

Debug no Eclipse

MC302 EF

Programação Orientada a Objetos

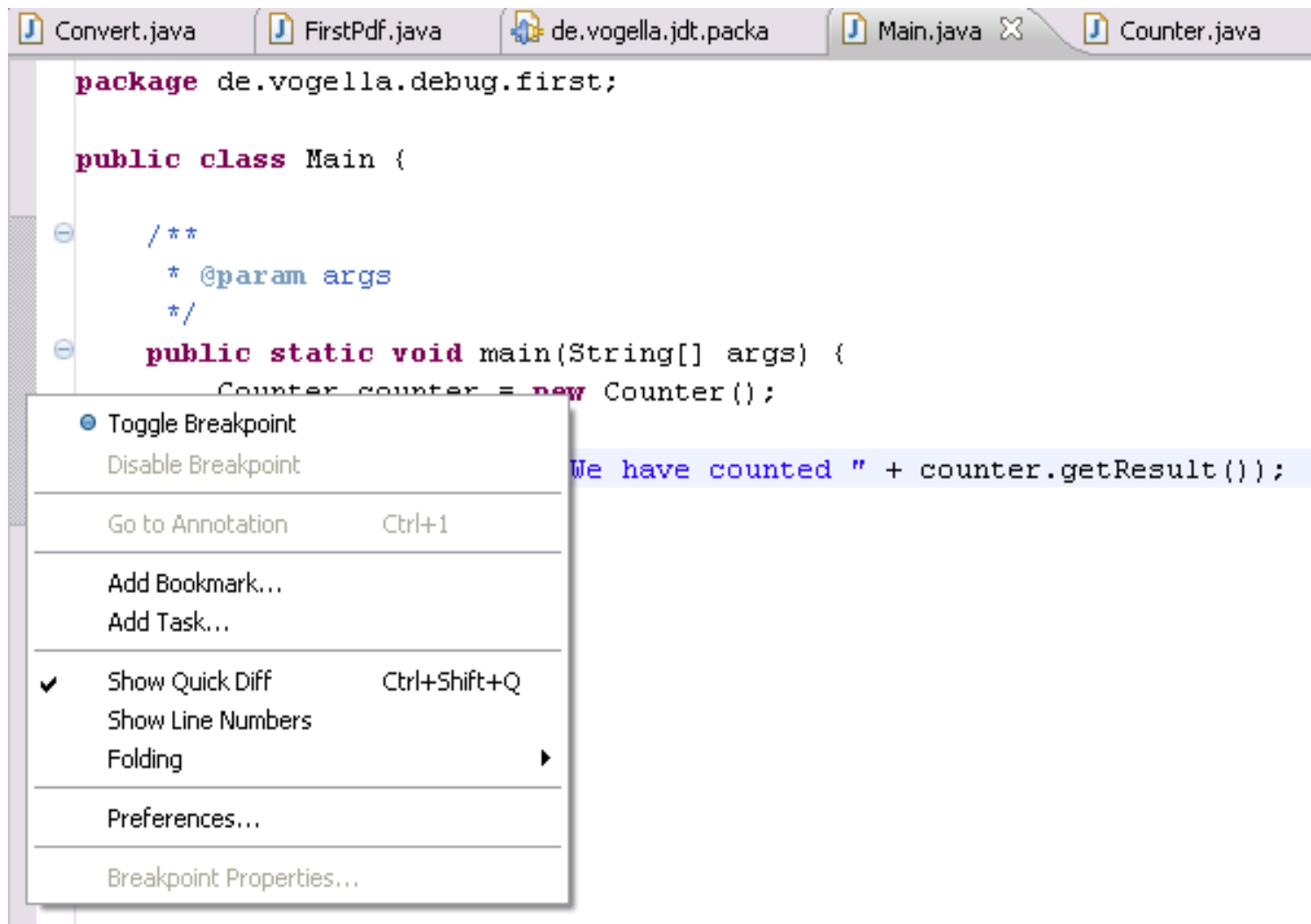
PED: Lucas Augusto Carvalho
lucas.carvalho@ic.unicamp.br

