

Introducción a Java

Módulo 5

Debugging

Debugging

A la hora de encontrar y resolver problemas en nuestros programas es crucial que seamos hábiles usando el **Debugger**.

Es una herramienta simple integrada en cualquier IDE que nos permite **definir un *breakpoint* (punto de parada) y ejecutar nuestro programa en modo debug (depuración)**. Al hacer esto, cuando el programa se detenga, podremos ir a paso a paso por nuestro programa y si llegamos a una línea que sea una función/procedimiento,

ejecutarla línea por línea para ir viendo cómo se modifican el valor de las variables y las acciones que realiza nuestro programa.

Características

- Deben **retornar un valor**.
- Pueden **recibir parámetros**, aunque no es obligatorio es recomendado.

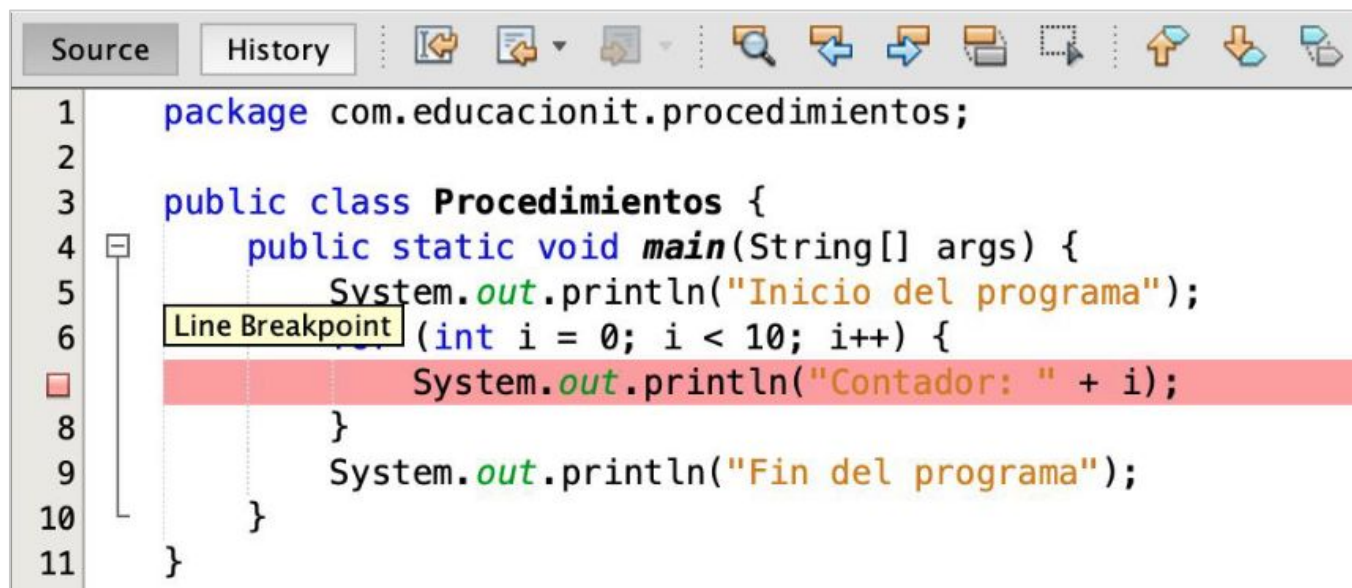
Pasos

1. Agregar uno o más *breakpoints*.
2. Iniciar el programa en modo *Debug*.
3. Ejecutar nuestro programa línea por línea.

Paso 1: Agregar *Breakpoint*

En el lado izquierdo del editor de código de la siguiente pantalla, vemos el número de cada línea. Al hacer clic sobre el número podemos indicar que deseamos agregar un *breakpoint* en la línea correspondiente.





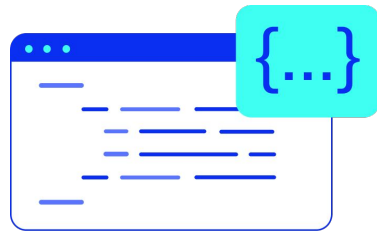
The screenshot shows an IDE window with a toolbar at the top containing icons for source, history, and various editing actions. The code editor displays the following Java code:

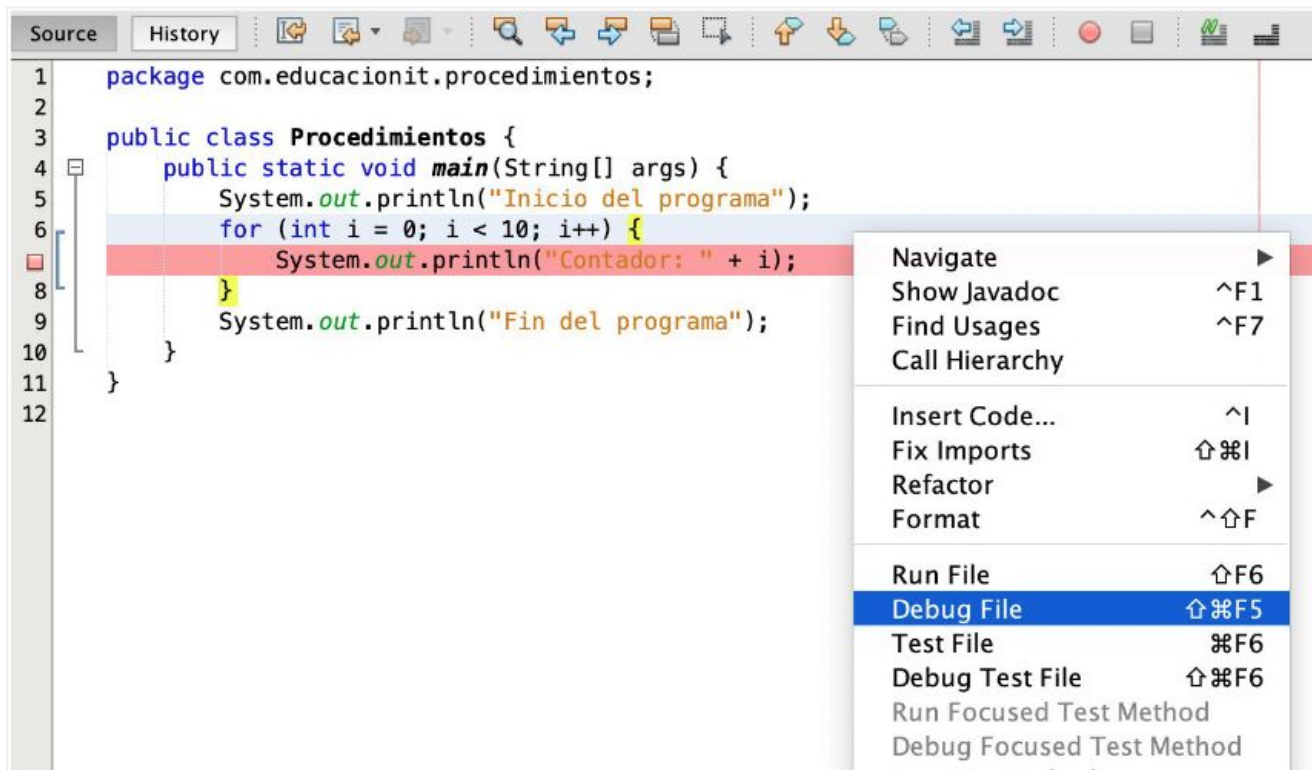
```
1 package com.educacionit.procedimientos;
2
3 public class Procedimientos {
4     public static void main(String[] args) {
5         System.out.println("Inicio del programa");
6         (int i = 0; i < 10; i++) {
7             System.out.println("Contador: " + i);
8         }
9         System.out.println("Fin del programa");
10    }
11 }
```

A line breakpoint is set on line 6, indicated by a small square icon in the left margin and a yellow box labeled "Line Breakpoint" over the code. The line of code on line 6 is highlighted in red.

Paso 2: Modo Debug

Al apretar el botón derecho para desplegar el **menú de opciones**, como se muestra en la referencia de la siguiente diapositiva, vemos una que se llama **“Debug File”**, la cual sirve para ejecutar el programa en **modo debug** (o sea, frenando el programa cuando se llega a una línea que tenga un *breakpoint*).





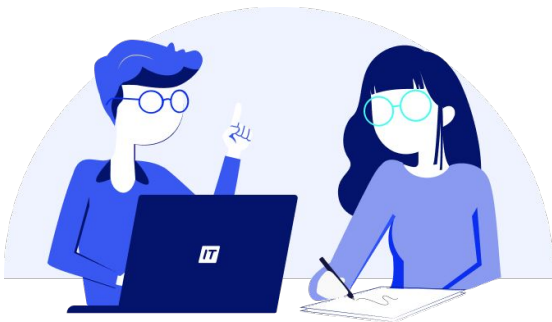
```
1 package com.educacionit.procedimientos;
2
3 public class Procedimientos {
4     public static void main(String[] args) {
5         System.out.println("Inicio del programa");
6         for (int i = 0; i < 10; i++) {
7             System.out.println("Contador: " + i);
8         }
9         System.out.println("Fin del programa");
10    }
11 }
12
```

- Navigate
- Show Javadoc [^]F1
- Find Usages [^]F7
- Call Hierarchy
- Insert Code... [^]I
- Fix Imports [⇧]⌘I
- Refactor
- Format [^]⌘F
- Run File [⇧]F6
- Debug File [⇧]⌘F5**
- Test File ⌘F6
- Debug Test File [⇧]⌘F6
- Run Focused Test Method
- Debug Focused Test Method

Variables

Al ejecutar el programa en modo *debug*, veremos algunos tabs, por ej. “Variables”, con los cuales podremos ver el valor actual de cada variable.

