



Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorios de docencia

Laboratorio de Computación Salas A y B

Profesor(a): René Adrián Dávila Pérez

Asignatura: Programación Orientada a Objetos

Grupo: 7

No de Práctica(s): 1 y 2

Integrante(s): 322044339

322094152

322114461

425093384

322150984

No. de brigada: 6

Semestre: 2026-1

Fecha de entrega: 3 de septiembre de 2025

Observaciones:

CALIFICACIÓN: _____

Índice

| | |
|-------------------------------------|----------|
| 1. Introducción | 2 |
| 2. Marco Teórico | 2 |
| 3. Desarrollo | 4 |
| 3.1. Main | 4 |
| 3.2. Factorial | 4 |
| 3.2.1. Prueba con 6 | 5 |
| 3.3. Serie de Fibonacci | 5 |
| 3.3.1. Prueba con 15 | 5 |
| 3.4. Conjetura de Collatz | 5 |
| 3.4.1. Prueba con 15 | 5 |
| 4. Resultados | 6 |
| 4.1. Factorial | 6 |
| 4.2. Serie de Fibonacci | 6 |
| 4.3. Conjetura de Collatz | 7 |
| 4.4. Opción no válida | 7 |
| 4.5. Salida | 8 |
| 5. Conclusiones | 8 |
| 6. Referencias | 9 |

1. Introducción

Cuando programamos, es común toparse con problemas, que aunque parezcan sencillos, terminan siendo desafiantes si los vemos desde el punto de vista de la eficiencia, claridad y aplicación de los conocimientos básicos como son la recursividad, iteración y manejo de estructuras de control. Con esto podemos identificar tres problemas que suelen tener que ver con lo anterior: el cálculo del factorial de un número, la impresión de la serie de Fibonacci y la conjetura de Collatz. Si bien son conceptos fáciles de entender, su mala aplicación puede llevar a problemas como resultados erróneos, bucles sin fin, uso masivo de recursos computacionales y falta de fragmentación del código para su mejor manejo (modulación). Esto es un problema, pues la poca estructura y modulación del código se presenta como una barrera a la comprensión del código, que tendrá consecuencias en problemas que sean muchos más complejos.

Resolver estos tres problemas nos da una ventaja: practicar criterios de diseño (separación por métodos, entradas/salidas claras) y comparar enfoques recursivos e iterativos cuando convenga. Nos obliga a validar datos, controlar errores y documentar el comportamiento del programa. Todo esto se traduce en código más legible y fácil de probar; habilidades importantes para la resolución de problemas más complejos y de forma más eficientes, mismas que se mejorarán para las siguientes prácticas.

Al desarrollar un menú en Java que resuelva de forma independiente los tres problemas y registrando evidencias de funcionamiento se espera: Aplicar correctamente recursividad, iteración y estructuras de control, establecer límites razonables y, cuando haga falta, estrategias de optimización, comparar resultados y, si es posible, tiempo de ejecución o conteo de llamadas y dejar una explicación clara de lo que hace cada módulo y de las decisiones de diseño tomadas.

2. Marco Teórico

Para poder realizar esta práctica se necesita conocer algunos conceptos

- Recursividad

Dentro del ámbito de la programación, se le conoce como recursividad a la idea de poder definir un proceso, o función, en términos de sí mismo a una técnica mediante la cual una función se llama a sí misma, esto con el objetivo de poder dividir un problema complejo, y de naturaleza repetitiva, en subproblemas más fáciles de resolver. [5]

Una función recursiva se puede volver en un proceso infinito si no se añade una condición que la detenga, a dicha condición se le conoce como caso(s) base, un subproblema en su forma más primitiva; por lo anterior podemos dividir una función recursiva en dos partes:

- El caso base:

Condición que detiene el proceso, el cual es a su vez un subproblema de naturaleza simple el cual podemos resolver.

- Parte recursiva:

Bloque de código donde la función se llama a sí misma. [1]

- Estructuras iterativas

Tipo de estructuras las cuales repetirán un proceso con ligeras o nulas variaciones entre cada iteración, son estructuras que sirven para poder representar procesos de naturaleza repetitiva de forma compacta, dichas estructuras necesitan de una condición para ser detenidas o entrarán en un proceso infinito, dicha condición queda explícita dentro de la declaración de la estructura.

