

# Introduction au Debugger d'Eclipse

## Qu'est ce qu'un debugger et pourquoi l'utiliser ?

Quand un programme est simple, c'est facile de trouver le bug :

- juste en relisant attentivement
- ou en plaçant quelques **System.out.print** à quelques endroits stratégiques.

Cependant **plus le programme devient complexe** et difficile à lire, **plus il devient intéressant d'utiliser un debugger.**

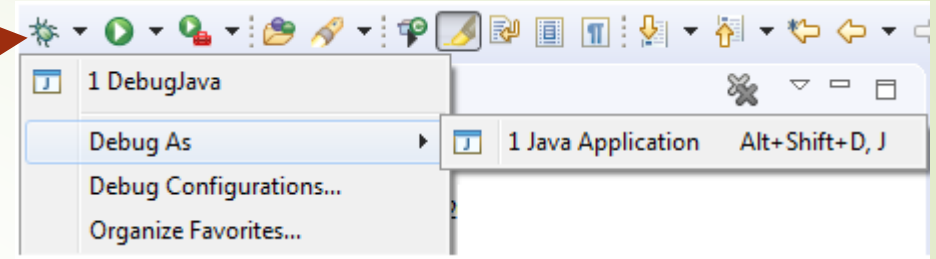
Un debugger est un **outil de développement très spécialisé qui s'attache à votre application en cours d'exécution et vous permet d'inspecter votre code.**

Le Debugger fourni dans Eclipse permet notamment :

- d'exécuter le programme ligne par ligne, en utilisant des **breakpoints** (point d'arrêts).
- de **suivre une variable** et la valeur qui lui est attribué.
- **marquer** d'une flèche l'endroit où l'exécution du programme s'est arrêté et a retourné une erreur.
- et d'autres fonctions encore...

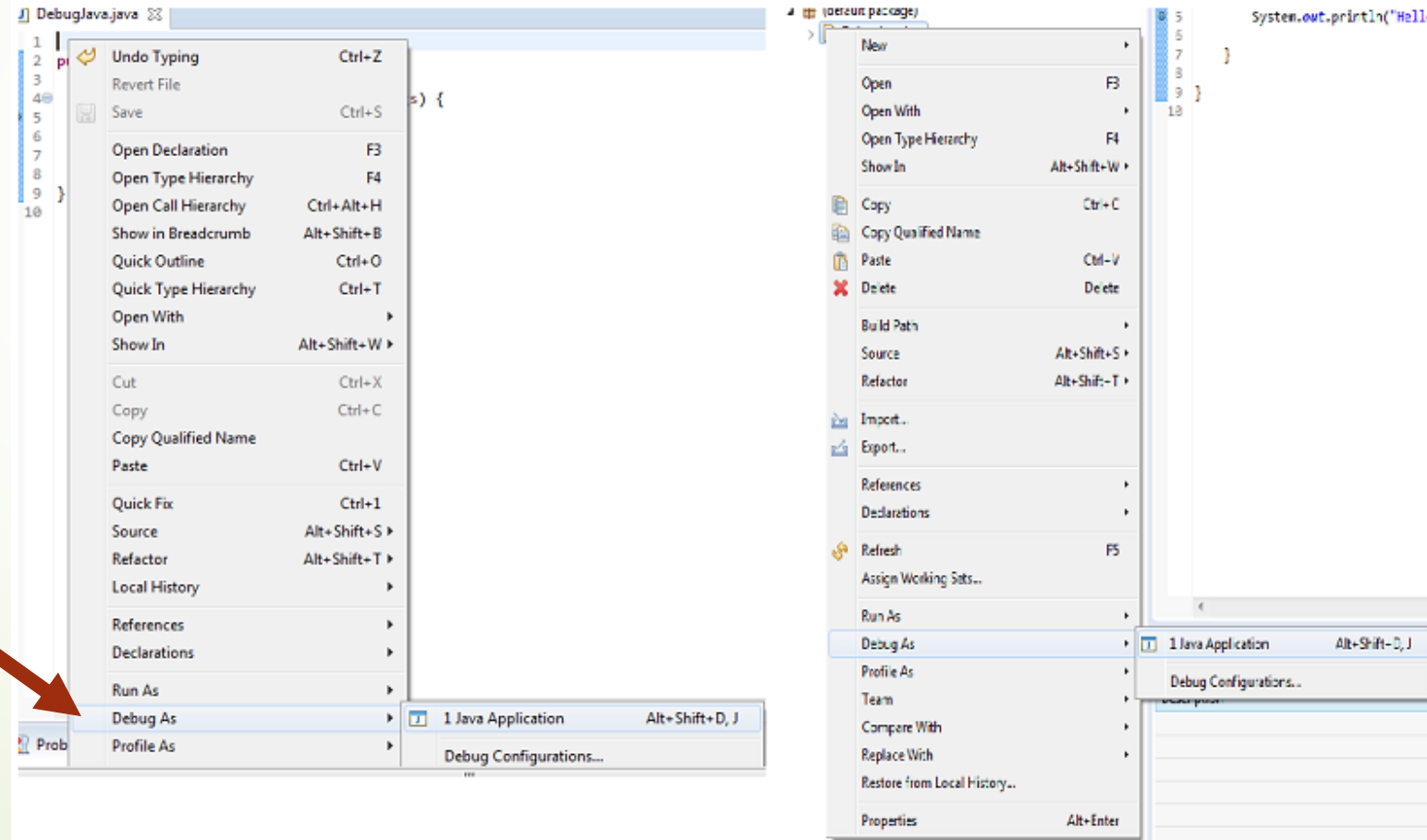
# Comment démarrer le Debugger ?