Prof. Fernando Vanini

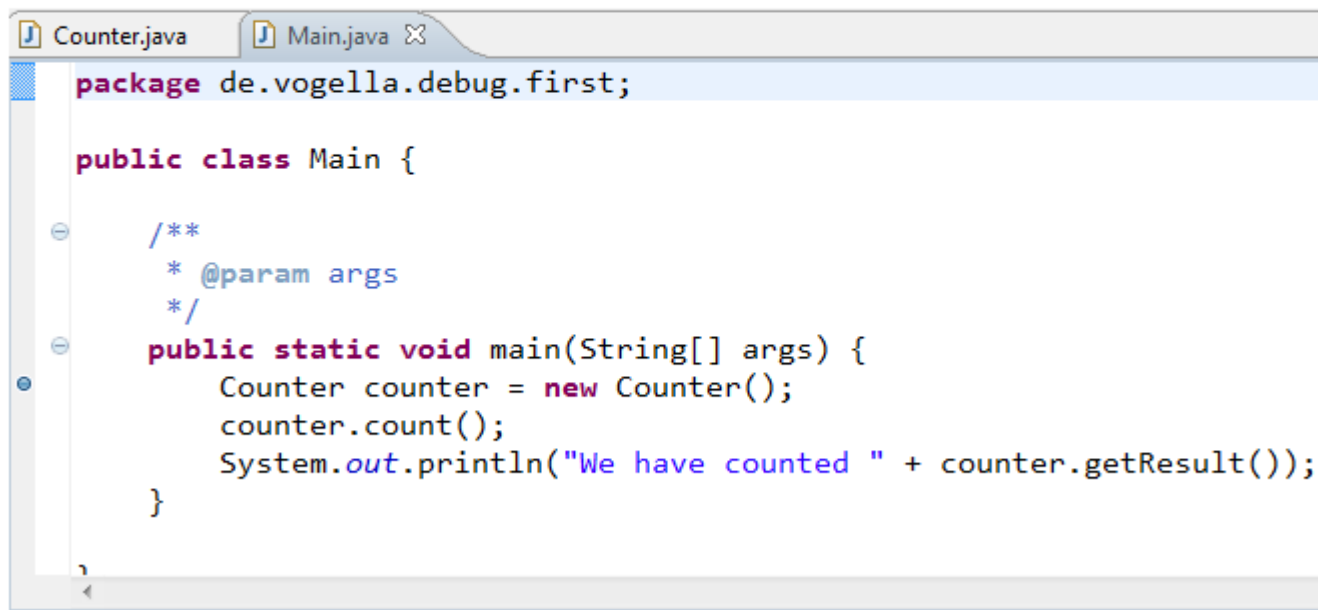
O que é Depuração?

- Permite a execução iterativamente de programas enquanto você observa o código fonte e as variáveis durante a execução.
- Breakpoint: ponto de parada na execução

Definindo Breakpoints



Definindo Breakpoints



```
Counter.java Main.java X
package de.vogella.debug.first;

public class Main {
    /**
     * @param args
     */
    public static void main(String[] args) {
        Counter counter = new Counter();
        counter.count();
        System.out.println("We have counted " + counter.getResult());
    }
}
```

Iniciando a Depuração

The screenshot displays an IDE interface with the following components:

- Package Explorer:** Shows a project named 'de.vogella.debug.first' with a source folder 'src' containing a package 'de.vogella.debug.first' and files 'Counter.java' and 'Main.java'.
- Context Menu:** Open over 'Main.java', listing actions such as 'New', 'Open', 'Copy', 'Paste', 'Delete', 'Refresh', and 'Debug As'. The 'Debug As' option is highlighted, and its sub-menu is also open, showing '1 Debug on Server' and '2 Java Application'.
- Main Editor:** Displays the source code of the 'Main' class:

```
package de.vogella.debug.first;

public class Main {

    /**
     * @param args
     */
    public static void main (String[] args) {
        Counter counter = new Counter ();
        counter.count ();
        System.out.println ("Counter: " + counter.getCount ());
    }
}
```
- Toolbar:** Located at the bottom, it includes icons for File, Edit, Source, Navigate, Search, Project, Git, Run, Window, and Help. The 'Debug' icon (a bug) is highlighted with a red box.

Perspectiva de Depuração

The screenshot shows the Eclipse IDE in debug mode. The main window displays the source code of `Main.java` with a breakpoint at line 9. The Variables view shows the `args` variable as a `String[]` (id=16). The Outline view shows the class structure. The Console view is empty.

```
package de.vogella.debug.first;

public class Main {

    /**
     * @param args
     */
    public static void main(String[] args) {
        Counter counter = new Counter();
        counter.count();
        System.out.println("We have counted " + counter.getResult());
    }
}
```

Name	Value
args	String[0] (id=16)

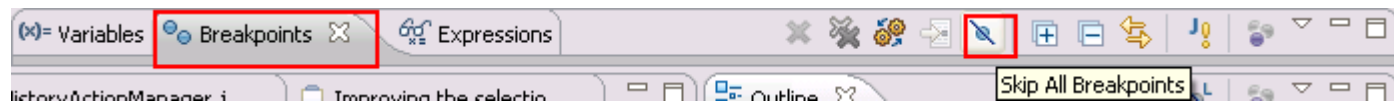
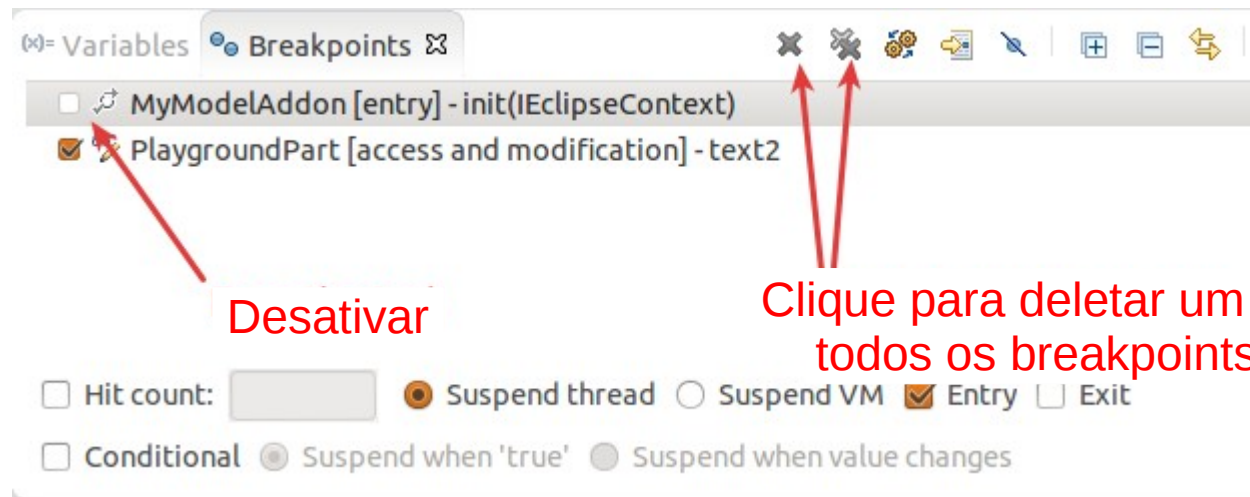
de.vogella.debug.first