- Estructura while

Es un tipo de estructura donde la condición no indica la cantidad de iteraciones que se llevarán a cabo, a diferencia de la estructura for, no indica un límite explícito de iteraciones

- Estructuras condicionales

Este tipo de estructuras toman decisiones que dependen de lo que haya ocurrido antes en el código y, en base a ello, ejecutan bloques de código diferentes según las decisiones tomadas.

- if-else

Esta estructura solo toma dos posibles decisiones "True." "False" dependiendo de la decisión se ejecutará el primer o segundo bloque de código, respectivamente

- switch-case

A diferencia de if-else, esta estructura puede escoger entre n decisiones, pues esta estructura, compara su entrada con los casos que puede ejecutar y en base a ello decide; si la opción ingresada no está dentro de alguno de los casos, la estructura no ejecutará nada y pasará a la siguiente línea de código. [4]

- Operadores

Los operadores dentro de cualquier lenguaje de programación, nos permiten trabajar con los datos dependiendo de su tipo, pues un mismo símbolo puede servir para realizar diferentes operaciones dependiendo los tipos de datos que estemos manipulando.

- Operador de concatenación

El símbolo "+" naturalmente es utilizado para sumar dos números, sin embargo, en programación, esto depende de con que tipo de datos estamos trabajando, si son enteros, o datos de punto flotante, su significado y función es la misma que en la aritmética clásica, pero, si trabajamos con

caracteres o cadenas de texto, este operador concatena dos cadenas o caracteres dependiendo el caso, es decir toma dos cadenas de texto y las une para formar una sola. [7]

3. Desarrollo

Para esta práctica fue necesario dividir el código en cuatro métodos, siendo un main y una función por cada uno de los tres ejercicios, el primero siendo el factorial de un número elegido por el usuario aplicando recursividad, el segundo generar la serie de Fibonacci hasta la cantidad de dígitos que desee el usuario, y por último, la conjetura de Collatz desde el número que ingrese el usuario.

Pero antes de siquiera definir la clase que lleva el nombre de **Menu**, es importante importar la biblioteca *java.util.Scanner*, con la cual podremos hacer uso de la clase Scanner permitiéndonos a través de un objeto recibir datos de entrada.

3.1. Main

En el método main ocurren tres cosas simples, pero a la vez esenciales para el funcionamiento del código, una de estas es la creación del objeto inp de la clase Scanner, con este se recolectarán los datos ingresados por el usuario, además se crearon dos variables de tipo int, option y n, la primera almacena la elección del usuario para mostrarle el ejercicio solicitado y la segunda interactúa directamente con los ejercicios pues almacena el número del que se desea ver el factorial o hasta que dígito de la secuencia quiere ver la serie de Fibonacci o donde iniciar la conjetura de Collatz

La segunda es la estructura do while combinada con if, en esta nos aseguramos que el menú se repita continuamente a menos que el usuario elija la opción "4.- Salir", finalizando el programa.

La tercera es la estructura condicional switch, en esta se distinguen los tres case de los ejercicios y un default que simplemente regresa al menú. En el case 1 n será el número al que se le calculará su factorial y se llamará a la función FactorialR con argumento n. En el case 2, n será igual a la cantidad de términos que se desean ver de la sucesión de Fibonacci, además se llamará a la función Fibonacci con argumento n. Por último, en el case 3, n será igual al número desde que iniciará la conjetura de Collatz y se llamará a la función Collatz con argumento n. Cabe mencionar que las tres funciones son estáticas, por lo que se permite que los métodos sean ejecutados sin necesidad de crear un objeto de la clase

3.2. Factorial

El factorial de un número es la multiplicación de este por sus anteriores enteros positivos hasta uno [3], por lo que puede ser calculado mediante un método recursivo.

El código de factorial recursivo es bastante simple, primero se determina si n es igual a 0, en ese caso simplemente retornará uno, pero si no lo es multiplicará a n por la misma función FactorialR, pero con el argumento de n-1 formando una pila

de recursividad, por lo que así seguirá hasta llegar al caso base en el que n es igual a cero, por lo que multiplicará hasta retornar el resultado de n factorial.

3.2.1. Prueba con 6

Como 6 es diferente de cero caerá en el caso recursivo por lo que retornaría $6 * \text{FactorialR}(5)$, a su vez 5 es diferente de cero por lo que retornaría $6 * 5 * \text{FactorialR}(4)$ y así hasta el caso base, siendo la salida el retorno de $6 * 5 * 4 * 3 * 2 * 1 * 1$ igual a 720.

