



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 :

- ✓ Reconocer las convenciones y estándares para una correcta codificación.
- ✓ Comprender cómo generar una aplicación de consola en Java

# LEARNING PATHWAY

4.4

Start! 🚩

**Desarrollo de  
aplicaciones de  
consola en Java**

Depuración de programas  
utilizando el IDE.  
Documentación de programas  
con JavaDoc.

Depuración y  
Documentación en Java -  
Eclipse

Depurando y  
Documentando

# OBJETIVOS DE APRENDIZAJE

¿Qué aprenderemos?



**Comprender cómo realizar la depuración en el IDE Eclipse**



**Aprender a documentar código con JavaDoc**





# Rompehielo

## ¡Encuentra el Bug!: 🙌

En la siguiente imagen veremos un código Java con algunas particularidades.

### Consigna: 📝 Contesten en el chat!

- ¿Cuántos errores pueden encontrar?
- ¿Cuáles son?
- ¿Cómo se llama el proceso de encontrar y corregir errores en código?



```
1 public class Main {
2     public static void main(String[] args) {
3         //variables del programa
4         char numero1 = 10;
5         int numero2 = '15';
6         int numero3 = 10
7
8         //condicional if
9         if (numero2 < numero3){
10             System.out.println "Se ejecuta el primer else if.";
11         } else if {
12             System.out.println("Se ejecuta el segundo else if.")
13         } else {
14             System.out.println("Se ejecuta el else.");
15         }
16     }
17 }
```

# › Depuración de programas utilizando el IDE





# Depuración

## ¿Qué es y por qué es importante?:

La depuración o debugging es el **proceso de encontrar y corregir errores** y defectos en el código de un programa informático.

La depuración de código es esencial para garantizar la calidad y el rendimiento adecuados de tus aplicaciones Java. Algunos **beneficios clave** de la depuración son:



- Identificar la **causa** raíz de errores y excepciones.
- Entender mejor el **flujo** de ejecución de un programa.
- Evaluar el **estado** de variables en tiempo real.
- **Comparar** los resultados esperados con los actuales.
- Encontrar y solucionar **problemas** lógicos.
- Mejorar la **calidad** y **confiabilidad** del software.



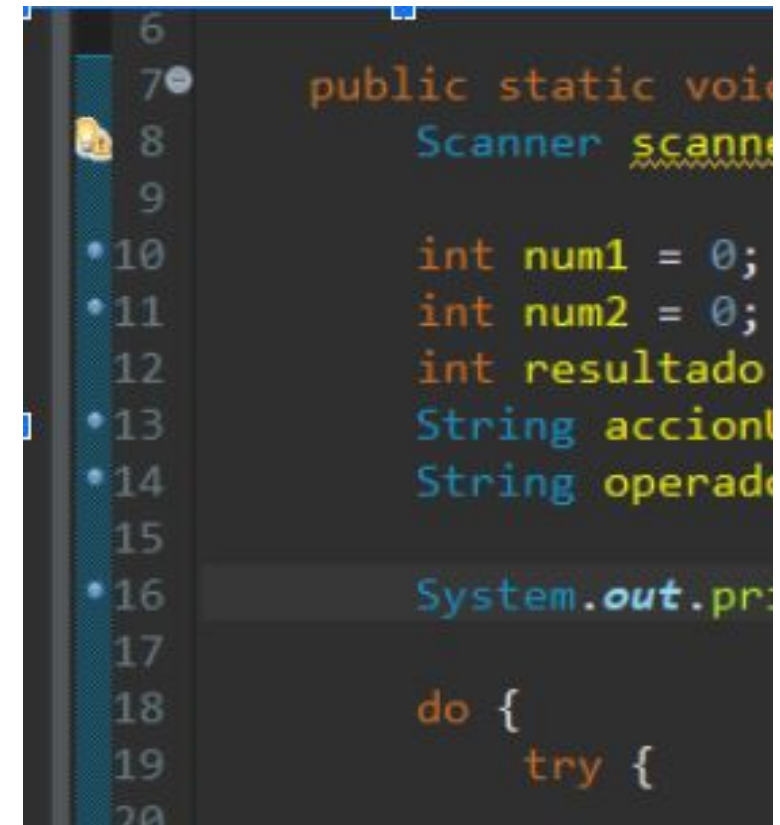
# Depuración en Eclipse



## ¿Cómo se hace?

Realizar una depuración de código en Eclipse IDE es una tarea bastante sencilla y te lo mostraremos paso a paso:

- **Colocar Breakpoints:** Los breakpoint son puntos que nosotros determinamos en nuestro código, **donde queremos que la ejecución se detenga** para poder evaluar estados de variables y procesos, los mismos se colocan haciendo doble clic sobre el número de línea seleccionado a la izquierda de nuestro código.



```
6
7 public static void
8     Scanner scanner
9
10     int num1 = 0;
11     int num2 = 0;
12     int resultado
13     String accion
14     String operado
15
16     System.out.pr
17
18     do {
19         try {
20
```

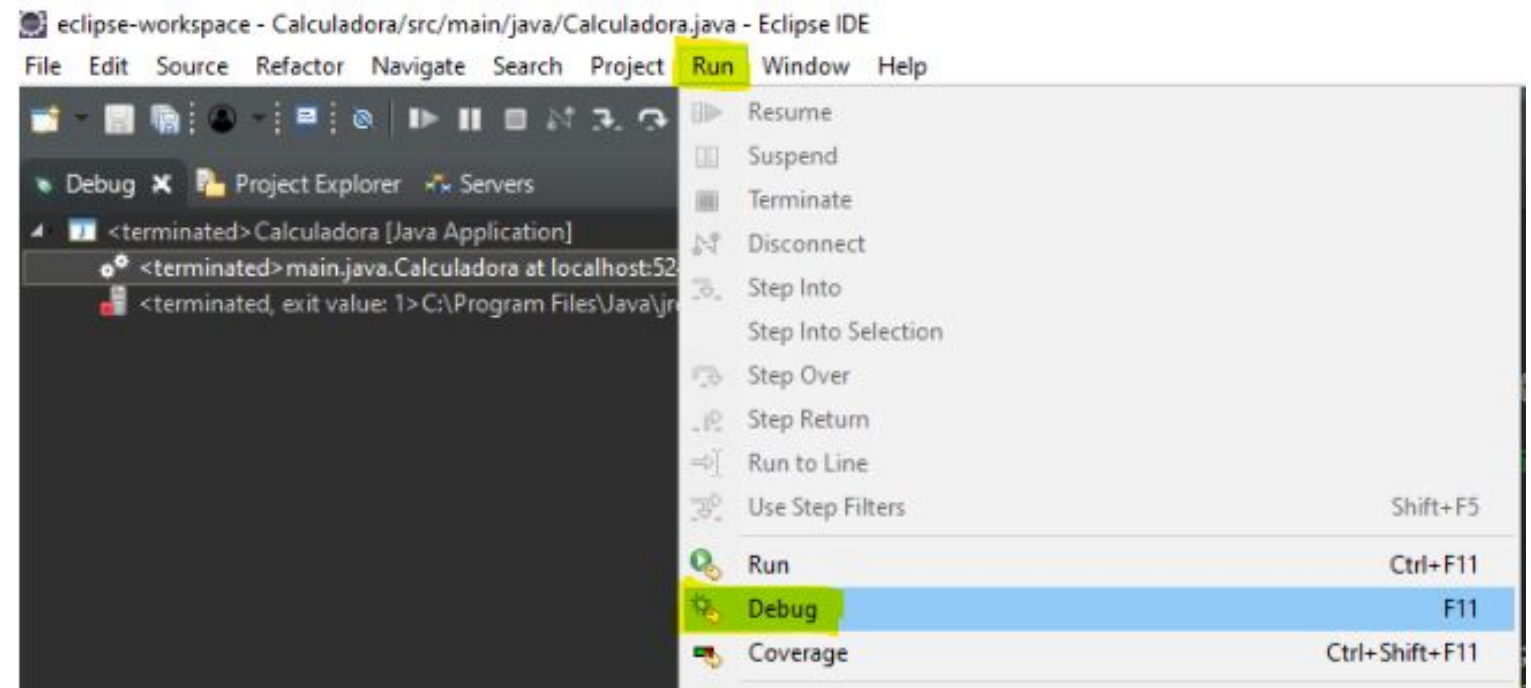


# Depuración en Eclipse



## ¿Cómo se hace?

- **Depurando código en modo Debug:** Para poder depurar nuestro código, debemos anticipar a nuestro IDE que lo haremos, esto se realiza de una forma muy sencilla, luego de haber colocado nuestros Breakpoints, buscaremos en la parte superior de Eclipse la opción Run o ejecutar y haremos click en **Debug**.





