



Recibe una cálida:

¡Bienvenida!

Te estábamos esperando 😊 +

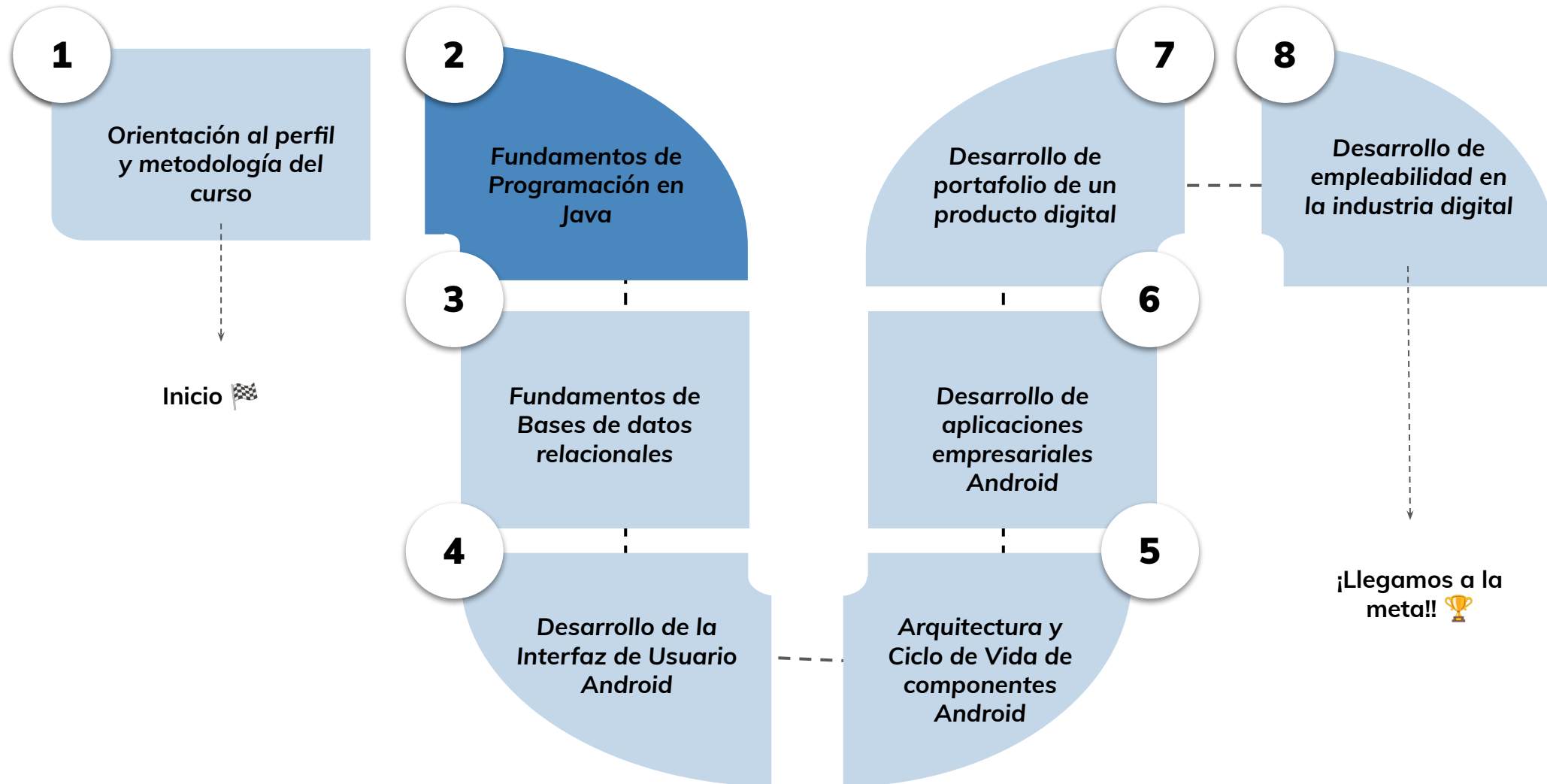


› Desarrollo de aplicaciones de consola en java

AE4.1: Implementar una aplicación básica de consola utilizando las buenas prácticas y convenciones para resolver un problema de baja complejidad acorde al lenguaje java.

