 312 614

```

Para este ejercicio creamos una función ordenar que tenga de parámetros un arreglo que ordenaremos y una cantidad que será hasta que posición del arreglo queremos ordenar, Tomaremos como caso base cuando la cantidad a ordenar sea llegue a 1, esto debido a que iremos ordenando por pares en el arreglo y tomar la cantidad de 1 sería imposible compararlo con otro, luego incluimos un for que será el que busque el mayor elemento hasta la cantidad indicada menos 1 ($f < \text{cantidad} - 1$) comparando par por par quien es mayor hasta que nuestro elemento mayor quede en la última posición. Como ya sabemos que nuestro elemento mayor es el que está en la última posición no será necesario volverlo a comparar así que volveremos a llamar a la función para que busque el siguiente mayor, pero sin contar el que ya encontramos así hasta que la cantidad sea de un solo elemento y pare la recursividad indicándonos que el arreglo ya está ordenado.

III. EJERCICIOS PROPUESTOS

- 1.- Invertir vector de enteros, permite ingresar tamaño y captura de valores del arreglo, el método `invertirArray` calcula y muestra el resultado.

```

5 public class EjercicioP1 {
6     public static void main(String[] ar) {
7         int[] vec1 = {1,2,3,4,5};
8         Recursividad1 r=new Recursividad1();
9         r.imprimir(vec1);
10        int[] vec2 = r.invertirArray(vec1);
11        r.imprimir(vec2);
12    }
13
14    static class Recursividad1{
15        int cantidad=0;
16        int[] B;
17
18        public int[] invertirArray(int[] A){
19            if(cantidad==0)
20                B=Arrays.copyOf(A, A.length);
21
22            if(cantidad<B.length/2){
23                int temp=B[cantidad];
24                B[cantidad]=B[B.length-1-cantidad];
25                B[B.length-1-cantidad]=temp;
26                cantidad++;
27                invertirArray(B);
28            }
29            return B;
30        }
31
32        void imprimir(int[] vec) {
33            for (int f = 0; f < vec.length; f++)
34                System.out.print(vec[f] + " ");
35            System.out.println("\n");
36        }
37    }

```

```

--- exec:3.1.0:exec (default-cli) @ Laboratorio1EDA ---
1 2 3 4 5

5 4 3 2 1
|
-----

```

Se define una clase que invierte un arreglo de enteros de forma recursiva, primero imprime el arreglo original, luego lo invierte intercambiando elementos desde los extremos hacia el centro usando recursividad, y finalmente imprime el arreglo invertido.

- 2.- Rotar a la Izquierda, permite ingresar tamaño y captura de valores del arreglo, el método `rotarIzquierdaArray` calcula y muestra el resultado.

```

5      public class EjercicioP2 {
6      public static void main(String[] ar) {
7          int[] vec1 = {1,2,3,4,5,6,7,8,9,10};
8          Recursividad1 r=new Recursividad1();
9          r.imprimir(vec1);
10         int[] vec2 = r.rotarIzquierdaArray(vec1,5);
11         r.imprimir(vec2);
12     }
13
14     static class Recursividad1{
15         int cantidad=0;
16         int[] B;
17
18         public int[] rotarIzquierdaArray (int[] A,int d){
19             if(cantidad==0)
20                 B=Arrays.copyOf(A, A.length);
21
22             if(cantidad < d){
23                 int temp=B[0];
24                 for(int i=0;i<B.length-1;i++){
25                     B[i]=B[i+1];
26                 }
27                 B[B.length-1]=temp;
28                 cantidad++;
29                 rotarIzquierdaArray(B,d);
30             }
31             return B;
32         }
33         void imprimir(int[] vec) {
34             for (int f = 0; f < vec.length; f++)
35                 System.out.print(vec[f] + " ");
36             System.out.println("\n");
37         }
38     }
39
40     --- exec:3.1.0:exec (default-cli) @ Laboratorio1EDA ---
41     1 2 3 4 5 6 7 8 9 10
42
43     6 7 8 9 10 1 2 3 4 5
44
45     -----

```

El código rota a la izquierda un arreglo de enteros `d` veces usando recursividad; primero imprime el arreglo original, luego realiza rotaciones hacia la izquierda una por una moviendo el primer elemento al final en cada llamada recursiva, y finalmente imprime el arreglo resultante.

• 3.- Triangulo recursivo 1.

