



Recibe una cálida:

# ¡Bienvenida!

---

Te estábamos esperando 😊 

# ➤ Desarrollo de aplicaciones de consola en Java

---

**Plan formativo:** Desarrollo de Aplicaciones Full Stack Java Trainee V2.0

# HOJA DE RUTA

¿Cuáles **skill** conforman el programa?



# REPASO CLASE ANTERIOR

En la clase anterior trabajamos :

- ✓ Comprender la importancia e implementación de la clase de utilidad Math

# LEARNING PATHWAY

4.4

Start! 🏁

**Desarrollo de  
aplicaciones de  
consola en Java**

Creando aplicaciones de  
consola. Buenas prácticas de  
codificación.

Archivo .jar para ejecutar  
en consola

ejecutando.jar

# OBJETIVOS DE APRENDIZAJE

¿Qué aprenderemos?



**Reconocer las convenciones y estándares para una correcta codificación.**



**Comprender cómo generar una aplicación de consola en Java**





# Rompehielo 🧊



## Consolas: 🙌

Estas consolas solo permitían ingresar información por teclado y mostraban el resultado en una pantalla.

## Respondan en el chat o levantando la mano: ✍️

- ¿Alguien recuerda haber jugado en una consola como esta?
- ¿Cómo interactuamos con estos dispositivos?
- ¿Qué otros dispositivos funcionan de manera similar, aceptando entrada por teclado y mostrando texto en pantalla?





# Rompehielo 🧊



## Consolas actuales: 🙌

Hoy en día podemos crear aplicaciones con Java que se ejecutan por consola en la computadora, aceptando entrada por teclado y mostrando resultados textuales, justo como los viejos videojuegos.



```
C:\Windows\system32\cmd.exe

C:\javatest>java -jar console.jar
Name:
edu4java
Hello edu4java
edu4java

C:\javatest>java -jar console.jar edup
Helloedup

C:\javatest>_
```





# › Estándares, convenciones y estilos de codificación




# Estándares, convenciones y estilos de codificación



## ¿Para qué sirven?:

Dentro de Java 8 existen algunas convenciones implícitas que si las respetamos destacaremos por nuestro buen código. Su propósito es crear código de calidad, seguro, eficiente, escalable y fácil de mantener por distintos programadores a lo largo del tiempo.



Algunos de sus propósitos son:

- Legibilidad
- Consistencia
- Mantenibilidad
- Reducción de errores
- Trabajo en equipo
- Mejores prácticas
- Calidad

# Estándares, convenciones y estilos de codificación

## Convención de nombres:



- Los nombres de **Clases** deben comenzar con una letra **mayúscula** y seguir la convención de nombres en **CamelCase**. Por ejemplo: MiClase.
- Las **Clases** en Java, **NUNCA** van en **plural**.
- Los nombres de **métodos** y **variables** deben comenzar con una letra **minúscula** y seguir la convención de nombres en **camelCase**. Por ejemplo: miMetodo, miVariable.
- Los nombres de **constantes** deben estar **completamente** en **mayúsculas**, con palabras separadas por guiones bajos. Por ejemplo: MI\_CONSTANTE.



# Estándares, convenciones y estilos de codificación



## Organización de código:



- Es ideal que realices el **'import'** sólo de las clases que necesitas en lugar de usar importaciones comodín (`import java.util.*;`). Esto hace que el código sea más claro y evita posibles conflictos de nombres.
  - Agrupa las importaciones en bloques separados, por ejemplo, las importaciones de paquetes del JDK y las importaciones de paquetes de terceros.
- 
- 



# Estándares, convenciones y estilos de codificación



## Estilo de Sangría y formato:



- Utiliza **sangría de 4 espacios** para mejorar la legibilidad del código. (No es necesario preocuparse por esto, por qué Eclipse IDE ya lo hace)
  - Coloca las **llaves de apertura en la misma línea que la declaración** (por ejemplo, Clase o método), y las **llaves de cierre en una línea separada**.
- 
- 



# Estándares, convenciones y estilos de codificación



## Comentarios:

- Utiliza comentarios para **explicar el propósito y la lógica** detrás de las secciones de código complejas o importantes.
  - **Documenta tus clases** y métodos utilizando JavaDoc.
- 
- 