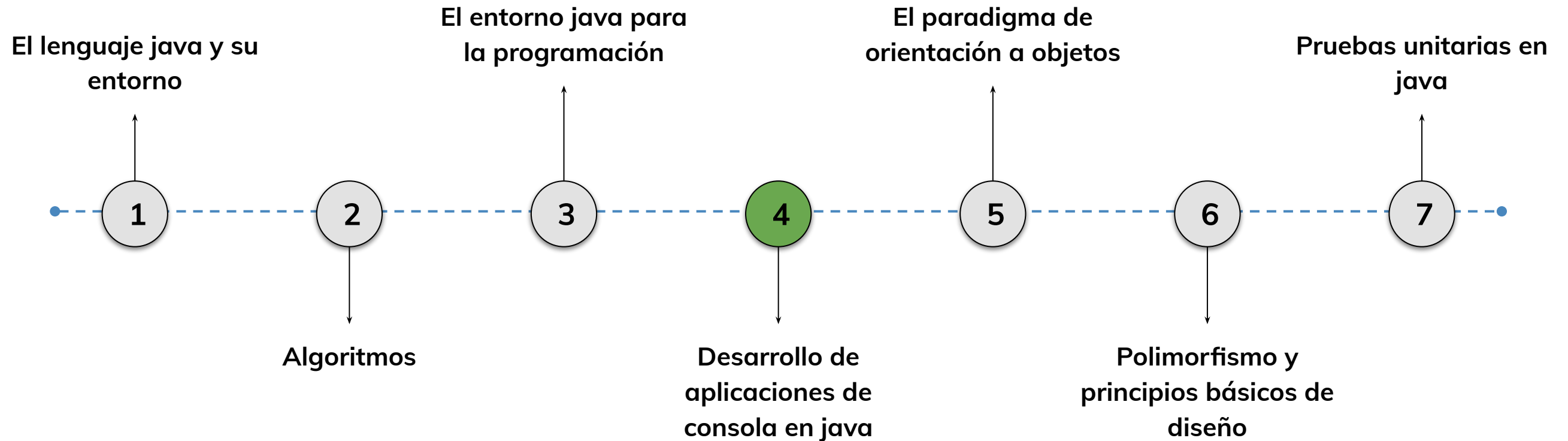
Hoja de ruta

¿Cuáles **módulos** conforman el programa?



Roadmap de lecciones

¿Cuáles lecciones estaremos estudiando en este módulo?



Learning Path

¿Cuáles temas trabajaremos hoy?

AE4.1

**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 aprenderás?



- 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 o levantando la mano:

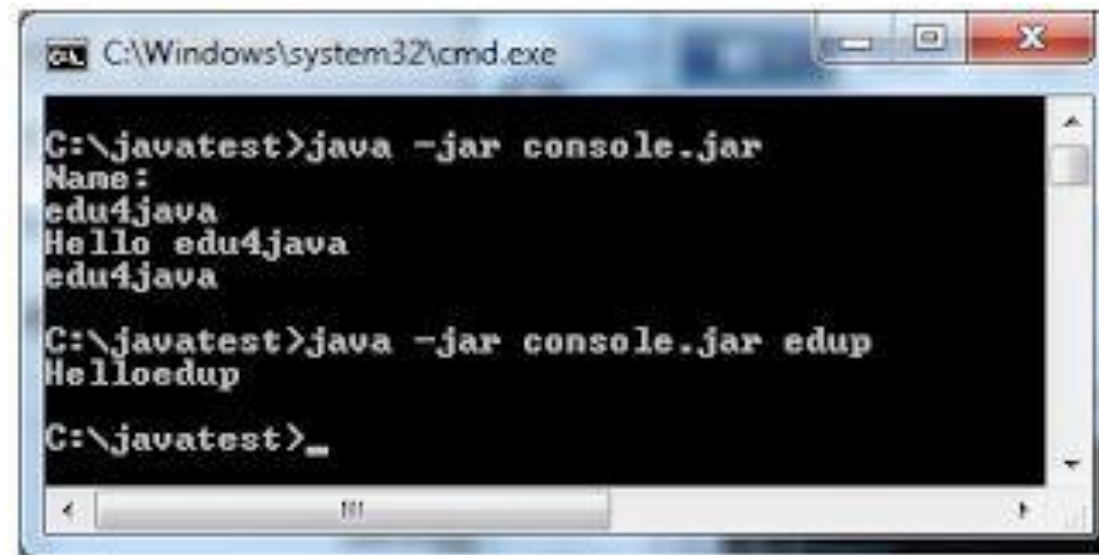
1. ¿Alguien recuerda haber jugado en una consola como esta?
2. ¿Cómo interactuamos con estos dispositivos?
3. ¿Qué otros dispositivos funcionan de manera similar, aceptando entrada por teclado y mostrando texto en pantalla?



< > 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 y otros IDEs ya lo hacen)
- 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

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.

LIVE CODING

```
import java.util.Scanner; // Importamos la clase Scanner para leer entradas del usuario

// Clase principal que sigue las buenas prácticas de nomenclatura
public class EjemploBuenasPracticas {

    public static void main(String[] args) {
        // Creamos un objeto Scanner para leer la entrada del usuario
        Scanner scanner = new Scanner(System.in);

        // Definimos las variables de acuerdo a las buenas prácticas (camelCase)
        String nombreUsuario; // Variable para almacenar el nombre del usuario
        int edadUsuario;       // Variable para almacenar la edad del usuario

        // Pedimos al usuario que ingrese su nombre
        System.out.print("Ingresa tu nombre: ");
        nombreUsuario = scanner.nextLine(); // Leemos el nombre del usuario

        // Pedimos al usuario que ingrese su edad
        System.out.print("Ingresa tu edad: ");
        edadUsuario = scanner.nextInt(); // Leemos la edad del usuario

        // Mostramos los datos ingresados por el usuario
        System.out.println("\nDatos ingresados:");
        System.out.println("Nombre del usuario: " + nombreUsuario);
        System.out.println("Edad del usuario: " + edadUsuario);

        // Cerramos el scanner para liberar recursos
        scanner.close();
    }
}
```




Momento:

Time-out!



5 -10 min.



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 u otro 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:

