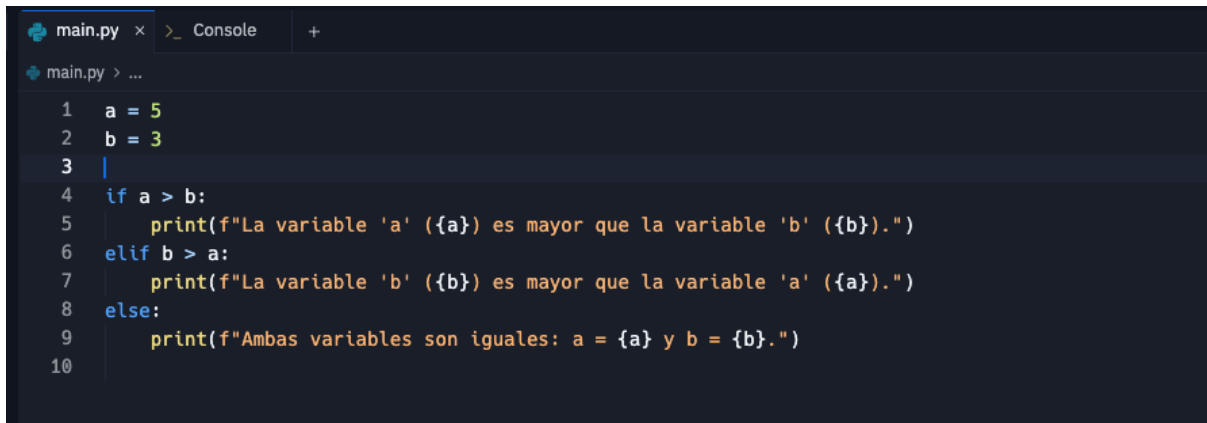


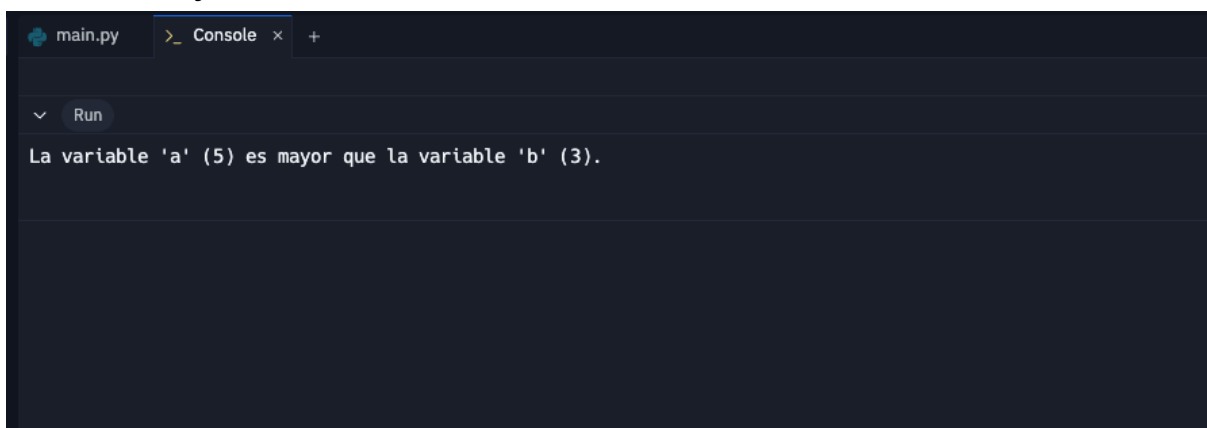
# PRÁCTICA TEMA 2: EVALUACIÓN DE LOS IDEs.