Méthode 1:  
bouton Debug




Notez que *Debug configurations* ouvre un menu avec les mêmes options que *Run configurations*, c'est-à-dire « ajouter un argument », etc..

Méthode 2:  
*Debug as -> Java Application*






Si vous cliquez sur Debug  sans avoir défini de **breakpoint**, vous remarquerez que ça lance juste l'application, comme si vous aviez cliqué sur Run. Pour lancer le debugger, il faut d'abord :

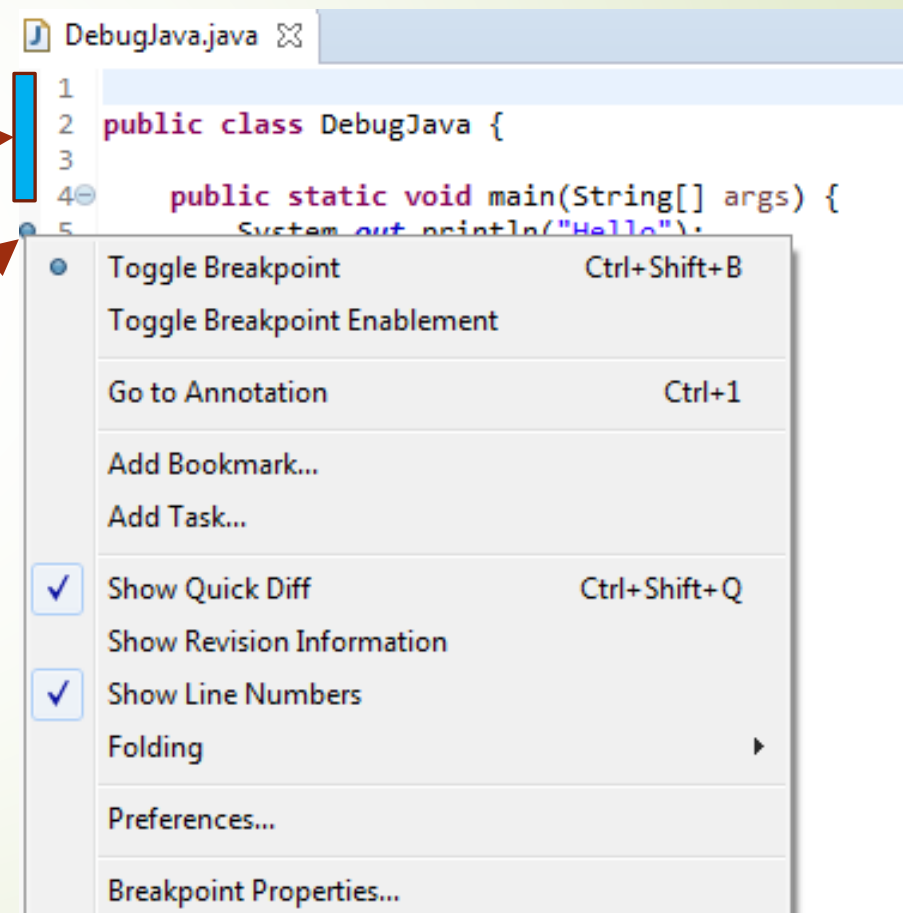
## Définir des breakpoints

**Double Clic** dans la zone à gauche des numéros de ligne

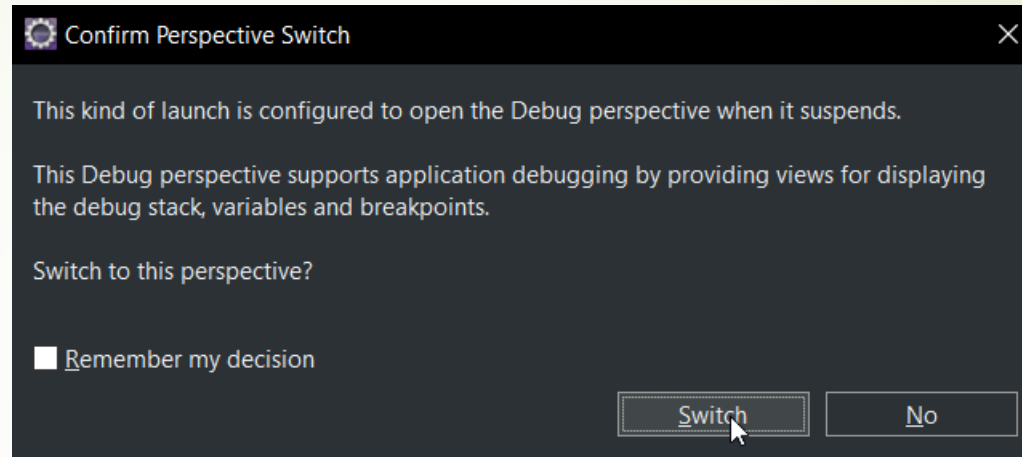
OU

**Clic droit**, puis:  
*Toggle breakpoint*

La présence d'un breakpoint sera signalée par un rond bleu [  ].



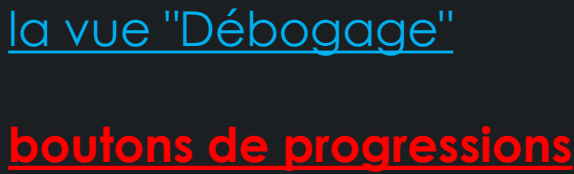
Après avoir défini un breakpoint, vous pourrez cliquer sur Debug  et la fenêtre suivante va apparaître :