Creando aplicaciones de consola en Java

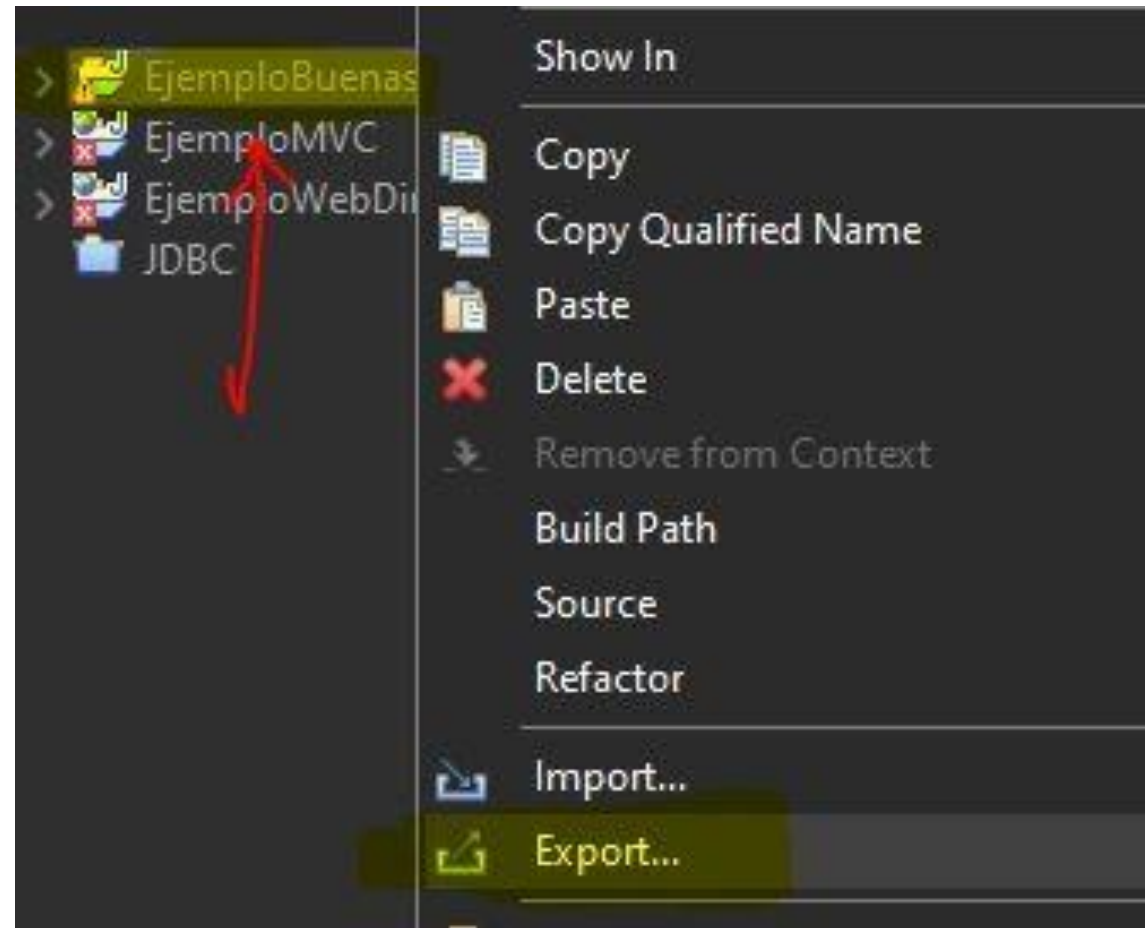
```
1 import java.util.Scanner;
2
3 public class Calculadora {
4
5     public static void main(String[] args) {
6         Scanner scanner = new Scanner(System.in);
7
8         int num1 = 0;
9         int num2 = 0;
10        int resultado = 0;
11        String accionUsuario = "";
12        String operador = "";
13
14        System.out.println("Bienvenido a la calculadora de terminal");
15
16        do {
17            try {
18
19                System.out.println("Ingrese el primer número:");
20                num1 = scanner.nextInt();
21                System.out.println("Ingrese el segundo número:");
22                num2 = scanner.nextInt();
23                System.out.println("Ingrese el operador (+, -, *, /):");
24                operador = scanner.next();
25
26                switch (operador) {
27                    case "+":
28                        resultado = Calculadora.sumar(num1, num2);
29                        break;
30                    case "-":
31                        resultado = Calculadora.restar(num1, num2);
32                        break;
33                    case "*":
34                        resultado = Calculadora.multiplicar(num1, num2);
35                        break;
36                    case "/":
37                        resultado = Calculadora.dividir(num1, num2);
38                        break;
39                    default:
40                        System.out.println("Operador inválido");
41                        break;
42                }
43            }
44        }
```

```
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    }
60
61    public static int sumar(int a, int b) {
62        return a + b;
63    }
64
65    public static int restar(int a, int b) {
66        return a - b;
67    }
68
69    public static int dividir(int a, int b) throws ArithmeticException {
70        return a / b;
71    }
72
73    public static int multiplicar(int a, int b) {
74        return a * b;
75    }
76
77 }
```

Creando aplicaciones de consola en Java

Haciendo el ejecutable .jar:

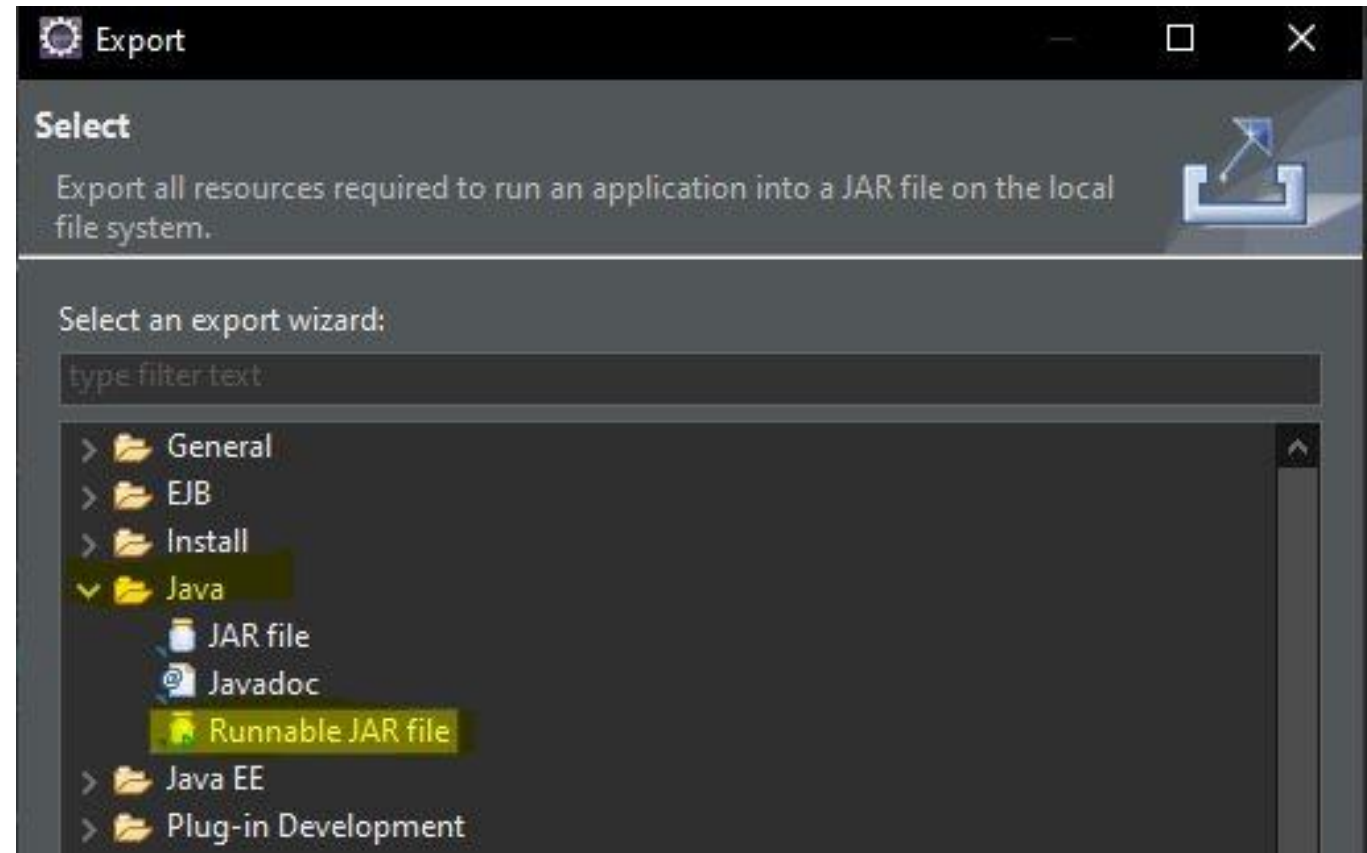
En el nombre del proyecto haces clic con el botón derecho y seleccionas la opción de “**Export**”.



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**.