- Main
 - main(String[]): void

Main (1) [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (06.07.2009 11:30:57)

Writable Smart Insert 9 : 1

Gerenciando os Breakpoints



Execução da Depuração

F5 – executa a linha corrente e passa para a próxima linha. Se a linha selecionada é uma chamada de método, o depurador executa o código associado.

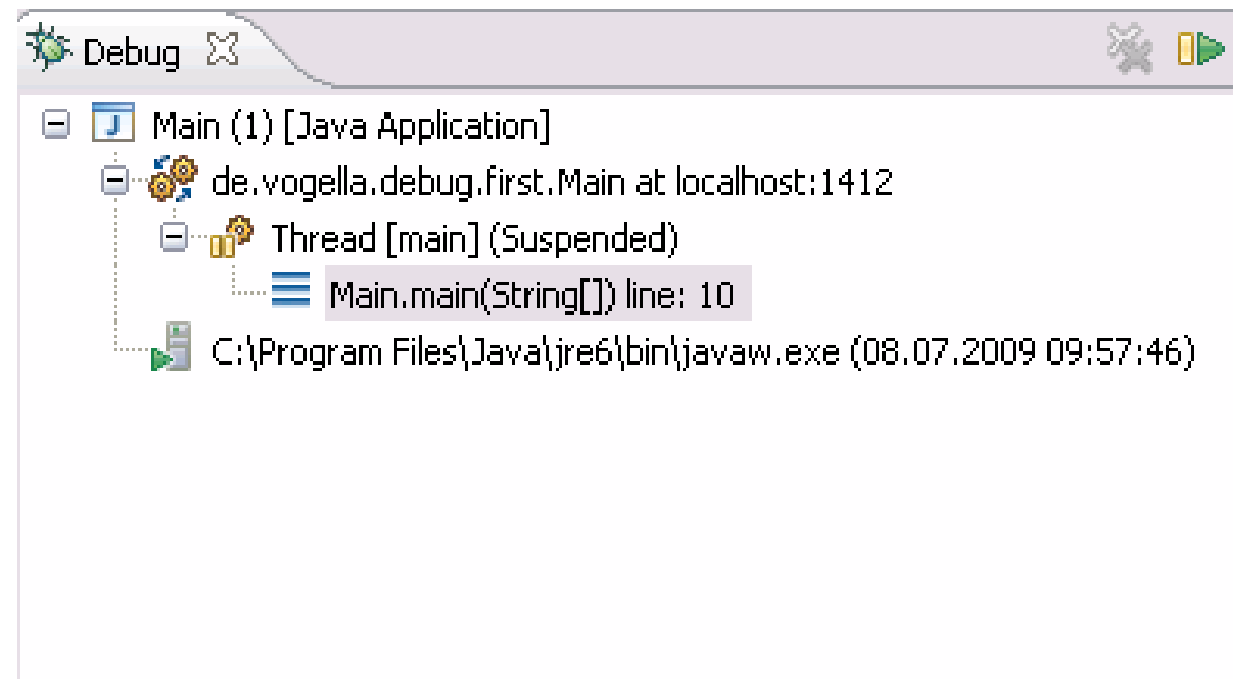
F6 – prossegue sobre uma chamada de método, ou seja, executa um método sem passar o depurador por cada linha do seu código.

F7 – prossegue para o invocador do método atualmente em execução. Isto finaliza a execução do método corrente e retorna para o invocador deste método.

F8 – informa ao depurador para continuar a execução do código do programa até que alcance o próximo breakpoint ou watchpoint.