## Ejercicio 1.



```
main.py x >_ Console +
main.py > ...
1 a = 5
2 b = 3
3 |
4 if a > b:
5     print(f"La variable 'a' ({a}) es mayor que la variable 'b' ({b}).")
6 elif b > a:
7     print(f"La variable 'b' ({b}) es mayor que la variable 'a' ({a}).")
8 else:
9     print(f"Ambas variables son iguales: a = {a} y b = {b}.")
10
```

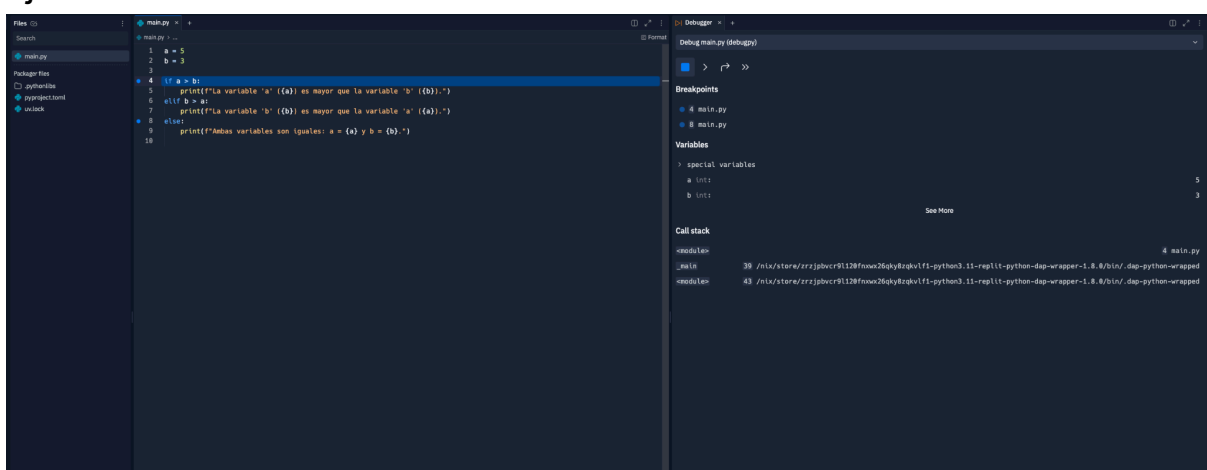
Este es el ejercicio hecho.



```
main.py >_ Console x +
Run
La variable 'a' (5) es mayor que la variable 'b' (3).
```

Este es el ejercicio ejecutado y mostrado por consola.