# LIVE CODING

Ejemplo en vivo

**Buenas prácticas:** *Vamos a desarrollar un programa bajo los parámetros que están enumerados abajo. Incluir comentarios y sangrías necesarias. Debemos adaptar lo pedido ya que no tiene buenas prácticas de nomenclatura.*

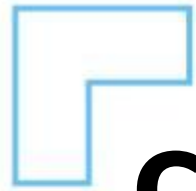
- 1. La clase debe llamarse “ejemplo buenas prácticas”.*
- 2. las variables a utilizar son “nombre usuario” y “edad usuario”.*
- 3. importar scanner.*
- 4. mostrar por consola los datos ingresados.*

 **Tiempo: 20 minutos**



# ➤ Creando aplicaciones de consola en Java





# Creando aplicaciones de consola en Java



Una de las posibilidades que nos ofrece Java con su JDK en la versión 1.8, es que podremos crear aplicaciones ejecutables en nuestras computadoras. Básicamente, podremos crear una aplicación con la cual podremos interactuar mediante la consola.



Las consideraciones que tendremos que tener en cuenta a la hora de hacerlo, son:

- Tener Java 1.8 u 8 instalado en la computadora.
- Tener instalado Eclipse IDE
- Tener alguna aplicación que hayamos programado, mediante la cual interactuamos mediante Scanner(System.in). En nuestro caso, utilizaremos el proyecto “Calculadora” que se encuentra detallado en el manual de la lección.



Vamos a trabajar sobre el proyecto “Calculadora” que se encuentra en el manual de la Lección 4 para crear una aplicación por consola. Aquí podemos ver la clase Calculadora completa:

```
1 import java.util.Scanner;
2
3 public class Calculadora {
4
5     public static void main(String[] args) {
6         Scanner scanner = new Scanner(System.in);
7
8         int num1 = 0;
9         int num2 = 0;
10        int resultado = 0;
11        String accionUsuario = "";
12        String operador = "";
13
14        System.out.println("Bienvenido a la calculadora de terminal");
15
16        do {
17            try {
18                System.out.println("Ingrese el primer número:");
19                num1 = scanner.nextInt();
20                System.out.println("Ingrese el segundo número:");
21                num2 = scanner.nextInt();
22                System.out.println("Ingrese el operador (+, -, *, /):");
23                operador = scanner.next();
24
25                switch (operador) {
26                    case "+":
27                        resultado = Calculadora.sumar(num1, num2);
28                        break;
29                    case "-":
30                        resultado = Calculadora.restar(num1, num2);
31                        break;
32                    case "*":
33                        resultado = Calculadora.multiplicar(num1, num2);
34                        break;
35                    case "/":
36                        resultado = Calculadora.dividir(num1, num2);
37                        break;
38                    default:
39                        System.out.println("Operador inválido");
40                        break;
41                }
42            }
43        }
```