3.3. Serie de Fibonacci

La serie de Fibonacci consiste en la generación de una secuencia partiendo desde el 0 y 1, en las que cada número es la suma de los dos anteriores [6], el enfoque esta vez es iterativo, partiendo desde el cero y parando en la n -ésima suma.

Para el código que respecta a Fibonacci optamos por la opción iterativa, en este se declara una variable para el anterior inicializada con cero, y otra de nombre fib inicializada con 1, mediante un for en el que i es igual a cero hasta n , se imprimirá fib para después asignarle su valor a una variable auxiliar, ahora a fib se le sumará su anterior y al anterior se le asignará el valor de la variable auxiliar.

3.3.1. Prueba con 15

Al definir que n es igual a 15, entonces se obtendrán 15 elementos de la serie de Fibonacci, por lo que la salida esperada debería de ser.

0->1->1->2->3->5->8->13->21->34->55->89->144->233->377

3.4. Conjetura de Collatz

La conjetura de Collatz nos dice que al comenzar con cualquier número entero positivo podremos llegar a uno si dividimos entre dos cuando sea par o lo multiplicamos por tres y sumamos uno cuando sea impar [2]. La forma de resolver este problema fue mediante un ciclo while y un if.

Para el método de Collatz primero prevenimos el bucle infinito diciendo que n es diferente de 1, por lo que al obtener un 1 habremos llegado a nuestra secuencia deseada. Ahora, si es que el número es par entonces a n se le asignará $n/2$, pero si es impar a n se le asignará $3n+1$, se imprimirá y se volverá a evaluar si es par o impar, así sucesivamente hasta que n sea igual a 1. Cuando el ciclo se rompe al final de la sucesión se imprime un uno y se da por terminada la secuencia

3.4.1. Prueba con 15

Para la conjetura de Collatz, si deseamos iniciar la conjetura con 15, entonces como salida se debería obtener

15->46->23->70->35->106->53->160->80->40->20->10->5->16->8->4->2->1

4. Resultados

A continuación se muestran los resultados de la ejecución de nuestro código y una breve explicación de los mismos.

4.1. Factorial

```
Bienvenido usuario
1.- Factorial de un número
2.- Serie de Fibonacci
3.- Conjetura de Collatz
4.- Salir
Por favor ingrese una de las opciones mencionadas
1
Ingrese el número del que desea conocer el factorial:
6
El factorial de 6 es: 720
```

Figura 1: Menú inicial y factorial de 6

Se muestra el menú inicial tan pronto el programa es ejecutado, se solicita una opción, se ingresa la opción 1, seguido de esto se solicita un número para trabajar, y una vez ingresado dicho número, se calcula y se muestra su factorial.

4.2. Serie de Fibonacci

```
1.- Factorial de un número
2.- Serie de Fibonacci
3.- Conjetura de Collatz
4.- Salir
Por favor ingrese una de las opciones mencionadas
2
Ingrese el número de terminos de la sucesión de Fibonacci:
15
La sucesión de Fibonacci hasta 15 es:
0->1->1->2->3->5->8->13->21->34->55->89->144->233->377->Fin.
```

Figura 2: Serie de Fibonacci hasta la posición 15

El programa regresa al menú, pero en esta ocasión se selecciona la opción 2, se ingresa la posición de la serie que se quieren observar y se imprimen todos los dígitos de la serie de Fibonacci hasta la posición elegida.

4.3. Conjetura de Collatz

```
1.- Factorial de un número
2.- Serie de Fibonacci
3.- Conjetura de Collatz
4.- Salir
Por favor ingrese una de las opciones mencionadas
3
Ingrese el número desde que iniciara la sucesión:
15
La sucesión generada es la siguiente:
15->46->23->70->35->106->53->160->80->40->20->10->5->16->8->4->2->1
```

Figura 3: Conjetura de Collatz con el número 15

El menú se muestra nuevamente, pero en esta ocasión se selecciona la opción 3, se ingresa un número con el cual se pondrá a prueba la conjetura de Collatz y se imprimen todos los pasos de la conjetura de Collatz hasta que se llega a "1".