## Ejercicio 2.

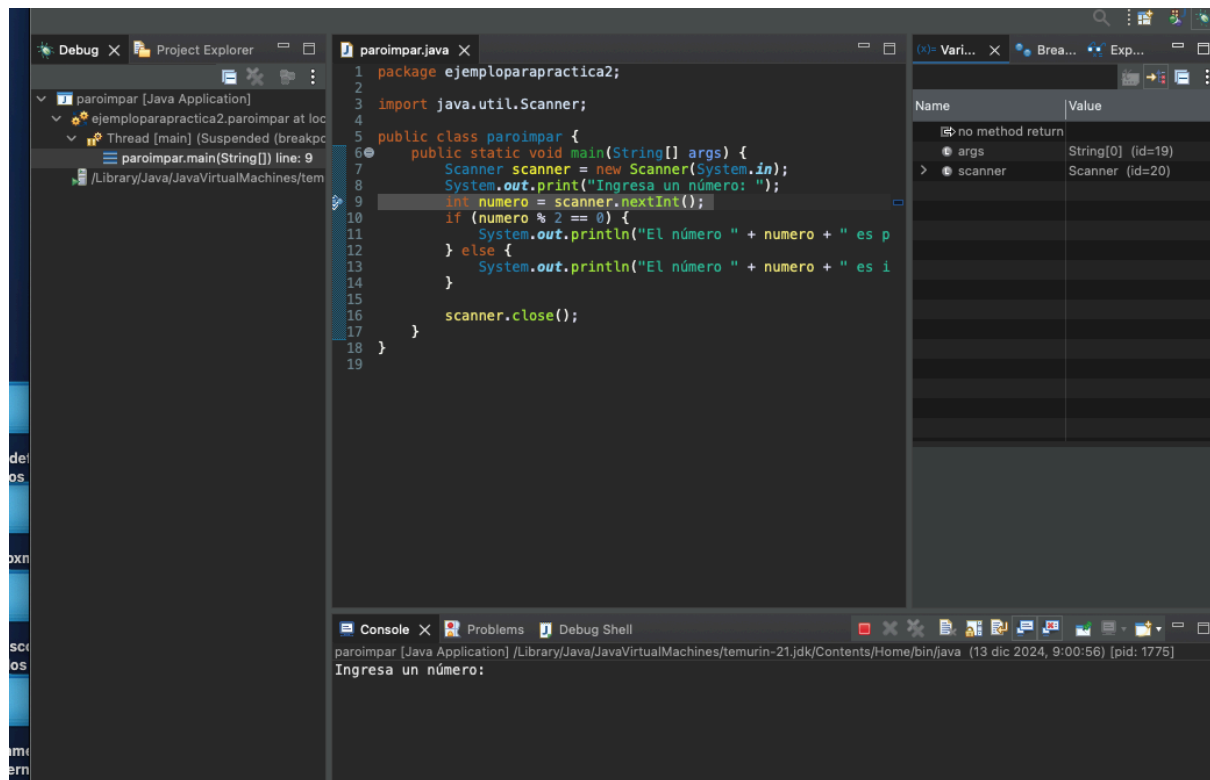


```
Files (1)
main.py
Package files
pythonista
pyproject.toml
venv

main.py x
1 a = 5
2 b = 3
3
4 if a > b:
5     print(f"La variable 'a' ({a}) es mayor que la variable 'b' ({b}).")
6 elif b > a:
7     print(f"La variable 'b' ({b}) es mayor que la variable 'a' ({a}).")
8 else:
9     print(f"Ambas variables son iguales: a = {a} y b = {b}.")
10

Debugger
Debug main.py (debugpy)
Breakpoints
4 main.py
8 main.py
Variables
> special variables
a: int: 5
b: int: 3
See More
Call stack
<module> 4 main.py
__main__ 39 /nix/store/rzjjbccc9l128fnwv26ky8zqkv1f-python3.11-repl-1.1/python-dap-wrapper-1.8.0/bin/dap-python-wrapped
<module> 43 /nix/store/rzjjbccc9l128fnwv26ky8zqkv1f-python3.11-repl-1.1/python-dap-wrapper-1.8.0/bin/dap-python-wrapped
```

## Ejercicio 4.



## Ejercicio 5.

- Navegar entre las pestañas (archivos) abiertas. Ctrl + Shift + Tab
- Comentar líneas o bloques de líneas. Ctrl + Shift + /
- Para ir a un número de línea. Ctrl + L
- Para cerrar el archivo actual. Ctrl + W
- Cerrar todos los archivos. Ctrl + Shift + W
- Buscar y/o reemplazar. Ctrl + F
- Para formatear el código (arreglar tabulaciones). Ctrl + Shift + F
- Borrar una línea completa. Ctrl + D
- Para mostrar las sugerencias del código escrito. Ctrl + Space

## Ejercicio 6.

### 1. Cambiar de workspace

Paso 1: En el menú superior, selecciona File.

Paso 2: Haz clic en Switch Workspace.

Paso 3: Elige Other... si quieres seleccionar otro workspace específico.

Paso 4: Aparecerá una ventana para que selecciones el nuevo directorio del workspace. Después de elegirlo, Eclipse se reiniciará automáticamente con ese nuevo workspace.

## 2. Generar código automáticamente (constructores, getters, setters)

Paso 1: Abre la clase donde quieres generar el código.

Paso 2: Haz clic derecho en el código o en cualquier parte de la clase.

Paso 3: En el menú contextual, selecciona Source.

Paso 4: Selecciona la opción deseada:

Generate Getters and Setters...: Para generar getters y setters para los campos de la clase.

Generate Constructor using fields...: Para generar un constructor que inicialice todos los campos de la clase.

Paso 5: Eclipse generará el código automáticamente en la clase.

Alternativamente, también puedes usar el atajo de teclado:

Windows/Linux: Alt + Shift + S y luego elegir la opción adecuada.

Mac: Cmd + Shift + S y luego elegir la opción.