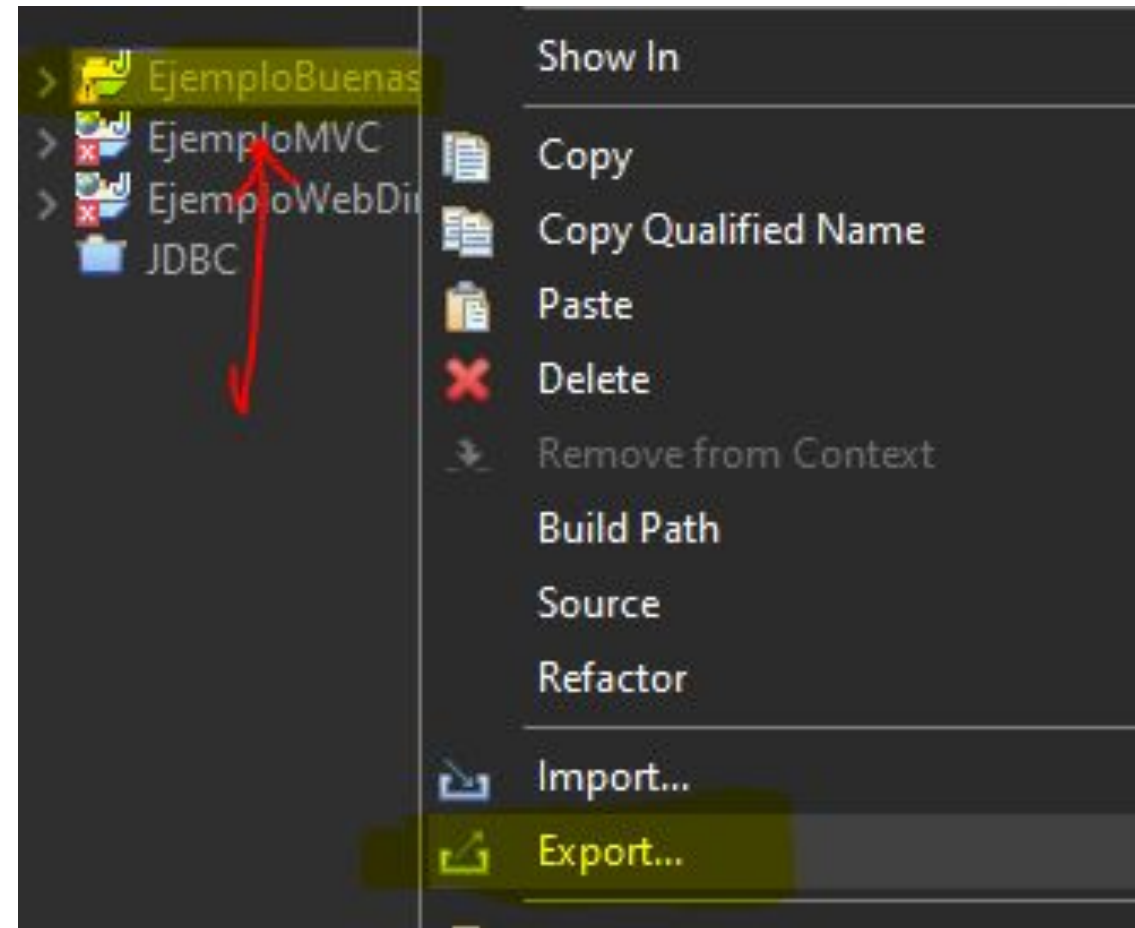
```
43
44        if (operador.equals("+") || operador.equals("-") || operador.equals("*") || operador.equals("/")) {
45            System.out.println("El resultado es: " + resultado);
46        }
47
48        } catch (Exception e) {
49            System.out.println("Error " + e.getMessage());
50        } finally {
51            System.out.println("-----");
52            System.out.println("Stop para detener la app, 's' para seguir");
53            accionUsuario = scanner.next();
54        }
55    }
56    } while (!accionUsuario.toLowerCase().equals("stop"));
57    System.out.println("Calculadora finalizada");
58
59    public static int sumar(int a, int b) {
60        return a + b;
61    }
62
63    public static int restar(int a, int b) {
64        return a - b;
65    }
66
67    public static int dividir(int a, int b) throws ArithmeticException {
68        return a / b;
69    }
70
71    public static int multiplicar(int a, int b) {
72        return a * b;
73    }
74
75    }
76 }
```