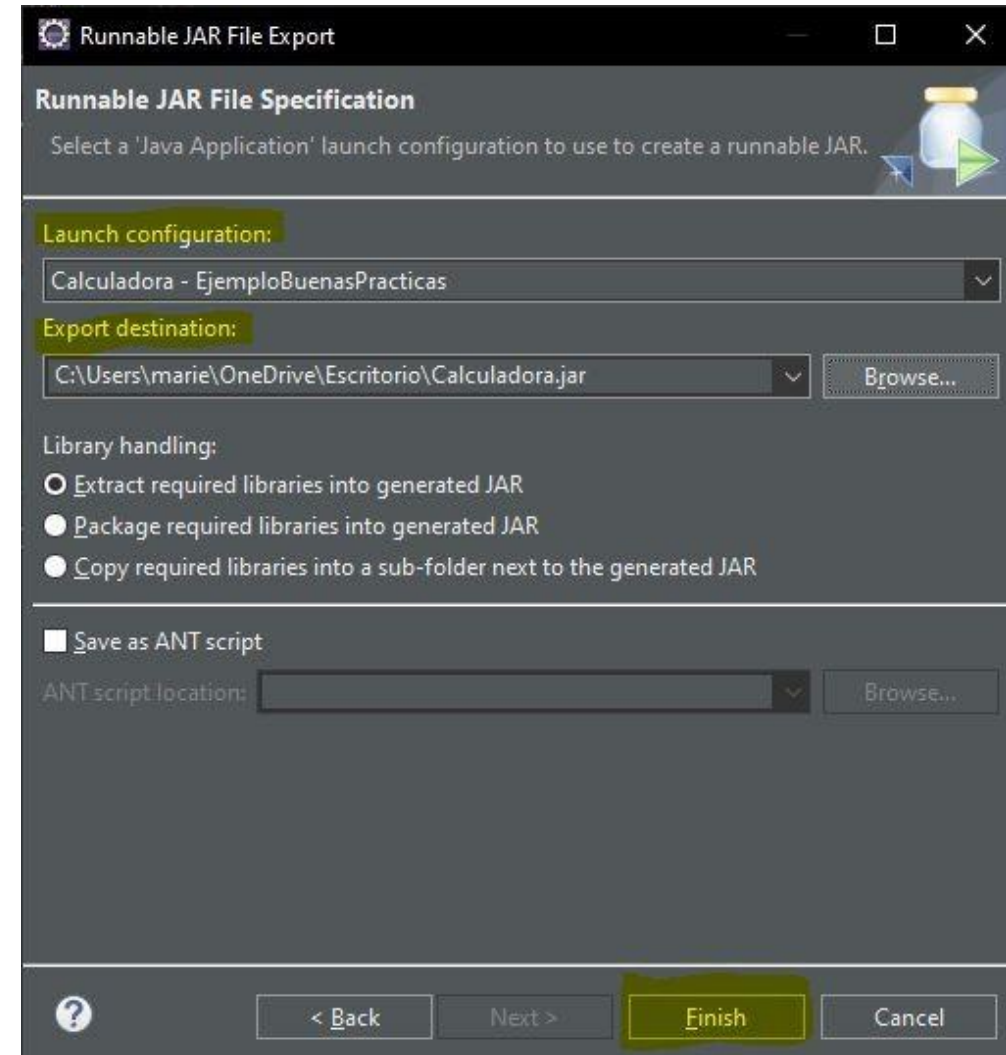


Creando aplicaciones de consola en Java

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)