## 3. Mostrar el terminal (consola)

Paso 1: En la parte inferior de la ventana de Eclipse, deberías ver diferentes vistas como "Console", "Problems", "Tasks", etc.

Paso 2: Si no ves la vista de "Console", ve al menú superior y selecciona Window -> Show View -> Console.

Paso 3: Se abrirá el terminal o la consola en la parte inferior de Eclipse, donde puedes ver la salida de tus programas.

## 4. Renombrar una variable en todo el proyecto (Refactorizar)

Paso 1: Haz clic derecho sobre la variable que quieres renombrar.

Paso 2: En el menú contextual, selecciona Refactor -> Rename (o usa el atajo Alt + Shift + R).

Paso 3: Escribe el nuevo nombre de la variable.

Paso 4: Eclipse te mostrará una vista previa de todos los lugares donde se cambiará el nombre. Asegúrate de que todo esté correcto y haz clic en OK para confirmar.

## 5. Acceder a los ajustes generales de Eclipse

Paso 1: Ve al menú superior y selecciona Window -> Preferences.

Paso 2: Se abrirá una ventana con todos los ajustes de Eclipse. Aquí podrás configurar todo, desde la apariencia hasta la configuración del compilador, la gestión de proyectos, y mucho más.

## 6. Cambiar el tema de Eclipse

Paso 1: Ve al menú Window -> Preferences.

Paso 2: En la ventana de preferencias, ve a la categoría General -> Appearance.

Paso 3: Selecciona Theme y elige el tema que prefieras (por ejemplo, Dark o Light).

Paso 4: Haz clic en Apply and Close para aplicar el nuevo tema.

## 7. Cambiar los colores del terminal (consola)

Paso 1: Ve al menú Window -> Preferences.

Paso 2: En el panel izquierdo, navega a General -> Appearance -> Colors and Fonts.

Paso 3: En el panel de la derecha, expande Debug y luego selecciona Console.

Paso 4: Aquí puedes cambiar los colores de la consola (por ejemplo, fondo, texto, errores, etc.).

Paso 5: Haz clic en OK para guardar los cambios.

## 8. Comprobar si hay actualizaciones de Eclipse

Paso 1: Ve al menú Help.

Paso 2: Selecciona Check for Updates.

Paso 3: Eclipse comprobará si hay actualizaciones disponibles y te notificará si las hay.

Paso 4: Si hay actualizaciones, puedes proceder a instalarlas.

## 9. Instalar un plugin

Paso 1: Ve al menú Help -> Eclipse Marketplace....

Paso 2: En el Eclipse Marketplace, puedes buscar el plugin que necesitas escribiendo el nombre en la barra de búsqueda.

Paso 3: Una vez encontrado, haz clic en el botón Install junto al plugin.

Paso 4: Sigue las instrucciones de instalación (puede requerir reiniciar Eclipse después de instalar el plugin).

## Ejercicio 7.

DevStyle: DevStyle es un plugin popular que mejora la experiencia visual de Eclipse al proporcionar un conjunto de temas personalizables, incluido un tema oscuro.

Características:

Temas Personalizables:

Incluye varios temas oscuros, como Darkest Dark, Dark, y Light.

Puedes elegir entre diferentes esquemas de colores y estilos de iconos.

Optimización para la programación:

Ofrece una experiencia más cómoda al trabajar en proyectos de programación, mejorando la visibilidad del código y la organización de la interfaz de usuario.

Añade íconos atractivos y bien definidos, mejorando la navegación.

Vista de consola mejorada:

La consola se integra bien con el tema oscuro, mejorando la legibilidad.

Soporte de compatibilidad:

Funciona con diversas versiones de Eclipse y es completamente compatible con las últimas actualizaciones.