# Creando aplicaciones de consola en Java

## Haciendo el ejecutable .jar:

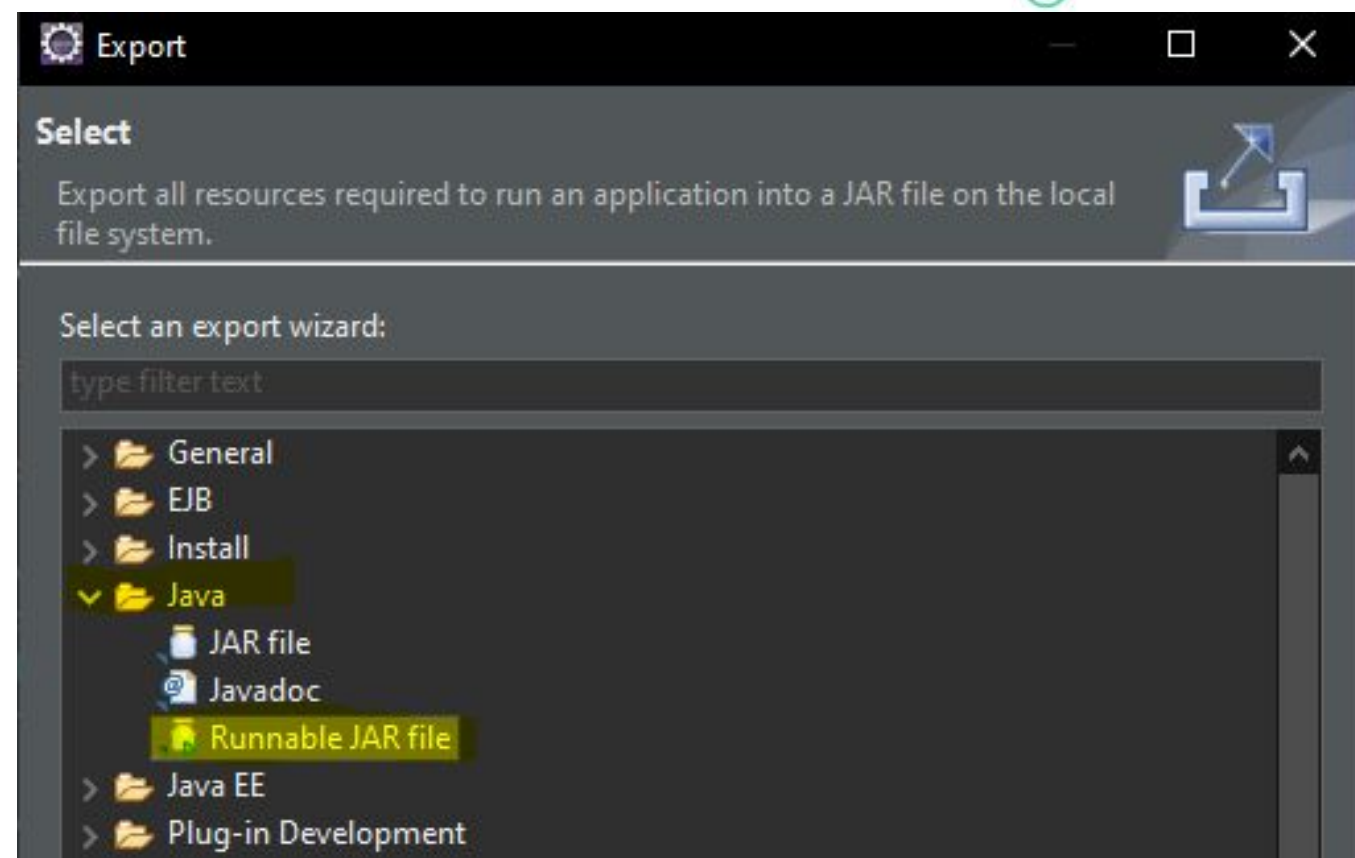
En el nombre del proyecto haces clic con el botón derecho y seleccionas la opción de “**Export**”.



# Creando aplicaciones de consola en Java

## Haciendo el ejecutable .jar:

En la ventana que se abre, abrimos la carpeta **Java** y seleccionamos la opción **Runnable JAR file**.