Si este tab no estuviera presente podemos ir a **“Window → Debugging → Variables”** (esto varía entre los distintos IDE).



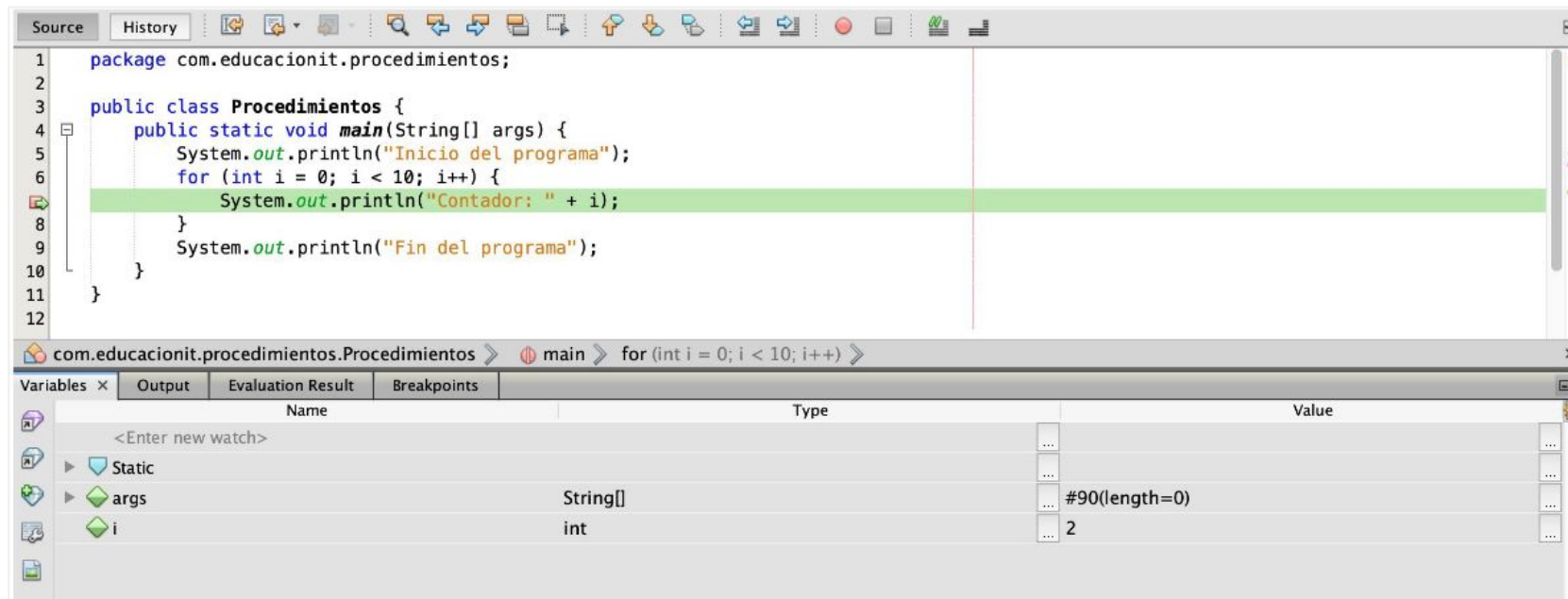
Output

Además, vemos un tab llamado “Output”, que nos va mostrando **las cosas que fue escribiendo nuestro programa a la fuente de salida**, en este caso la consola.

Si este tab no estuviera presente podemos ir a **“Window → Output”** (esto varía entre los distintos IDE).