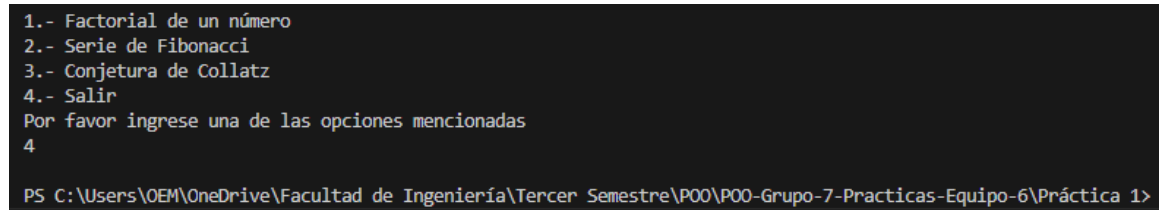
4.4. Opción no válida

```
Por favor ingrese una de las opciones mencionadas
23
Opcion no valida
1.- Factorial de un número
2.- Serie de Fibonacci
3.- Conjetura de Collatz
4.- Salir
Por favor ingrese una de las opciones mencionadas
█
```

Figura 4: Entrada no válida

Se muestra el menú y se solicita que se ingrese una opción para comenzar el programa, sin embargo, se ingresa una opción que no viene en las opciones, por consiguiente se muestra un mensaje de error y se solicita de nuevo que se ingrese una opción.

4.5. Salida



```
1.- Factorial de un número
2.- Serie de Fibonacci
3.- Conjetura de Collatz
4.- Salir
Por favor ingrese una de las opciones mencionadas
4
PS C:\Users\OEM\OneDrive\Facultad de Ingeniería\Tercer Semestre\P00\P00-Grupo-7-Practicas-Equipo-6\Práctica 1>
```

Figura 5: Salida

El programa regresa al menú, pero en esta ocasión se selecciona la opción 4, por lo que el programa se termina.

5. Conclusiones

El desarrollo de esta práctica nos permitió fortalecer conceptos fundamentales en la programación como la iteración, la recursividad y el empleo de estructuras de control, implementándolos para solucionar tres problemas: el factorial de un número, la conjetura de Collatz y la serie de Fibonacci. A pesar de que cada algoritmo parecía simple, su elaboración demostró lo relevante que es tener en cuenta la claridad y eficiencia al programar y modular el código.

Entre los principales aprendizajes de esta práctica se pueden destacar:

- La importancia de la **claridad y eficiencia** en la programación para evitar y manejar errores, así como mejorar el rendimiento de los programas.
- El uso de **iteración y recursividad** como herramientas para resolver problemas, eligiendo el enfoque más adecuado en cada caso.
- La relevancia de la **modularidad** para organizar el código y facilitar su comprensión.
- La práctica de **control de errores y registro de resultados** para lograr programas confiables.

En general, esta práctica generó un conocimiento muy importante en nuestras habilidades para programar, ya que no solo consolidó los conocimientos vistos anteriormente, sino que también brindó herramientas para enfrentarnos a problemas más complejos en el futuro, con mayor facilidad y claridad en la resolución.

6. Referencias

Referencias

- [1] Apuntes de Programador – Apuntes de Golang. *Recursividad*. [En línea]. Accedido: 25-ago-2025. 2025. URL: <https://apuntes.de/golang/recursividad/#gsc.tab=0>.
- [2] Alejandro Gil Asensi. “La conjetura de Collatz”. En: *TEMat* 6 (2022), págs. 65-81. ISSN: 2530-9633. URL: <https://temat.es/articulo/2022-p65>.
- [3] Khan Academy. *The factorial function — Algoritmos recursivos*. [En línea]. Accedido: 25-ago-2025. s.f. URL: <https://es.khanacademy.org/computing/computer-science/algorithms/recursive-algorithms/a/the-factorial-function>.
- [4] Cristina Nieto Romero. *Tutorial Básico de Java 7: Estructuras de Control — Bucles y Condicionales*. [En línea]. Accedido: 25-ago-2025. 2025. URL: <https://medium.com/@crisnieromero/tutorial-b%C3%A1sico-de-java-7-estructuras-de-control-bucles-y-condicionales-4061c5f2bfe1>.
- [5] UNAM FES Cuautitlán. *Programación Recursiva*. [En línea]. Accedido: 25-ago-2025. 2025. URL: https://virtual.cuautitlan.unam.mx/intar/?page_id=185.
- [6] Universidad de Almería, Jardín de los matemáticos. *Sucesión de Fibonacci*. [En línea]. Accedido: 25-ago-2025. s.f. URL: <https://www2.ual.es/jardinmatema/sucesion-de-fibonacci/>.
- [7] Milan Vucic. *Concatenación de cadenas en Java*. [En línea]. Accedido: 25-ago-2025. 2023. URL: <https://codegym.cc/es/groups/posts/es.868.concatenacion-de-cadenas-en-java>.