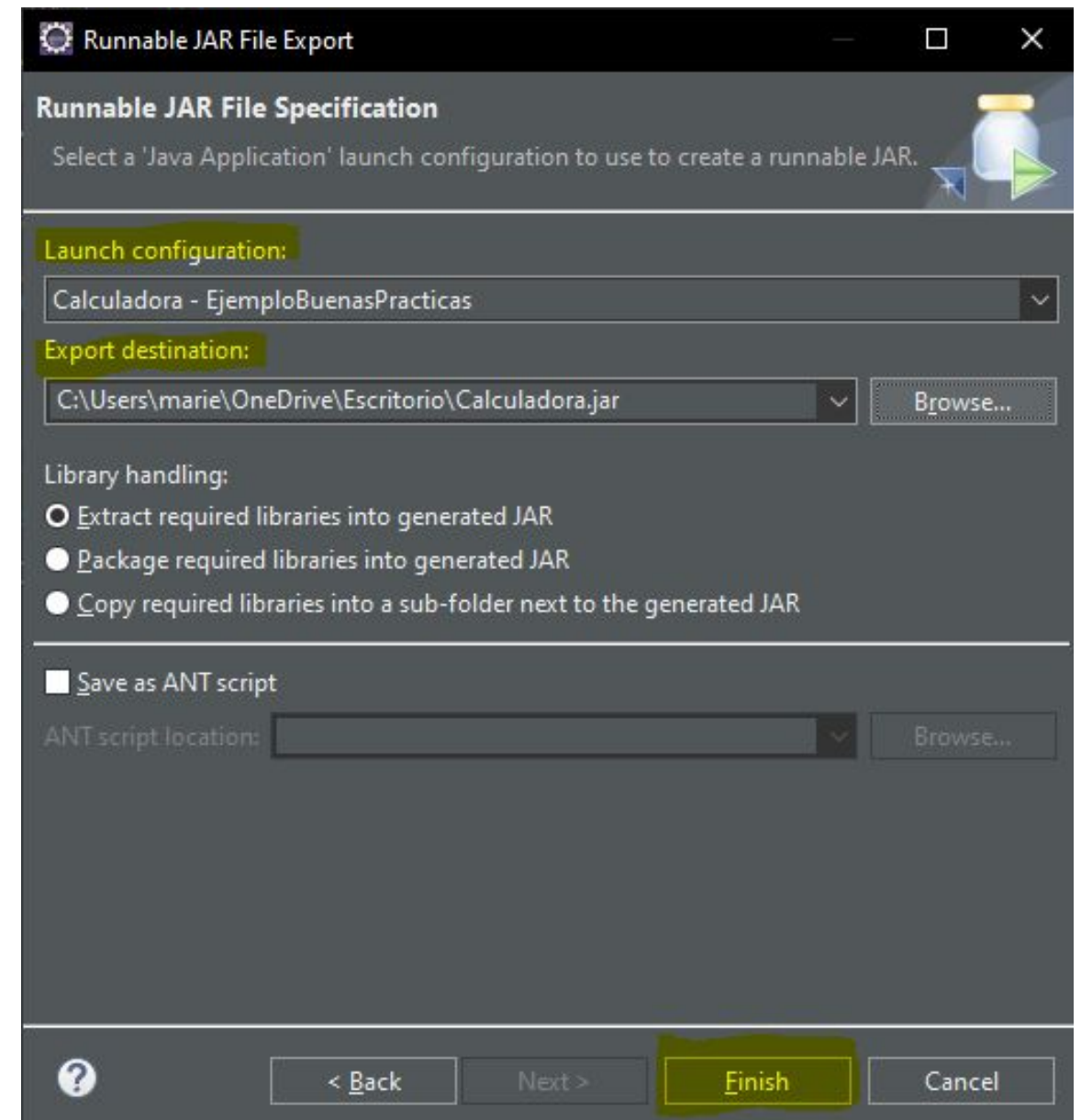


# Creando aplicaciones de consola en Java

## Haciendo el ejecutable .jar:

A continuación seleccionamos el destino de la “**Launch configuration**” del proyecto, y seleccionamos el “**Export destination**” donde se generará el ejecutable, en mi caso será el escritorio.

Luego, seleccionamos “Finish”.







# Creando aplicaciones de consola en Java

## Haciendo el ejecutable .jar:

Finalmente se generará un archivo ejecutable con extensión **.jar** (en este caso en el escritorio)