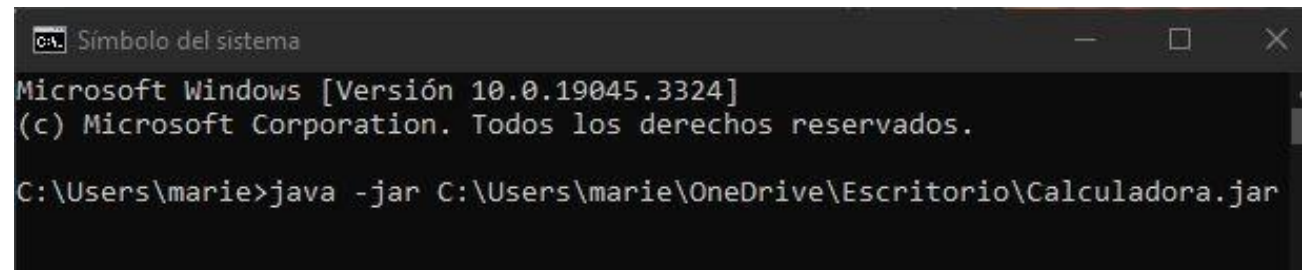
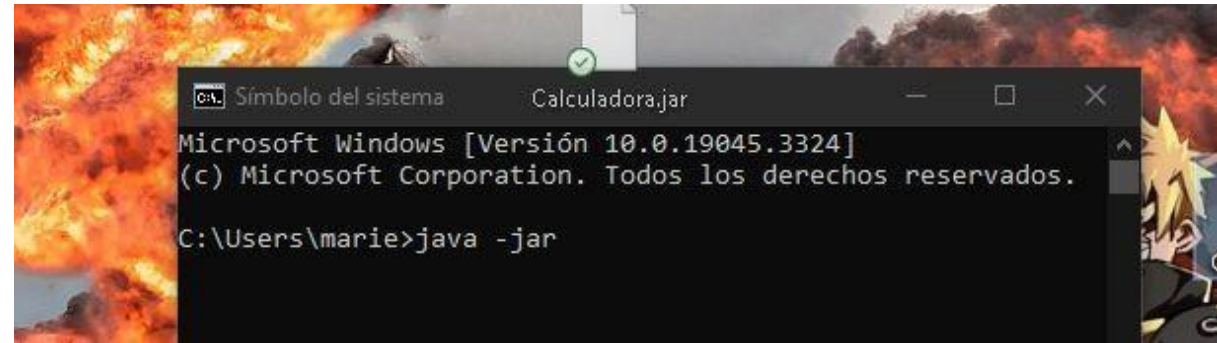
Creando aplicaciones de consola en Java

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.