Pilha de Chamadas



Variáveis

Name	Value
args	String[0] (id=16)
counter	Counter (id=20)
result	0

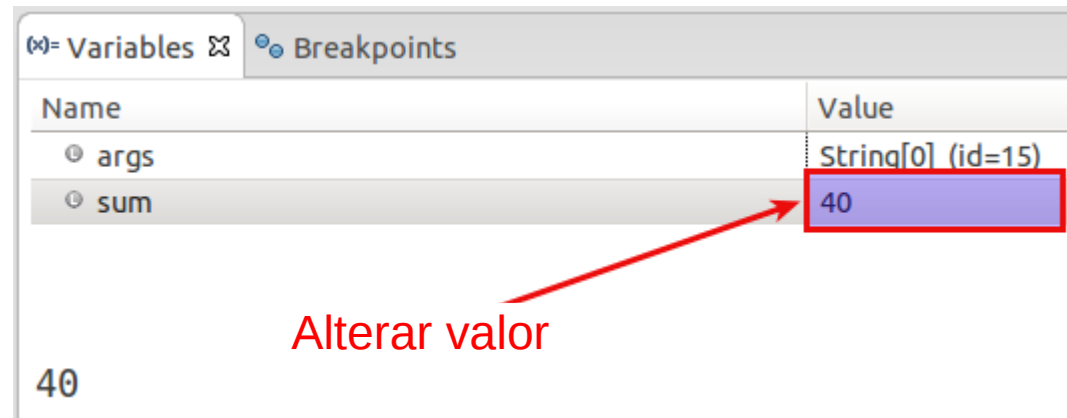
Name	Value	Layout
args	String	
counter	Count	
result	0	

- Show Constants
- Show Static Variables
- Show Qualified Names
- Show Null Array Entries
- Show References
- Java Preferences...

de.vogella.debug.first.Counter@587c94

- Vertical View Orientation
- Horizontal View Orientation
- Variables View Only
- Show Columns
- Select Columns...

Modificação de Variáveis

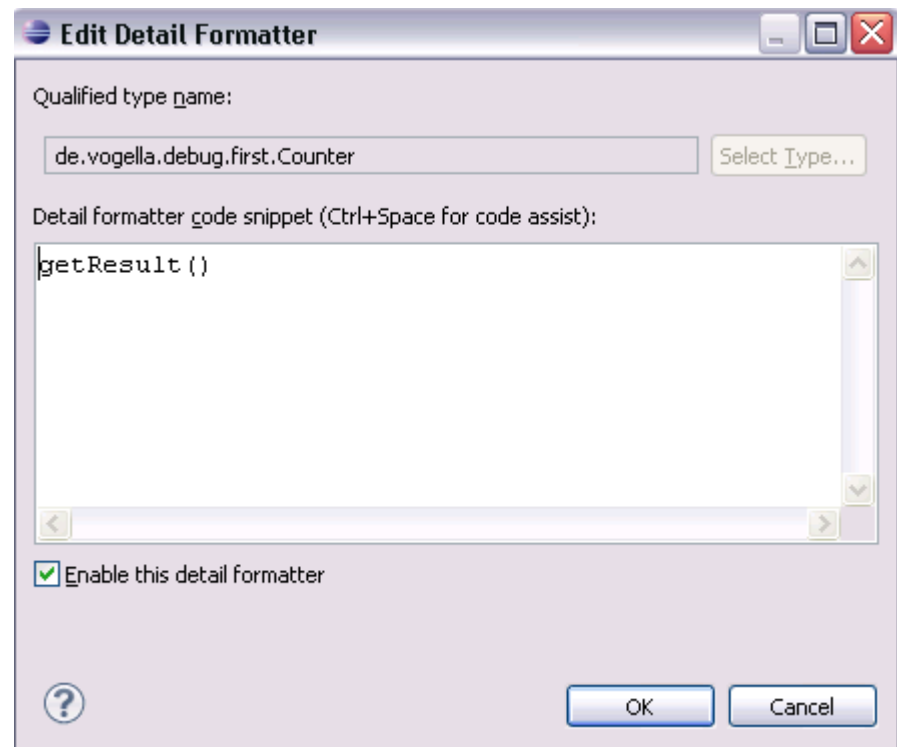
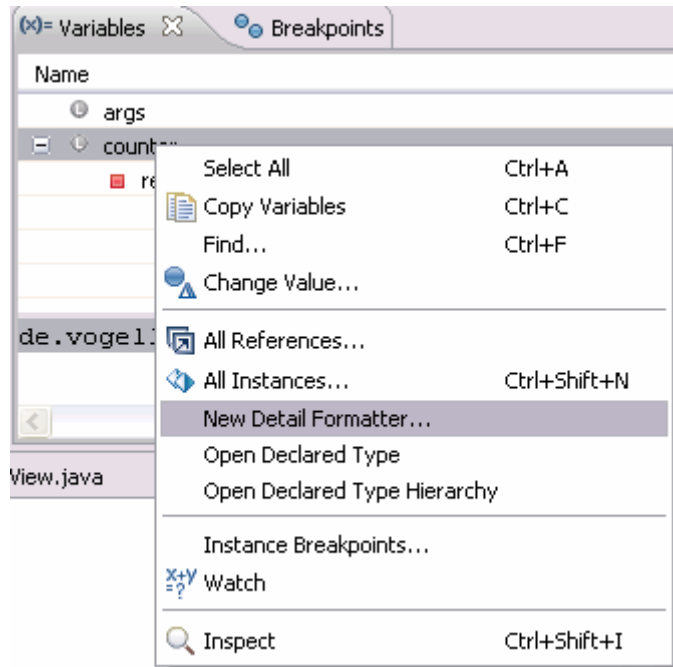


The image shows a debugger's Variables window. The window has two tabs: "Variables" (selected) and "Breakpoints". Below the tabs is a table with two columns: "Name" and "Value".