# Creando aplicaciones de consola en Java

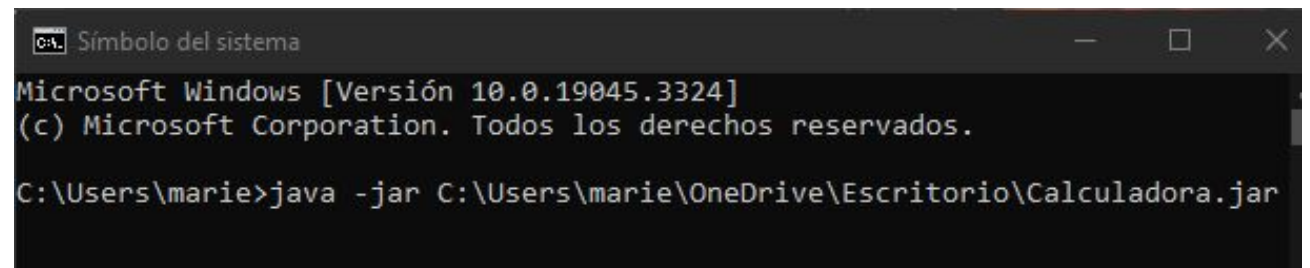
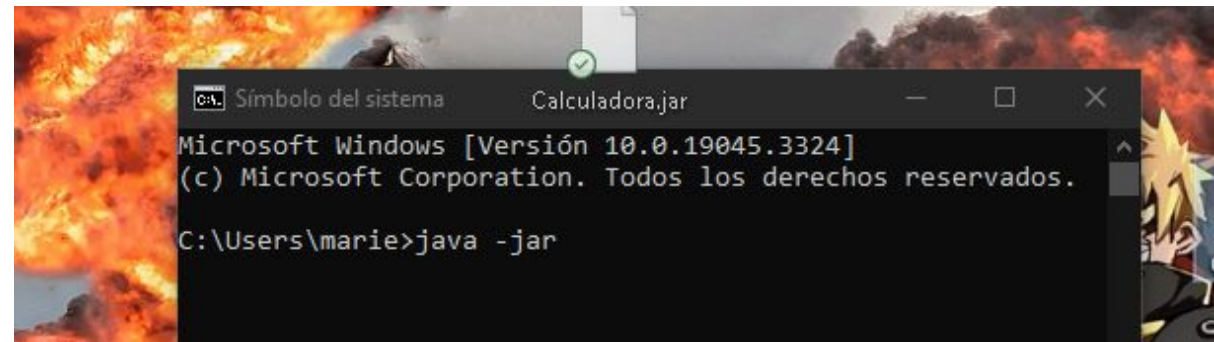


## Ejecutando el archivo .jar desde la terminal.

Buscamos en el sistema de la computadora nuestra terminal (cmd). y en ella escribimos el siguiente comando:

```
java -jar RutaHaciaMiApp.jar
```

Si no conoces la ruta exacta, puedes escribir “java -jar ” y arrastrar a la terminal la app.jar, como se muestra en la imagen.