Je vous conseille de cocher *Remember my decision*, puis de cliquer sur *Switch*.

Vous allez ainsi basculer sur une interface spécialement dédiée au débogage, appelée la **Perspective Debug**.

C'est tout simplement **un ensemble d'outils et de fenêtres qui vont apparaître dans l'éditeur, et qui vont nous aider à déboguer.**



## Les vues variables, breakpoints et expressions

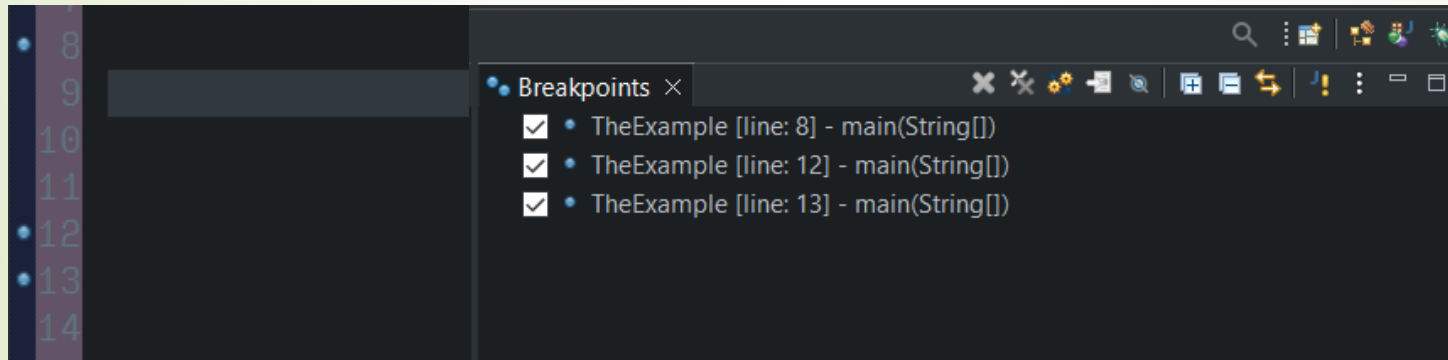


## Revenons aux breakpoints. A quoi servent-ils ?

**Un breakpoint est un signal qui dit au Debugger de temporairement suspendre l'exécution de mon programme à un point précis du code.**





Par exemple, si je place un breakpoint à la ligne 22, et que je clique sur Debug, mon application ne s'exécutera que jusqu'à la ligne 22 (exclue).