Configuración fácil:

Puedes cambiar entre temas de manera sencilla desde las preferencias de Eclipse.

```
1 package ejemplopractica2;
2
3 import java.util.Scanner;
4
5 public class paraimpar {
6     public static void main(String[] args) {
7         Scanner scanner = new Scanner(System.in);
8         System.out.print("Ingresa un número: ");
9         int numero = scanner.nextInt();
10        if (numero % 2 == 0) {
11            System.out.println("El número " + numero + " es par.");
12        } else {
13            System.out.println("El número " + numero + " es impar.");
14        }
15        scanner.close();
16    }
17 }
18
19
```

Aquí se ve cómo ha cambiado el fondo a más oscuro con el plugin DevStyle.

## Ejercicio 8.

### CheckStyle.

#### Características:

Verificación del estilo de código: Analiza tu código fuente para comprobar si cumple con las reglas de estilo de codificación configuradas.

Personalización: Permite configurar reglas personalizadas según los estándares de tu equipo o proyecto.

Integración directa: Marca los problemas de estilo directamente en el editor de Eclipse.

Soporte para múltiples estándares: Viene con configuraciones predefinidas como Google Java Style o Sun Coding Conventions.

Informes detallados: Proporciona un resumen de las violaciones detectadas para ayudarte a priorizar las correcciones.

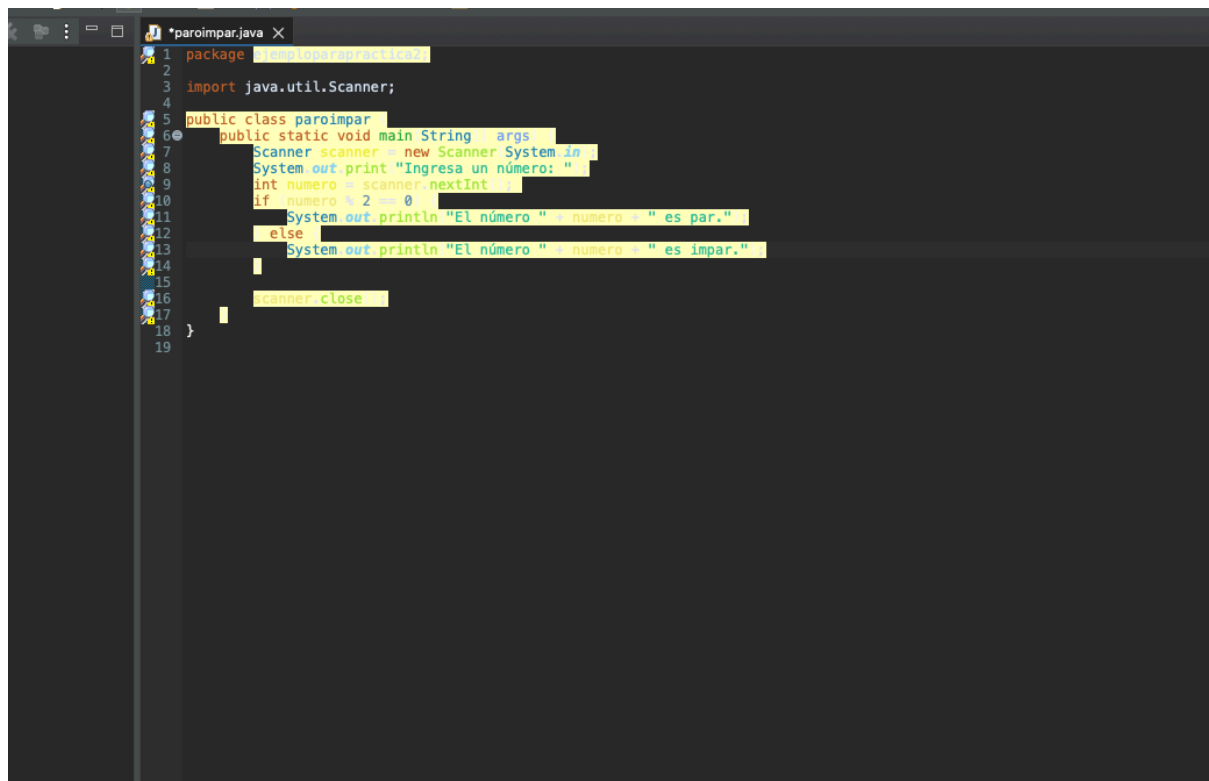
CheckStyle es útil porque:

- Mantiene la consistencia del código

- Detecta errores comunes

- Acelera revisiones

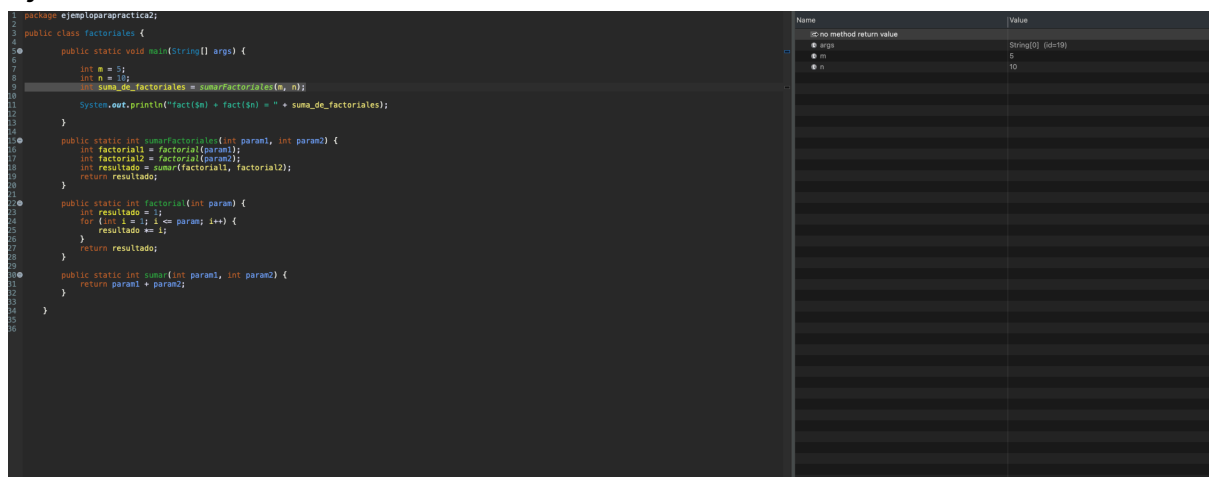
Mejora la calidad  
Ahorra tiempo



```
1 package ejemplopractica2;
2
3 import java.util.Scanner;
4
5 public class paroimpar {
6     public static void main(String[] args) {
7         Scanner scanner = new Scanner(System.in);
8         System.out.print("Ingresa un número: ");
9         int numero = scanner.nextInt();
10        if (numero % 2 == 0) {
11            System.out.println("El número " + numero + " es par.");
12        } else {
13            System.out.println("El número " + numero + " es impar.");
14        }
15    }
16    scanner.close();
17 }
18
19 }
```

Aquí se puede ver el checkstyle funcionando.

## Ejercicio 9.



```
1 package ejemplopractica2;
2
3 public class factoriales {
4     public static void main(String[] args) {
5         int n = 5;
6         int n2 = 10;
7         int suma_de_factoriales = sumaFactoriales(n, n2);
8         System.out.println("fact(" + n + ") + fact(" + n2 + ") = " + suma_de_factoriales);
9     }
10
11     public static int sumaFactoriales(int param1, int param2) {
12         int factorial1 = factorial(param1);
13         int factorial2 = factorial(param2);
14         int resultado = sumar(factorial1, factorial2);
15         return resultado;
16     }
17
18     public static int factorial(int param) {
19         int resultado = 1;
20         for (int i = 1; i <= param; i++) {
21             resultado = i;
22         }
23         return resultado;
24     }
25
26     public static int sumar(int param1, int param2) {
27         return param1 + param2;
28     }
29 }
30
31 }
```

Name	Value
no method return value	
args	String[] (ids=10)
n	5
n2	10

Aquí se puede ver el código y el valor de las variables.





Aquí se ve el resultado.

me	Value
factorial() returned	3628800
param1	5
param2	10
factorial1	120

aquí se puede ver como me he metido dentro de factorial2 y he hecho step return en el factorial como indica arriba.