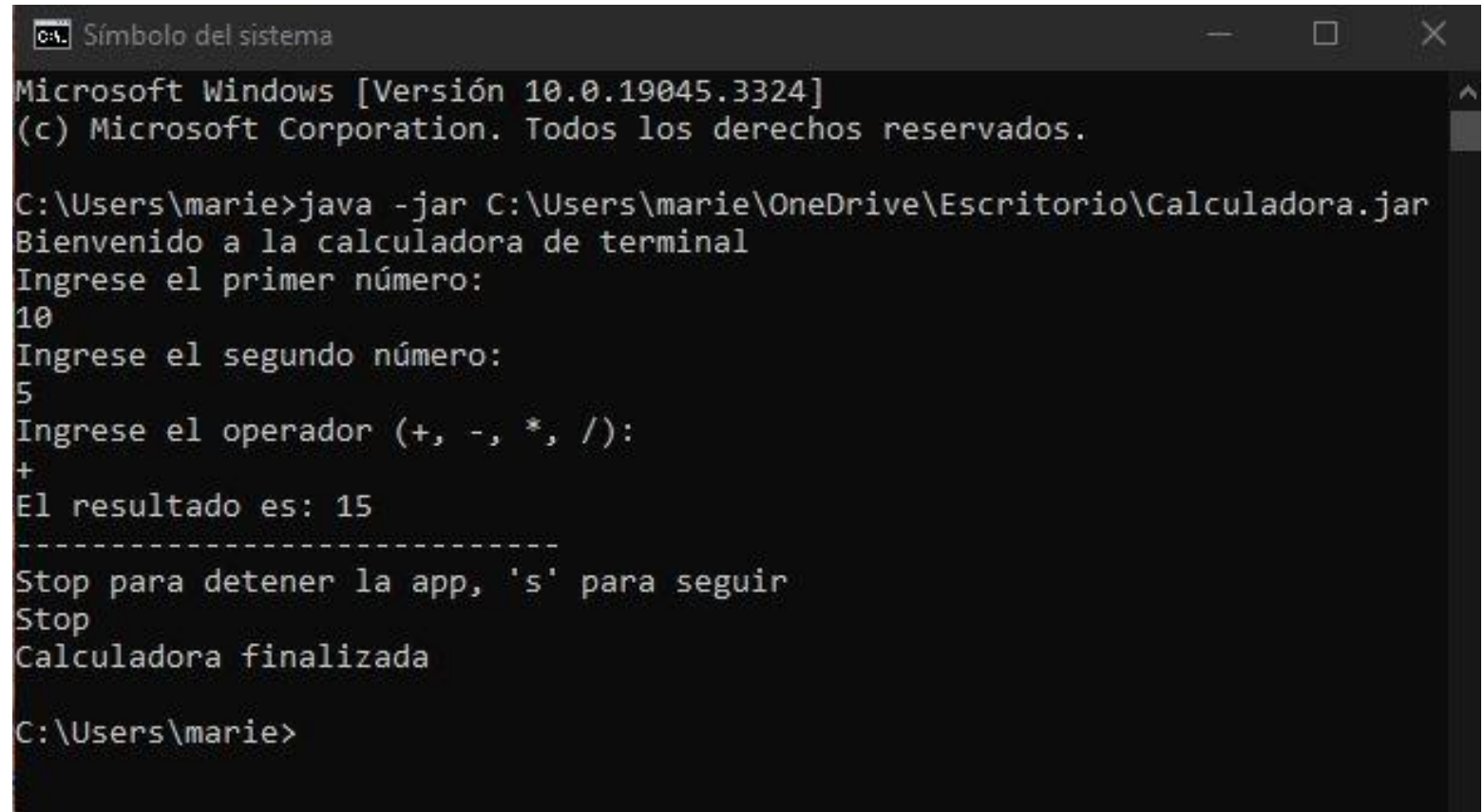


Creando aplicaciones de consola en Java

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.



```
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

Ejecutando desde la consola:

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

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

LIVE CODING

Calculadora Simple con Operaciones Aritméticas (Buenas Prácticas)

Objetivo: Desarrollar una calculadora básica que permita al usuario realizar operaciones de suma, resta, multiplicación y división. Se aplicarán buenas prácticas de código, como la modularización y el manejo de errores.

LIVE CODING

Instrucciones:

Entrada de datos: El usuario ingresará dos números y el tipo de operación que desea realizar (suma, resta, multiplicación o división).

Procesamiento: Según la operación seleccionada, el programa calculará el resultado y lo mostrará en consola.

Manejo de errores: El programa validará entradas incorrectas y evitará la división por cero.

Buenas prácticas aplicadas:

Uso de nombres claros: Las variables y métodos tienen nombres descriptivos que indican su propósito.

Modularización: Separación de la lógica en métodos, evitando que el método main sea demasiado largo.

Validación de entradas: Se valida la operación seleccionada y se maneja correctamente la división por cero.

Comentarios claros: Se incluyen comentarios explicativos en puntos clave del código para mejorar la comprensión

LIVE CODING



□ <https://gist.github.com/Mariocanedo/654bac644e59d88adf8a347a02acbb0c>



Ejercicio N° 1

Ejecutando.jar



Ejecutando.jar



Contexto: 🙌

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



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

¿Alguna consulta?





¡Ponte a prueba!

Momento de ejercitación

Te invitamos a aprovechar esta última sección del espacio sincrónico para realizar de manera individual las **actividades disponibles en la plataforma**. Estas propuestas son clave para afianzar lo trabajado y **forman parte obligatoria del recorrido de aprendizaje**.

👉 **Análisis de caso** ————— 👉 **Selección Múltiple**

👉 **Comprensión lectora**

Si al resolverlas surge alguna duda, compartela o tráela al próximo encuentro sincrónico.

< **¡Muchas gracias!** >