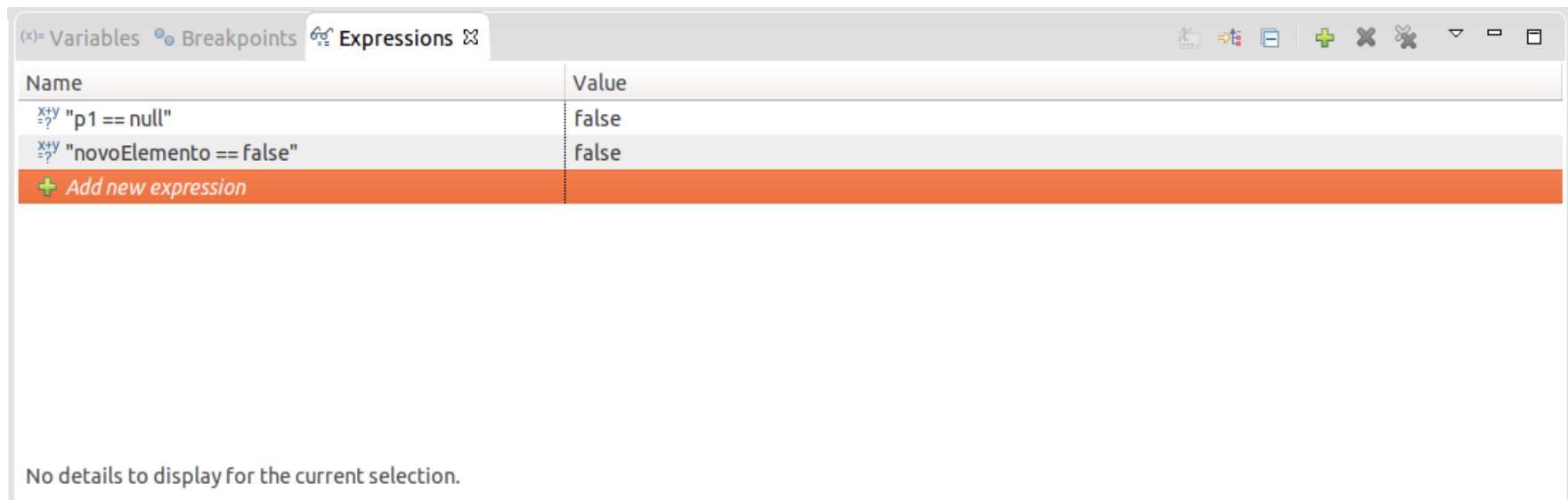
Name	Value
args	String[0] (id=15)
sum	40

A red arrow points from the text "Alterar valor" to the value "40" in the "sum" row. The value "40" is highlighted with a red border. Below the table, the number "40" is displayed in a separate box.

Visualização de Variáveis



Expressões



The screenshot shows a debugger window with the 'Expressions' tab selected. The window title is '(x) Variables Breakpoints Expressions'. The main area contains a table with two columns: 'Name' and 'Value'. The first row shows the expression '"p1 == null"' with a value of 'false'. The second row shows the expression '"novoElemento == false"' with a value of 'false'. Below the table is an orange button with a plus sign and the text 'Add new expression'. At the bottom of the window, it says 'No details to display for the current selection.'

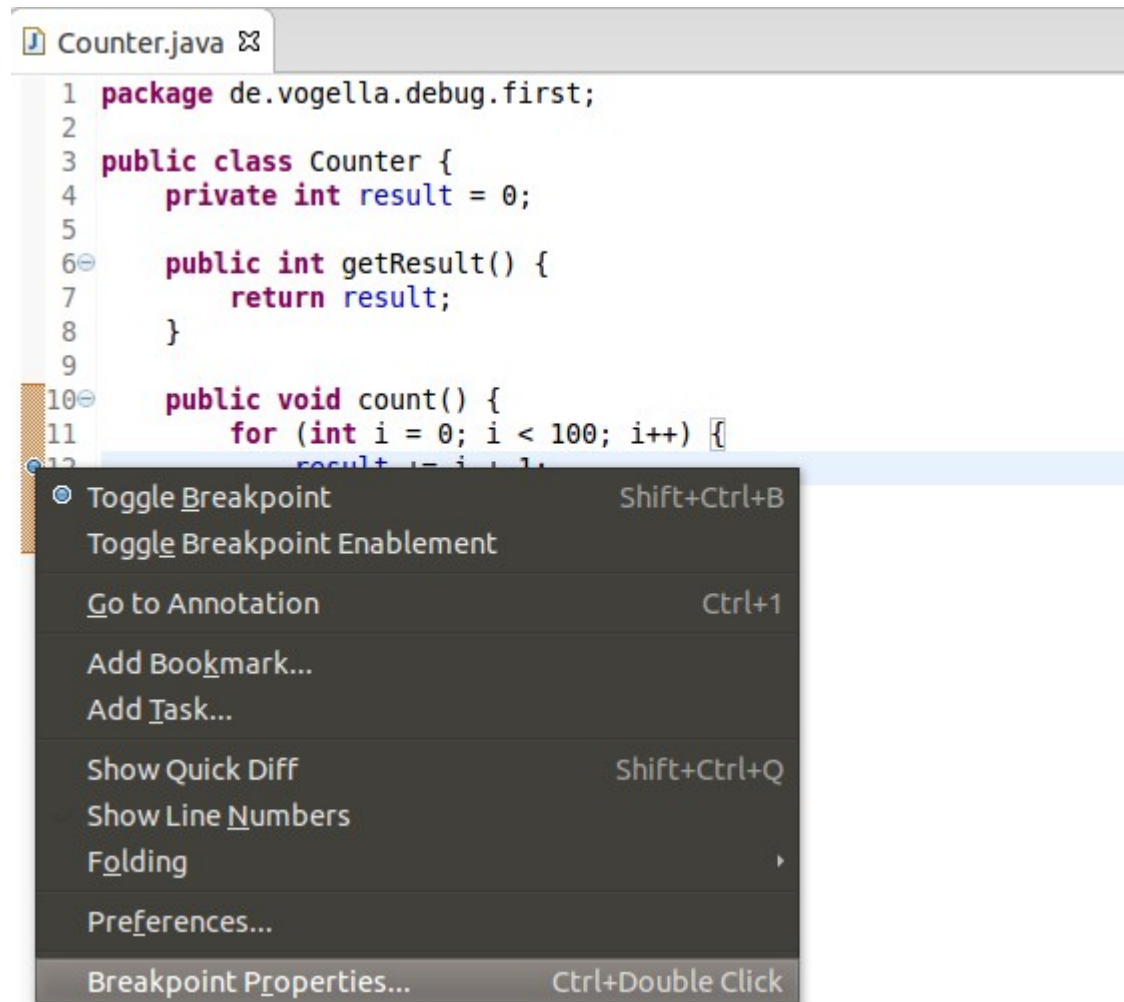
Name	Value
<code>"p1 == null"</code>	false
<code>"novoElemento == false"</code>	false