	Name	Value
10	no method return value	
11	param1	120
12	param2	3628800

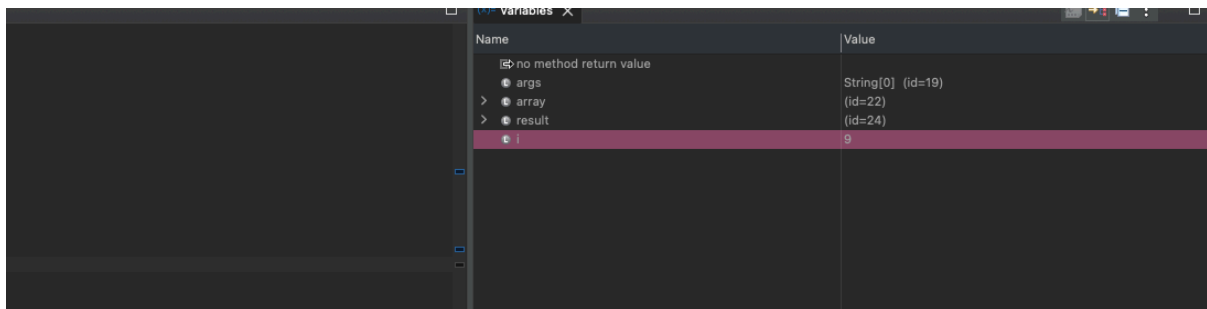
1	package ejemplopractica2;	
2	public class factoriales {	
3	4	public static void main(String[] args) {
5	6	int n = 5;
7	8	int m = 10;
9	10	int suma_de_factoriales = sumarFactoriales(m, n);
11	12	System.out.println("fact(\$n) + fact(\$n) = " + suma_de_factoriales);
13	14	}
15	16	public static int sumarFactoriales(int param1, int param2) {
17	18	int factorial1 = factorial(param1);
19	20	int factorial2 = factorial(param2);
21	22	int resultado = sumar(factorial1, factorial2);
23	24	return resultado;
25	26	}
27	28	public static int factorial(int param) {
29	30	int resultado = 1;
31	32	for (int i = 1; i <= param; i++) {
33	34	resultado = i;
35	36	}
37	38	return resultado;
39	40	}
41	42	public static int suma(int param1, int param2) {
43	44	return param1 + param2;
45	46	}

aquí se puede ver como el debugger se ha parado dentro del método sumar.

Console X
<terminated> factoriales [Java Application] [Library:Java\JavaVirtualMachines\temurin-21-jdk\Contents\Home\bin\java (13 dic 2024, 11:50:32 - 11:51:55) [pid: 2664]
fact(\$n) + fact(\$n) = 3628920

Aquí se puede ver como con el resume se muestra el resultado por consola.

## Ejercicio 10.



aquí se puede ver como la  $i = 9$ .

```
dam1@Mac-Mini-07 ejemploparapractica2 % cd src
dam1@Mac-Mini-07 src % ls
ejemploparapractica2  module-info.java
dam1@Mac-Mini-07 src % cd ejemploparapractica2
dam1@Mac-Mini-07 ejemploparapractica2 % ls
ejer04.java          factoriales.java      paraimpar.java
dam1@Mac-Mini-07 ejemploparapractica2 % javac ejer04.java
dam1@Mac-Mini-07 ejemploparapractica2 % java ejer04.
Error: no se ha encontrado o cargado la clase principal ejer04.
Causado por: java.lang.ClassNotFoundException: ejer04.
dam1@Mac-Mini-07 ejemploparapractica2 % java ejer04.java
[5, 8, 3, 4, 1, 10, 4, 0, 7, -3]
Array original: [5, 8, 3, 4, 1, 10, 4, 0, 7, -3]
Array de muestra: [3, 1, 4, 4, 0, 5, -3, 7, 8, 10]
Array ordenado: [-3, 0, 1, 3, 4, 4, 5, 7, 8, 10]
dam1@Mac-Mini-07 ejemploparapractica2 %
```

aquí se puede ver el ejercicio ejecutado desde la terminal.