On peut placer plusieurs breakpoints, à des endroits stratégiques, qui nous permettront, en avançant pas à pas, d'observer les changements survenus d'un breakpoint à l'autre.







**La vue breakpoint:**  
permet de suivre d'un coup d'œil tous les breakpoints, les désactiver, etc..

Avancer pas à pas,  
avec les **boutons de progressions** :



F8   Stop   F5   F6   F7

| Shortcut         | Toolbar   |   |
|------------------|---|---|
| F5 (Step Into)   |  | <b>Avance jusqu'à la prochaine ligne.</b> Si la prochaine ligne est un appel de méthode, <b>rentre dans la dite méthode.</b>            |
| F6 (Step Over)   |  | <b>Avance jusqu'à la prochaine ligne.</b> Si la prochaine ligne est un appel de méthode, <b>exécute la méthode sans rentrer dedans.</b> |
| F7 (Step Return) |  | <b>Lorsqu'on est à l'intérieur d'une méthode,</b> permet de sauter toutes les étapes qu'elle contient jusqu'au <b>Return</b>            |
| F8 (Resume)      |  | <b>Continue l'exécution jusqu'au prochain breakpoint (ou jusqu'à la fin du programme</b> si aucun autre breakpoint n'a été défini.      |

```
1 public class DebuggingExample {
2
3     public static void main(String[] args) {
4
5         int firstInt = 7;
6         int secondInt = 8;
7
8         int sum = add(firstInt, secondInt);
9
10        System.out.println("The sum is: " + sum);
11    }
12
13    private static int add(int int1, int int2) {
14        int sum = int1 + int2;
15        return sum;
16    }
17 }
18
19
20
```

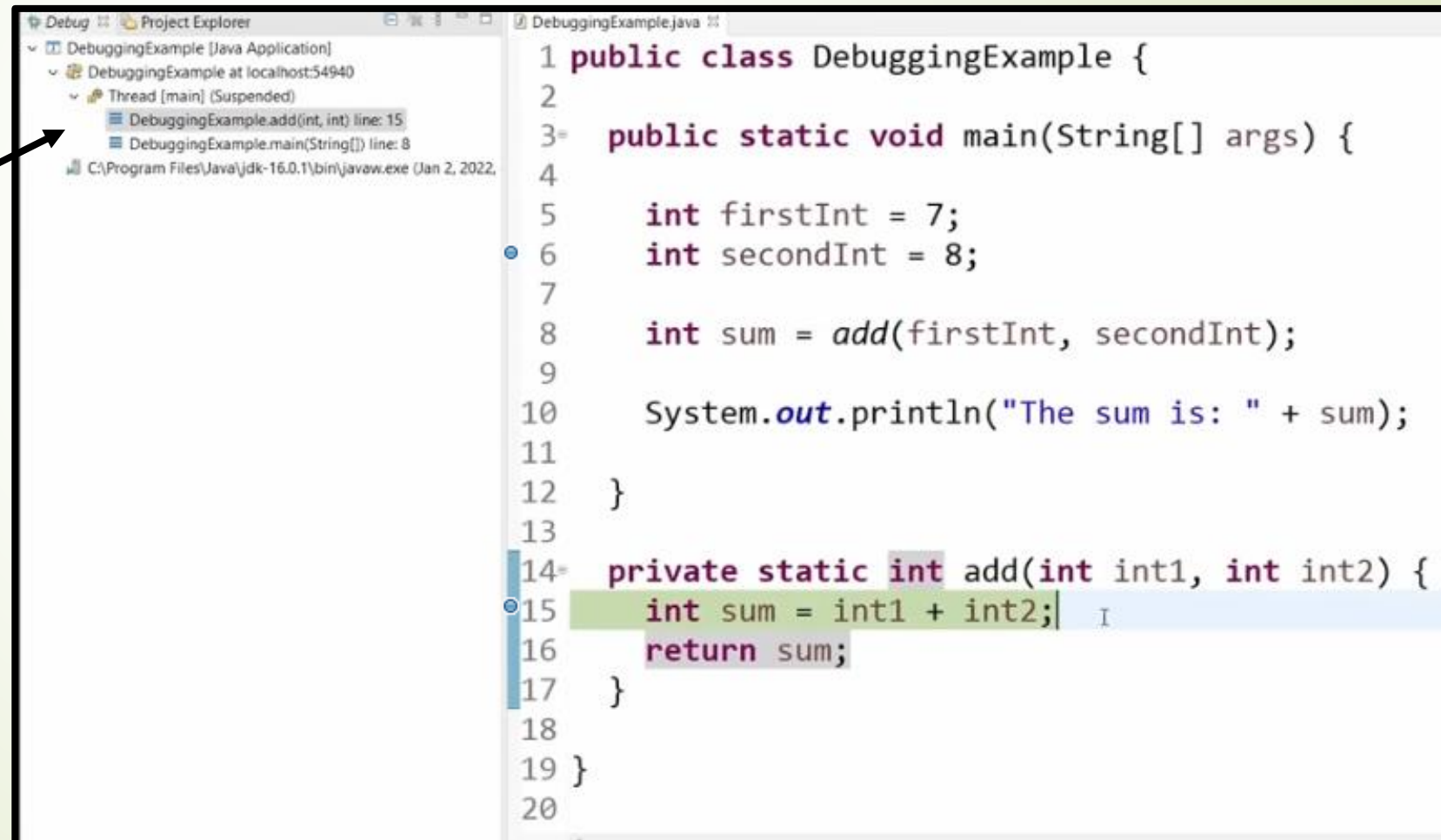
## La vue Debogage: affiche sous la forme d'une arborescence, les différents processus en cours d'exécution ou terminés.