```

4      public class EjercicioP3 {
5
6      public static void main(String[] ar) {
7          Recursividad1 r=new Recursividad1();
8          r.trianguloRecursivo1(5);
9      }
10
11     static class Recursividad1{
12
13     public void trianguloRecursivo1(int n){
14         if(n>=1){
15             trianguloRecursivo1(n-1);
16             for(int i=0;i<n;i++)
17                 System.out.print("*");
18             System.out.println();
19         }
20     }
21 }

```

--- exec:3.1.0:exec (default-cli) @ Laboratorio1EDA ---

```

*
**
***
****
*****

```

Se implementa el método recursivo y le definimos su caso base cuando n llegue a 1 para detener la recursividad, como los * están ordenados de menor a mayor usaremos primero el caso de recursividad dentro del if hasta que encuentre el menor caso que seria 1 y luego empezara a imprimir los * usando un for hasta el valor del n que tenga en el parámetro, como resultado nuestra recursividad ira de atrás hasta llegar a nuestro n actual

• 4.- Triangulo recursivo 2

```

4      public class EjercicioP4 {
5
6      public static void main(String[] ar) {
7          Recursividad1 r=new Recursividad1();
8          r.trianguloRecursivo2(10);
9      }
10
11     static class Recursividad1{
12         int fila=0;
13     public void trianguloRecursivo2(int n){
14         if(fila<=n){
15             for(int i=0;i<n-fila;i++)
16                 System.out.print(" ");
17             for(int j=0;j<fila;j++)
18                 System.out.print("*");
19             System.out.println();
20             fila++;
21             trianguloRecursivo2(n);
22         }
23     }

```

```

--- exec:3.1.0:exec (default-cli) @ Laboratorio1EDA ---

      *
     **
    ***
   ****
  *****
 
```

Se implementa un método recursivo que imprime un triángulo de asteriscos alineado a la derecha. El caso base se alcanza cuando la variable fila supera a n, deteniendo la recursividad. En cada llamada, se imprimen primero espacios para alinear y luego asteriscos según el valor actual de fila. La recursividad incrementa fila en cada llamada, por lo que el triángulo se construye desde arriba hacia abajo, comenzando con una línea de un asterisco hasta llegar a n asteriscos.

5.-Triangulo recursivo 3

```

4      public class EjercicioP5 {
5
6      public static void main(String[] ar) {
7          Recursividad1 r=new Recursividad1();
8          r.trianguloRecursivo3(10);
9      }
10
11     static class Recursividad1{
12         int fila=1;
13         int col=0;
14         public void trianguloRecursivo3(int n){
15             int anchoTotal=n*2-1;
16             if (fila<=n) {
17                 if (col >= anchoTotal) {
18                     System.out.println();
19                     fila++;
20                     col = 0;
21                     trianguloRecursivo3(n);
22                 } else {
23                     int inicio = (anchoTotal-(fila*2-1))/2;
24                     if (col<inicio) {
25                         System.out.print(" ");
26                     } else if ((col-inicio)%2==0 && (col-inicio)/2<fila) {
27                         System.out.print("*");
28                     } else {
29                         System.out.print(" ");
30                     }
31                     col++;
32                     trianguloRecursivo3(n);
33                 }
34             }
35         }
36     }

```

```

--- exec:3.1.0:exec (default-cli) @ Laboratorio1EDA ---

      *
     **
    ***
   ****
  *****
 
```


Se implementa un método recursivo para imprimir un triángulo centrado de asteriscos. Se utiliza una lógica de control con variables fila y col para manejar la posición actual en la impresión. El caso base se alcanza cuando fila supera a n. En cada llamada recursiva, se imprimen espacios o asteriscos dependiendo de la posición en la línea (col) y el ancho total del triángulo. Una vez que se completa una fila, se hace un salto de línea, se incrementa fila y se reinicia col, continuando recursivamente hasta formar el triángulo completo de altura n.



• 6.- Cuadrado recursivo