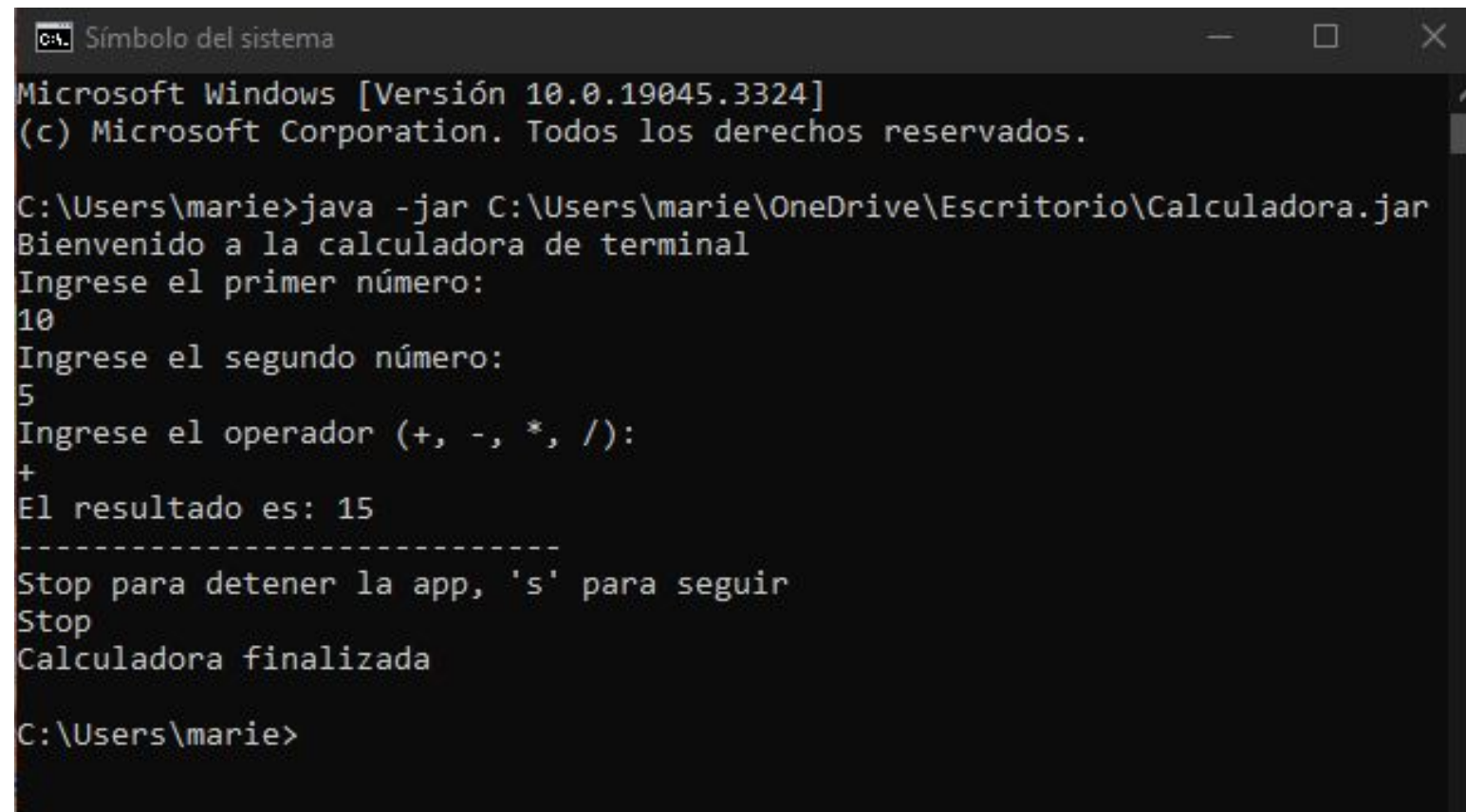


# Creando aplicaciones de consola en Java

## Ejecutando el archivo .jar desde la terminal.

De esta manera tenemos corriendo la app en la terminal, ahora solo debemos interactuar con ella y probarla.

Si quieres detener la ejecución de la app, solo tendrás que finalizar la app, puedes apretar “Ctrl+c” en la terminal.



```
C:\> Símbolo del sistema

Microsoft Windows [Versión 10.0.19045.3324]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\marie>java -jar C:\Users\marie\OneDrive\Escritorio\Calculadora.jar
Bienvenido a la calculadora de terminal
Ingrese el primer número:
10
Ingrese el segundo número:
5
Ingrese el operador (+, -, *, /):
+
El resultado es: 15
-----
Stop para detener la app, 's' para seguir
Stop
Calculadora finalizada

C:\Users\marie>
```



# LIVE CODING

Ejemplo en vivo

**Ejecutando desde la consola:**

*¡Vamos a practicar lo aprendido! Es hora de ejecutar desde la consola el EjemploBuenasPracticas que desarrollamos en vivo.*

1. *Crear el archivo .jar*
2. *Ejecutarlo desde la CMD*

 **Tiempo: 15 minutos**



# **Ejercicio N° 1**

# **ejecutando.jar**



# ejecutando.jar

## Ejecutando desde la consola: 🙌

Ejecutar archivos .jar por consola nos ahorra tiempo y configuración de IDEs de desarrollo, entre otros beneficios. Conocer esta técnica permite aprovechar al máximo la portabilidad de Java, automatizar tareas, trabajar como un desarrollador profesional y ejecutar aplicaciones de forma rápida y versátil.



## Consigna: ✍️

- 1- Crear una aplicación que solicite al usuario su nombre, su edad y su ocupación. La salida del programa será un saludo mostrando los datos ingresados.
- 2- Crear el archivo .jar y ejecutar por consola.
- 3- Aplicar buenas prácticas como sangrías, comentarios y nombres de variables claros.

**Tiempo** 🕒: 20 minutos

○

# ¿Alguna consulta?

+



# RESUMEN

¿Qué logramos en esta clase?

- ✓ **Reconocer los beneficios de los estándares, convenciones y estilos de codificación**
- ✓ **Aprender a generar un archivo .jar y ejecutarlo desde la terminal**



# #WorkingTime

Continuemos ejercitando

¡Antes de cerrar la clase! Te invitamos a: 🙌 🙌 🙌

1. Repasar nuevamente la grabación de esta clase
2. Revisar el material compartido en la plataforma de Moodle (lo que se vio en clase y algún ejercicio adicional)
  - a. Material 1 (Foro)
  - a. *Lectura Módulo 4, Lección 4: páginas 1 - 9*
3. Traer al próximo encuentro, todas tus dudas y consultas para verlas antes de iniciar nuevo tema.

# ¡Muchas Gracias!

Nos vemos en la próxima clase 🙌



Momento: ✚

# Time-out!

🕒 5 min.