[+ Add new expression](#)

No details to display for the current selection.

Propriedades do Breakpoint

- Condições



The screenshot shows a code editor window titled "Counter.java" with the following code:

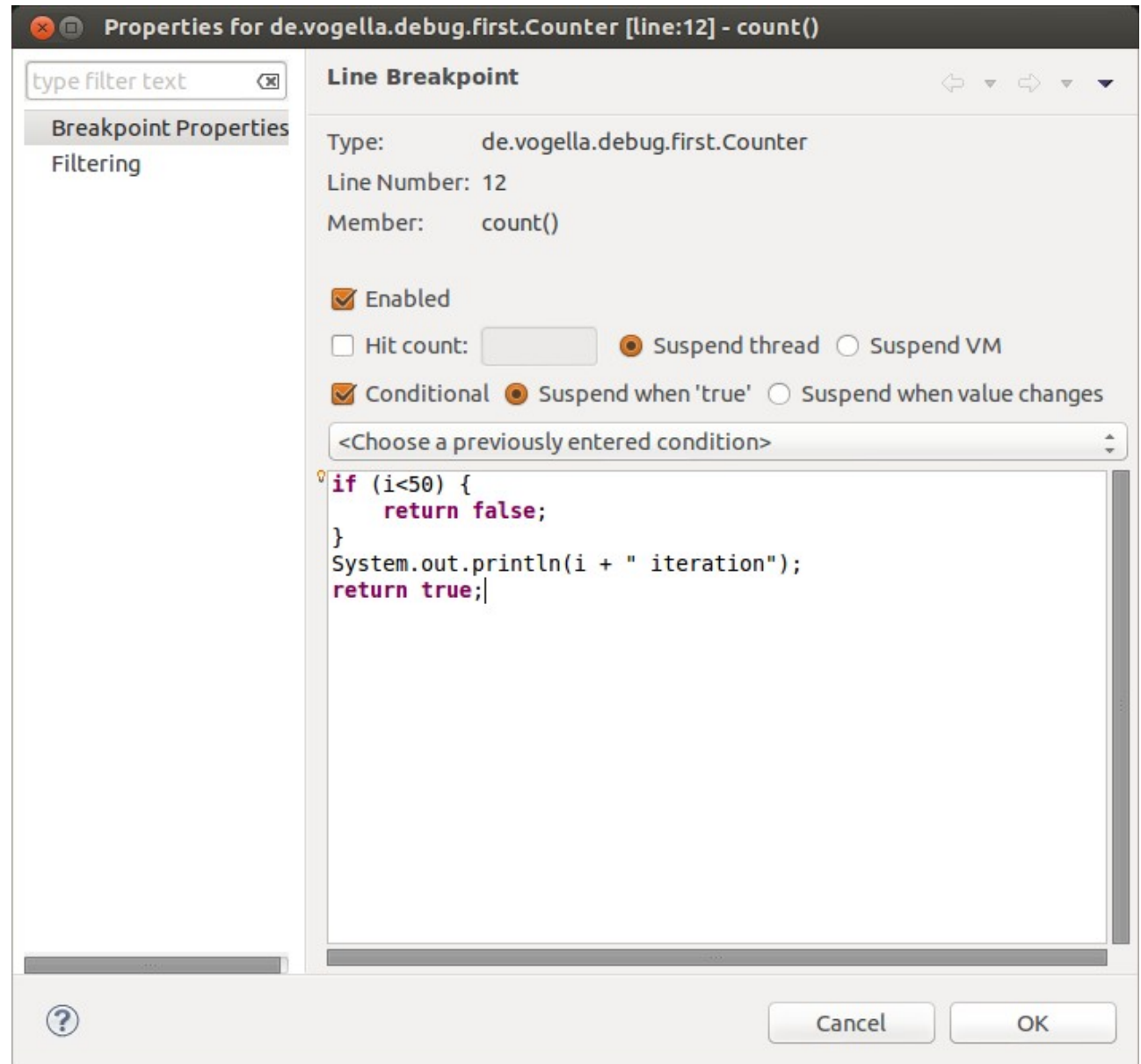
```
1 package de.vogella.debug.first;
2
3 public class Counter {
4     private int result = 0;
5
6     public int getResult() {
7         return result;
8     }
9
10    public void count() {
11        for (int i = 0; i < 100; i++) {
12            result = i + 1;
13        }
14    }
15 }
```