```
public class EjercicioP6 {  
  
    public static void main(String[] ar) {  
        Recursividad1 r=new Recursividad1();  
        r.cuadradoRecursivo(10);  
    }  
  
    static class Recursividad1{  
        int fila=0;  
        int col=0;  
        public void cuadradoRecursivo(int n){  
            if(fila<n){  
                if(col>=n){  
                    System.out.println();  
                    fila++;  
                    col=0;  
                    cuadradoRecursivo(n);  
                } else {  
                    if(fila==0||fila==n-1||col==0||col==n-1)  
                        System.out.print("*");  
                    else  
                        System.out.print(" ");  
                    col++;  
                    cuadradoRecursivo(n);  
                }  
            }  
        }  
    }  
}
```



```
--- exec:3.1.0:exec (default-cli) @ Laboratorio1EDA ---  
*****  
*   *  
*   *  
*   *  
*****  
-----
```

Se implementa un método recursivo para imprimir un cuadro hueco de asteriscos de tamaño n x n. Las variables fila y col controlan la posición actual. Si se completa una fila (col >= n), se imprime un salto de línea, se avanza a la siguiente fila y se reinicia la columna. En cada posición, se imprime un asterisco si está en los bordes del cuadro, y un espacio en caso contrario. La recursión continúa hasta que se imprimen todas las filas.

	<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: 10</p>

IV. SOLUCIÓN DEL CUESTIONARIO

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

Al desarrollar los ejercicios, una de las principales dificultades fue manejar correctamente la recursividad, especialmente en cuanto al control de variables globales como `fila` o `col`, que deben reiniciarse adecuadamente para evitar errores lógicos. Otra dificultad fue diseñar la lógica de impresión para estructuras visuales como los triángulos centrados, ya que requiere cálculos precisos de espacios y símbolos.

2. Diferencias entre algoritmos de secuencialidad, decisión e iteración.

- **Secuencialidad:** Son instrucciones que se ejecutan una tras otra en el orden en que fueron escritas, sin condiciones ni repeticiones. Es la base de todo algoritmo
- **Decisión (o condicionales):** Permiten que el programa tome diferentes caminos según se cumpla o no una condición (`if`, `else`, `switch`). Son esenciales para la toma de decisiones dentro del código
- **Iteración:** Son estructuras que repiten un bloque de código mientras se cumpla una condición. Esto puede hacerse con bucles (`for`, `while`) o mediante recursividad, como en los ejercicios desarrollados

3. Que son las clases y métodos genéricos



Las clases y métodos genéricos son un concepto un poco confuso al principio, son una forma de escribir código que puede funcionar con distintos tipos de datos sin tener que repetir todo. Por ejemplo, en lugar de hacer una clase que solo funcione con enteros o con cadenas, podemos usar los genéricos para que esa misma clase funcione con cualquier tipo. Algo como `T` en lugar de `int` o `String`. Me di cuenta de que esto hace que el código sea más flexible y más fácil de reutilizar.

V. CONCLUSIONES

Desarrollar estos ejercicios me permitió entender mejor la lógica detrás de la recursividad, el manejo de arreglos y la impresión de estructuras en consola. Aunque hubo dificultades, especialmente al controlar variables dentro de métodos recursivos, fue una buena forma de fortalecer mi lógica. También aprendí a diferenciar entre algoritmos secuenciales, condicionales e iterativos, y comprendí cómo los métodos y clases genéricos permiten escribir código más flexible y reutilizable. En general, fue una experiencia desafiante pero muy útil para seguir mejorando como programador.

RETROALIMENTACIÓN GENERAL

El uso de variables globales en métodos recursivos puede facilitar el control de ciertas posiciones o estados, como en los ejercicios de impresión de figuras, pero también puede causar errores si no se manejan con cuidado, ya que su valor persiste entre llamadas recursivas. Es importante tener claro cuándo y cómo se modifican para evitar comportamientos inesperados. En cuanto a la construcción de figuras con recursividad, es un excelente ejercicio para desarrollar la lógica, ya que obliga a pensar en el orden en que se ejecutan las llamadas y cómo se va armando la figura paso a paso. Aunque más complejo que usar bucles, usar recursividad

	<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: 11</p>

para este tipo de estructuras ayuda a entender a fondo el flujo del programa y mejora la capacidad de resolución de problemas.

REFERENCIAS Y BIBLIOGRAFÍA