On peut avoir plusieurs processus de débogage en cours:

- dans différents projets,
- mais aussi dans une même classe.

Dans cet exemple de vue Debogage, on voit que l'exécution en est actuellement à la Ligne 15.

On voit aussi que la Ligne 15 est placée au dessus du processus Ligne 8, ce qui signifie qu'on est à l'intérieur de la méthode appelée ligne 8.





## La vue Variables

```
1 public class TheExample {
2
3
4
5 public static void main(String[] args) {
6     int[] arr = {20, 30, 40, 50};
7
8     int value = 10;
9
10
11
12 for (int i=0; i<arr.length; i++) {
13     value = arr[i];
14 }
15
16
17
18 int[] anotherArr = arr;
19
20
21
22 System.out.println(value);
23 System.out.println(anotherArr);
24 }
25
26 }
```

| Name                   | Value             |
|------------------------|-------------------|
| no method return value |                   |
| args                   | String[0] (id=20) |
| arr                    | (id=22)           |
| value                  | 10                |

Elle permet de **suivre** en un coup d'œil **les variables** de mon programme (**nom et valeur**).  
Lorsque la variable est un **Objet**, elle a en plus un **ID unique**.



La vue Variables ne montre **que les variables et les changements survenus avant le breakpoint** où vient de s'arrêter l'exécution.

L'endroit où je me trouve (càd où s'est arrêtée l'exécution) est indiquée par une flèche bleue ➡

Si, durant l'exécution, la valeur d'une variable déclarée plus haut change au moins une fois, elle sera **surlignée en jaune dans la vue Variables**. (ou en rose si vous utilisez le Dark Theme)