# Depuración en Eclipse

## ¡Siendo expertos en depuración!

Una vez que hayamos corrido el modo Debug, se nos abrirá una perspectiva distinta donde tendremos la posibilidad de observar las variables creadas hasta el momento del breakpoint específico, los breakpoint existentes y nos dará la posibilidad de crear expresiones en base a variables.

Por último, cuando estemos en un breakpoint, la ejecución se detendrá. Para poder continuarla deberemos decidir cómo hacerlo, es por eso que les compartimos las posibilidades:

**Tecla F8** nos llevará al **siguiente breakpoint**, en caso de no existir continuará la ejecución de forma normal.

**Tecla F5** nos llevará al **interior del proceso** que continua, por ejemplo si en una línea se tiene que ejecutar una función, F5 nos llevará hasta esa función.

**Tecla F6** continuará la **ejecución paso a paso**, o línea a línea, sin adentrarse en las posibles funciones a llamarse.

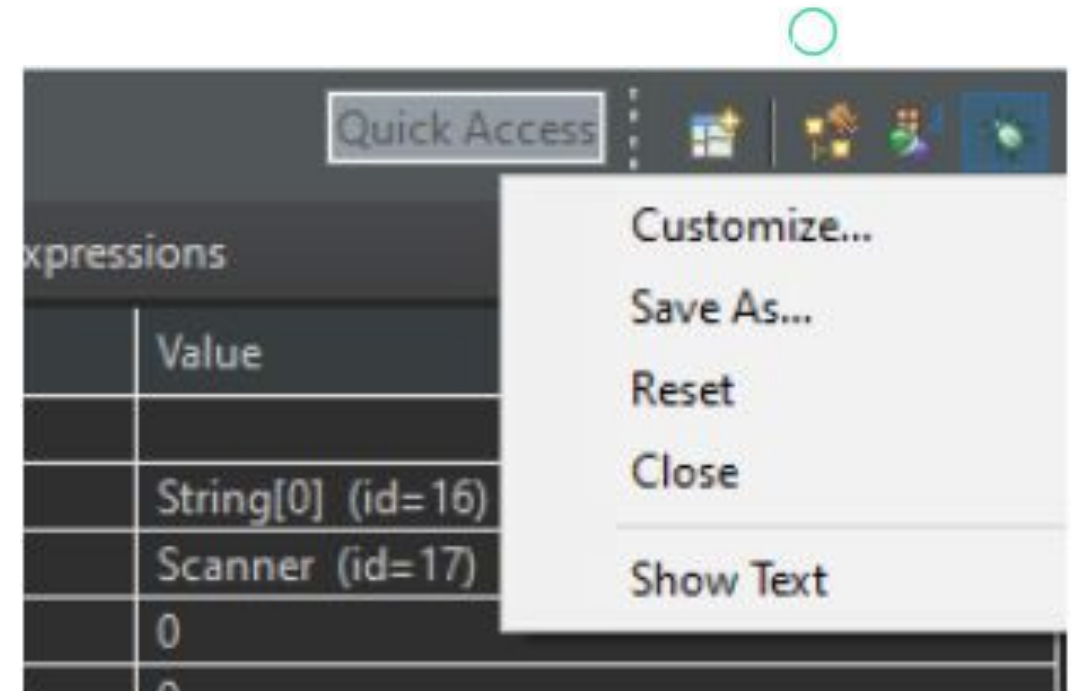


# Depuración en Eclipse



Por último, en el caso que modifiquen la perspectiva de eclipse y no puedan volver a ver las Variables, Breakpoints o Expressions, hacer lo siguiente:

Hacer click en “**Reset**”, y podrás ver la perspectiva del Debug por defecto, también aplica a la perspectiva de Java.



# LIVE CODING

Ejemplo en vivo

## Poniendo en práctica:


*Vamos a depurar el proyecto de la calculadora que trabajamos en la clase anterior, para poder ver el acceso a métodos y el cambio de flujo del programa.*

1. *Debuggear el proyecto EjemploConsola (calculadora). De ésta manera podremos aprender a visualizar cada ámbito que deseemos depurar.*

 **Tiempo: 20 minutos**



# ➤ Documentando el código con JavaDoc



# JavaDoc



## ¿Qué es?:

La documentación en Java se realiza utilizando **JavaDoc**, una herramienta que genera documentación en formato HTML a partir de **comentarios especiales** escritos en el código fuente.