A breakpoint is set on line 12. A context menu is open over the breakpoint, listing the following options:

- Toggle Breakpoint (Shift+Ctrl+B)
- Toggle Breakpoint Enablement
- Go to Annotation (Ctrl+1)
- Add Bookmark...
- Add Task...
- Show Quick Diff (Shift+Ctrl+Q)
- Show Line Numbers
- Folding
- Preferences...
- Breakpoint Properties... (Ctrl+Double Click)

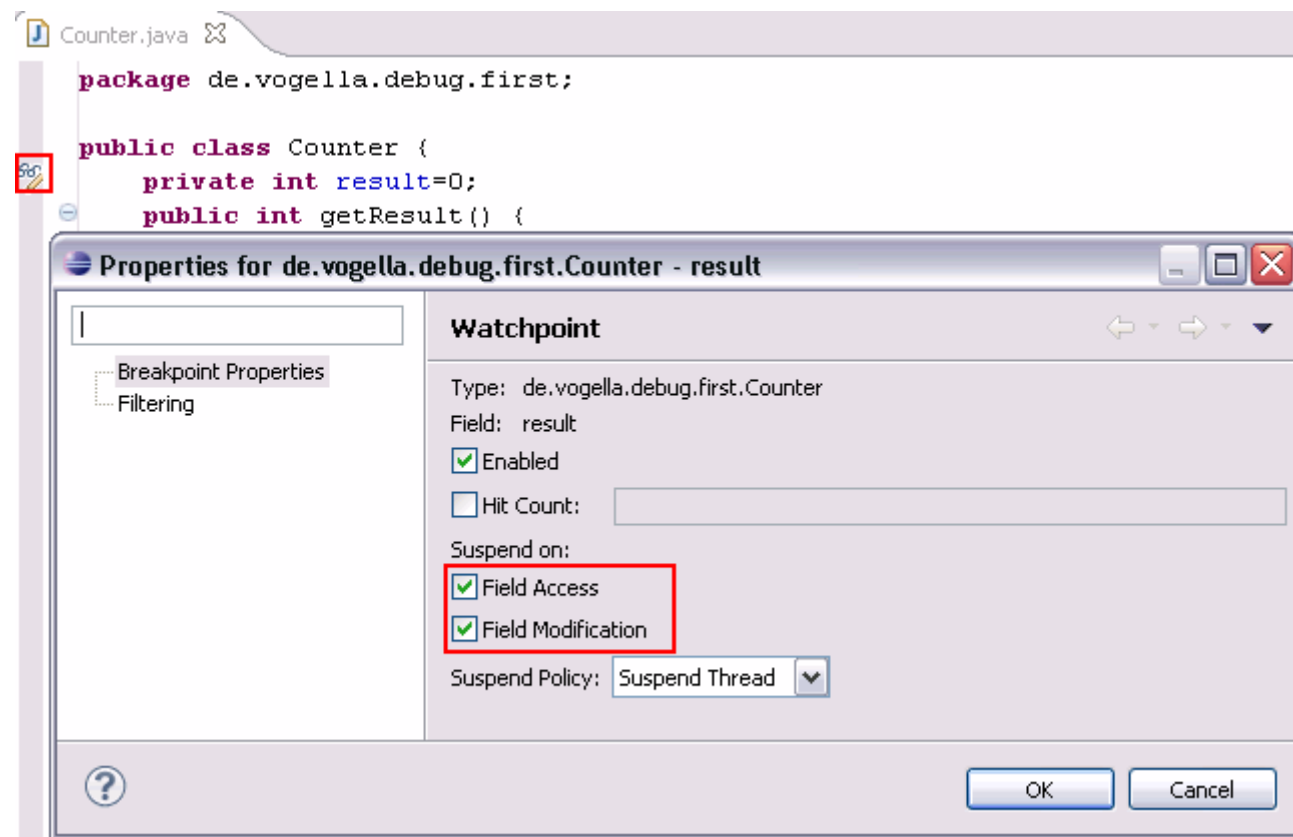
Propriedades do Breakpoint

- Condições

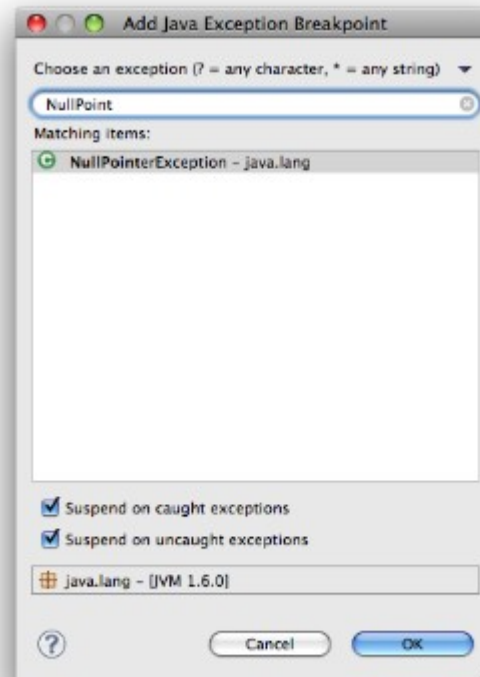


Propriedades do Watchpoint

- Watchpoint: um breakpoint definido em um campo



Exception Breakpoints



Method Breakpoints

The screenshot shows an IDE window with a Java code snippet and a dialog box for configuring a method breakpoint. The code is as follows:

```
public void count() {  
    for (int i = 0; i < 100; i++) {  
        result += i + 1;  
    }  
}
```

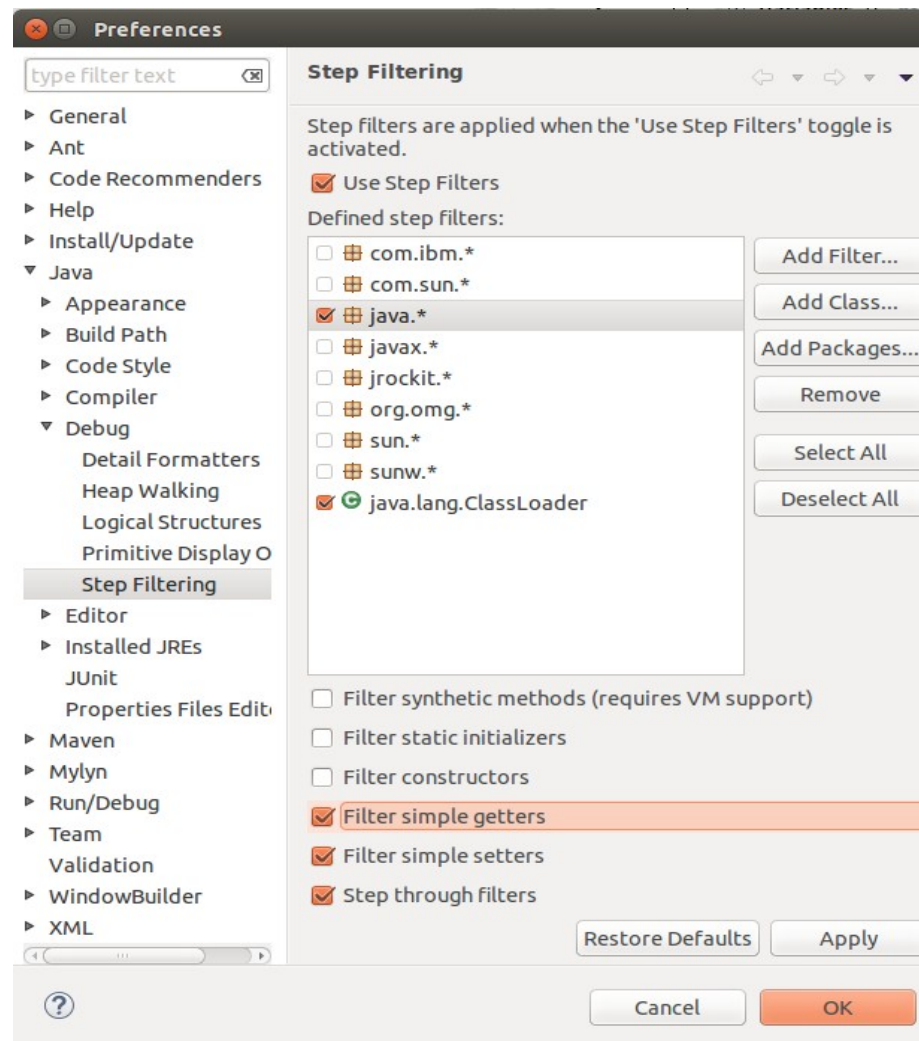
The dialog box, titled "Properties for de.vogella.debug.first.Counter - count()", has the following settings:

- Method Breakpoint** section:
 - Type: de.vogella.debug.first.Counter
 - Method: count()
 - Enabled
 - Hit Count: [empty text box]
 - Enable Condition (Ctrl+Space for code assist)
 - Condition editor: [empty text area]
 - Suspend when:
 - condition is 'true'
 - value of condition changes
 - Suspend on:
 - Method Entry
 - Method Exit
 - Suspend Policy: Suspend Thread

The "Suspend on:" section and the "Method Entry" checkbox are highlighted with a red box in the original image. The "Method Exit" checkbox is also highlighted with a red box. The "Suspend Policy" dropdown is set to "Suspend Thread".

Step Filter

- Window → Preferences → Java → Debug → Step Filtering



Referência

Java Debugging with Eclipse – Tutorial

<http://www.vogella.com/tutorials/EclipseDebugging/article.html>

Apostila Java e Orientação a Objetos. Apêndice - Debugging.

<http://www.caelum.com.br/apostila-java-orientacao-objetos/apendice-debugging/>