# Vidéo de démonstration générale + suivi de l'évolution des variables dans la vue Variables

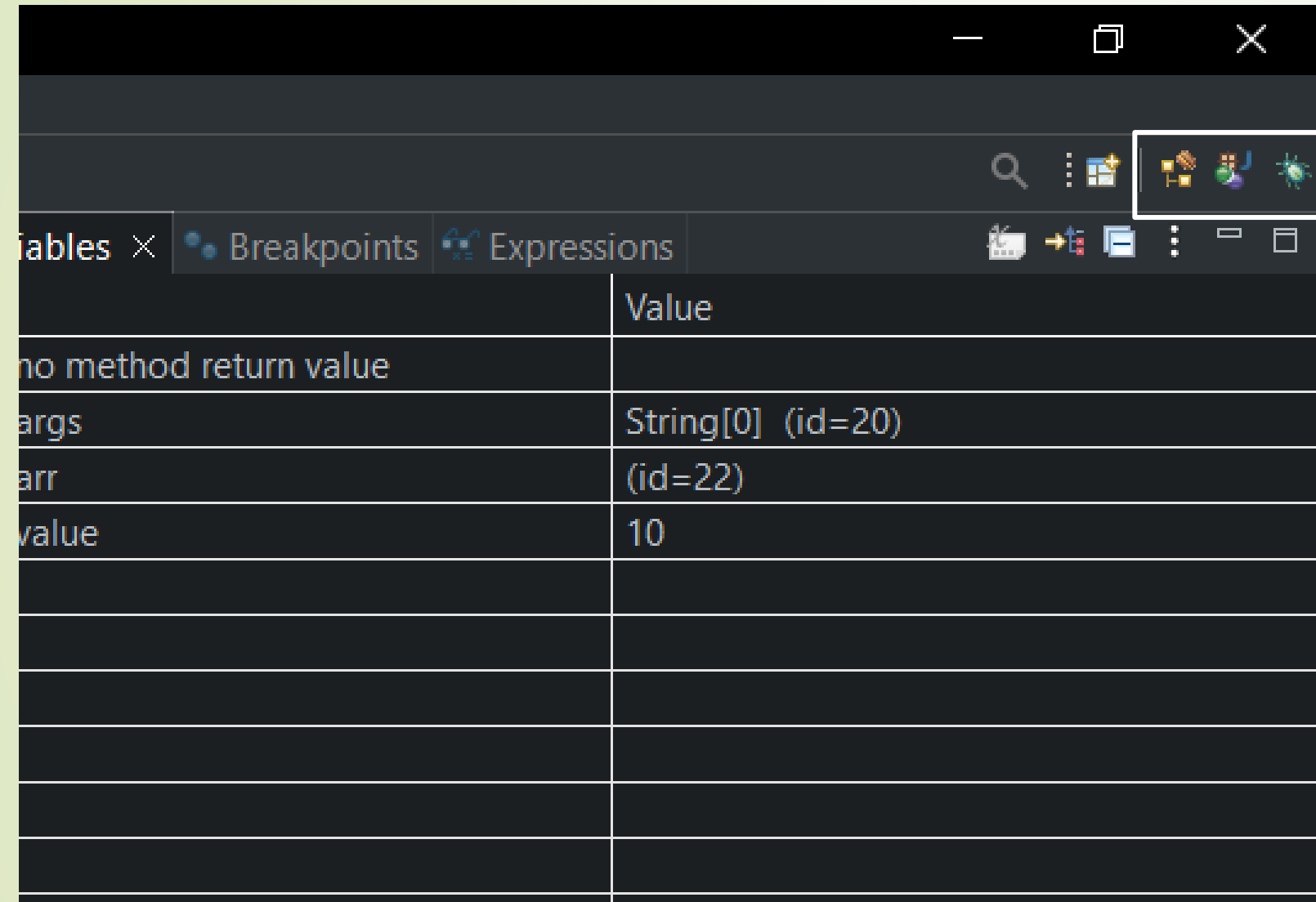
The screenshot displays the Eclipse IDE interface. The Package Explorer on the left shows a project named 'DebugExample' with several sub-packages and source files. The 'TheExample.java' file is open in the editor, showing the following code:

```
1  
2 public class TheExample {  
3  
4  
5     public static void main(String[] args) {  
6         int[] arr = {20, 30, 40, 50};  
7  
8         int value = 10;  
9  
10  
11  
12         for (int i=0; i<arr.length; i++) {  
13             value = arr[i];  
14         }  
15  
16  
17  
18         int[] anotherArr = arr;  
19  
20  
21  
22         System.out.println(value);  
23         System.out.println(anotherArr);  
24     }  
25  
26 }
```

The Console window at the bottom shows the output of the program:

```
<terminated> TheExample [Java Application] C:\Users\De-albuquerqueD\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre\bin\javaw.exe (26 janv. 2023, 1:  
50  
[I@7a30d1e6
```

## Sortir de la perspective Debug/Retourner à la perspective Java



En haut à droite d'Eclipse,  
Il y a des icones qui permettent  
de switcher d'une perspective  
à l'autre.

Ici, de gauche à droite,  
- la Perspective JavaEE,  
- La perspective Java  
- La perspective Debug

Si vous cliquez sur la perspective  
Java, vous retrouverez vos  
fenêtres habituelles.

## Ressources

Article par la communauté d'Eclipse:

### **Beginner's Guide to Quick Start Debugging**

[https://www.eclipse.org/community/eclipse\\_newsletter/2017/june/article1.php](https://www.eclipse.org/community/eclipse_newsletter/2017/june/article1.php)

Vidéo: Eclipse Debugger Full Tutorial, par Coding with John :

<https://www.youtube.com/watch?v=aqcJsKdjvU>

Déboguer du code Java avec Eclipse, par JM Doudoux :

<https://www.jmdoudoux.fr/java/dejae/chap008.htm>