Los comentarios JavaDoc se agregan **encima de declaraciones** de clases, métodos, campos y paquetes, y proporcionan información sobre su propósito, funcionamiento y uso.



Debemos recordar que un código bien documentado es un código mucho más legible y mantenible.







## ¿Para qué sirve?:

- Generar documentación legible de API a partir de comentarios especiales en el código.
- Explicar la funcionalidad y uso de clases, métodos, atributos, etc.
- Crear documentación actualizada de forma sincronizada con el código.
- Facilitar el entendimiento y lectura del código por parte de otros desarrolladores.
- Publicar la documentación en formato web para compartir fácilmente.
- Seguir estándares y convenciones de documentación de código.
- Complementar documentos técnicos con descripciones embebidas en el código.
- Proveer ejemplos de uso directamente en la documentación generada.
- Automatizar el proceso de generar, actualizar y publicar documentación.





# JavaDoc

## ¿Cómo se comenta?:

Los comentarios de documentación se inician con `/**` y se cierran con `*/`. Pero simplemente colocando `/**` o `/*` y dando “**Enter**”, nos generará la plantilla para documentar.

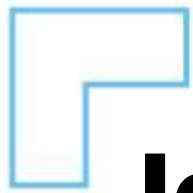
Podemos ver cómo de manera automática genera los comentarios para cada parámetro esperado por el método.



```
public class Calculadora {  
    /**  
     *  
     * @param args  
     */  
    public static void main(String[] args) {  
        // ...  
    }  
}
```



```
/**  
 *  
 * @param a  
 * @param b  
 * @return  
 */  
public static int sumar(int a, int b) {  
    return a + b;  
}
```



# JavaDoc

## Documentando una Clase:



```
1 package main.java;
2
3 /*
4  * Descripcion de la clase.
5  */
6 public class Logica {
7
8     /**
9      * Descripcion de la variable
10     */
11     private MiEnum variableEnum;
12
13
14     public Logica() {
15         // TODO Auto-generated constructor stub
16     }
17
18     /**
19      * Descripcion de la funcion,
20      * @param a sera una variable entera
21      * @param b sera una variable entera
22      * @return Esta funcion devolvera un valor de la division de a y b
23      * @throws ArithmeticException posible error que lance la funcion
24     */
25     public static int dividir(int a, int b) throws ArithmeticException {
26         return a / b;
27     }
28 }
29
```

# JavaDoc

## Documentando un Enum:

```
package main.java;

public enum MiEnum {
    VALOR1, // Descripción del primer valor
    VALOR2  // Descripción del segundo valor
}
```

# JavaDoc

Documentando una Interfaz:

```
2
3 /**
4  * Descripcion de la interfaz
5  * @author AsteriX
6  *
7  */
8 public interface MiInterfaz {
9
10     /**
11     *
12     * @param a
13     * @param b
14     */
15     public void func(int a, int b);
16 }
```

# LIVE CODING

Ejemplo en vivo

## Documentar:

*Vamos a documentar el proyecto de la calculadora que trabajamos en el ejercicio anterior.*

- 1. Agregar comentario de Clase.*
- 2. Agregar comentarios de variables.*
- 3. Agregar comentarios de métodos.*

 **Tiempo: 20 minutos**



# **Ejercicio N° 1**

# **Depurando y Documentando**



# Depurando y documentando



🙌 Escribe el siguiente código en una Main Class de java:

**Consigna:** ✍️

- 1- Depurar el programa para buscar un error y encontrar la causa.
- 2- Documentar el programa

**Tiempo** 🕒: 25 minutos



```
public static void main(String[] args) {  
    int x=0;  
    depurar(x);  
}  
public static void depurar(int x) {  
    while(x<1){  
        System.out.println("Ciclo");  
    }  
}
```



○

# ¿Alguna consulta?

+



# RESUMEN

¿Qué logramos en esta clase?

- ✓ Reconocer la depuración como elemento importante del desarrollo
- ✓ Identificar la implementación del debugging
- ✓ Comprender la utilización de JavaDoc para comentar el código.



# #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)
  - b. *Lectura Módulo 4, Lección 4: páginas 9 - 14*
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.