Variables



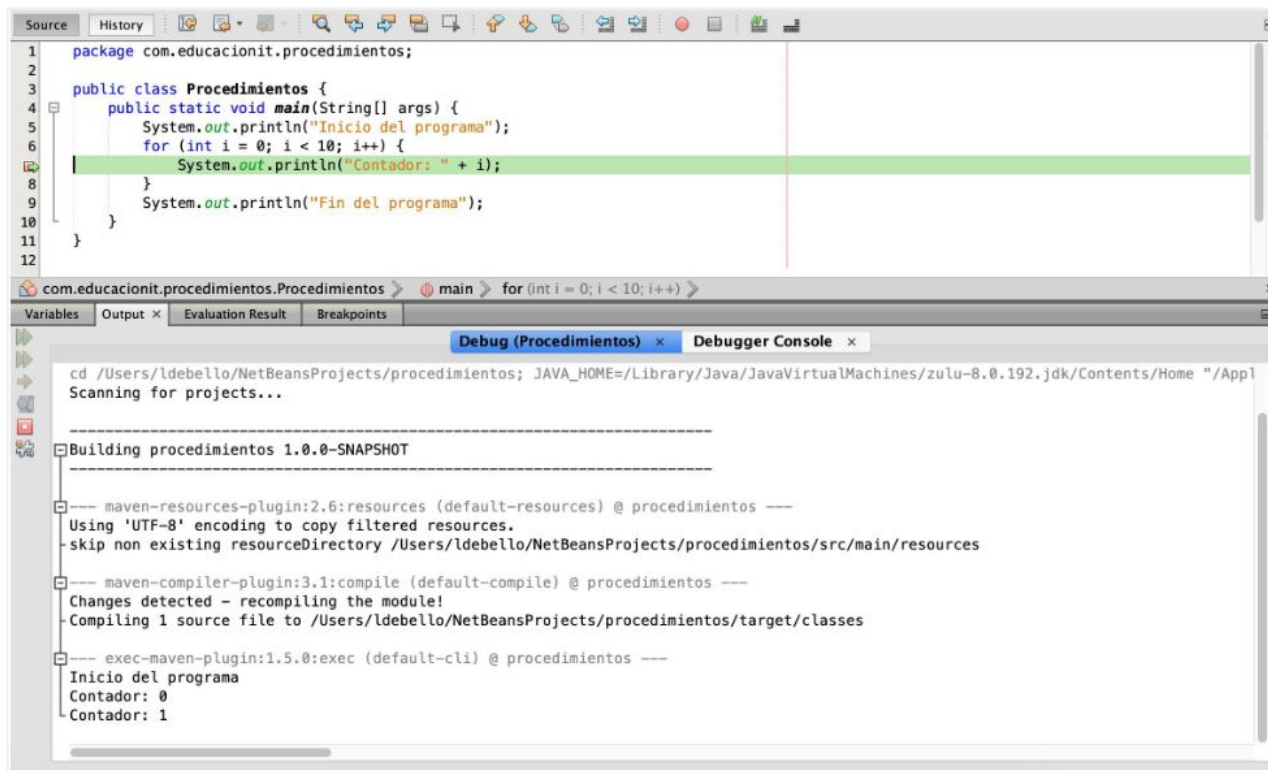
The screenshot shows an IDE with a Java source file and a variable watch window. The source file contains the following code:

```
1 package com.educacionit.procedimientos;
2
3 public class Procedimientos {
4     public static void main(String[] args) {
5         System.out.println("Inicio del programa");
6         for (int i = 0; i < 10; i++) {
7             System.out.println("Contador: " + i);
8         }
9         System.out.println("Fin del programa");
10    }
11 }
12
```

The variable watch window is open, showing the following variables:

Name	Type	Value
<Enter new watch>		
Static		
args	String[]	#90(length=0)
i	int	2

Output



The screenshot displays an IDE window with a Java source file and its execution output. The source code is as follows:

```
1 package com.educacionit.procedimientos;
2
3 public class Procedimientos {
4     public static void main(String[] args) {
5         System.out.println("Inicio del programa");
6         for (int i = 0; i < 10; i++) {
7             System.out.println("Contador: " + i);
8         }
9         System.out.println("Fin del programa");
10    }
11 }
12
```

The IDE's output window shows the following execution details:

```
cd /Users/ldebello/NetBeansProjects/procedimientos; JAVA_HOME=/Library/Java/JavaVirtualMachines/zulu-8.0.192.jdk/Contents/Home "/App1
Scanning for projects...

-----
Building procedimiento 1.0.0-SNAPSHOT
-----
--- maven-resources-plugin:2.6:resources (default-resources) @ procedimiento ---
Using 'UTF-8' encoding to copy filtered resources.
-skip non existing resourceDirectory /Users/ldebello/NetBeansProjects/procedimientos/src/main/resources

--- maven-compiler-plugin:3.1:compile (default-compile) @ procedimiento ---
Changes detected - recompiling the module!
-Compiling 1 source file to /Users/ldebello/NetBeansProjects/procedimientos/target/classes

--- exec-maven-plugin:1.5.0:exec (default-cli) @ procedimiento ---
Inicio del programa
Contador: 0
Contador: 1
```

Paso 3: Línea a línea

Depurar nos permite ejecutar nuestro programa **línea a línea**. Para ello, cada IDE utiliza algún botón o combinación de teclas para avanzar a la línea siguiente.

En NetBeans, esa tecla es **F8**. Podemos consultar esta tecla en “**Debug** → **Step Over**”.



**¡Sigamos
trabajando!**

