



Java

## Debug no Eclipse

### METODOLOGIA

- › Interprete o documento calmamente e com atenção.
- › Acompanhe a execução do exercício no seu computador.
- › Não hesite em consultar o formador para o esclarecimento de qualquer questão.
- › Não prossiga para o ponto seguinte sem ter compreendido totalmente o ponto anterior.
- › Caso seja necessário, execute várias vezes o exercício até ter compreendido totalmente o processo.

### CONTEÚDO PROGRAMÁTICO

1. [Debug no Eclipse](#)
2. [Breakpoints](#)
3. [Valores de variáveis e expressões](#)
4. [Block Comments](#)

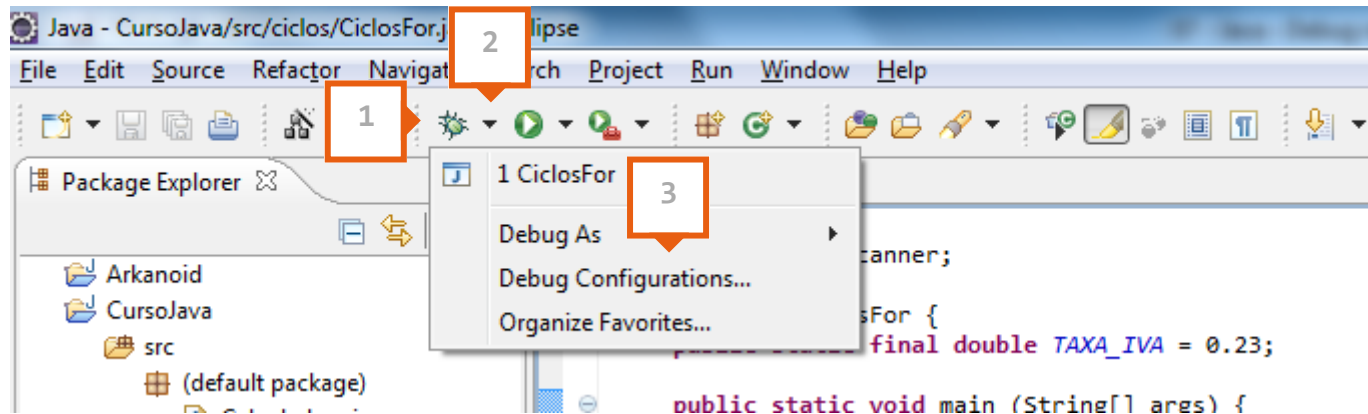
## 1. Debug no Eclipse

A expressão “debug”, que provavelmente já ouviu falar, corresponde ao processo de achar bugs ou erros. Isto é especialmente útil quando o programa em questão é grande e complexo, tornando-se difícil detetar rapidamente qual o problema no código. O **debug** permite-nos acompanhar a execução do código, assim como a alteração dos valores das variáveis e outras expressões úteis.

Vamos a um exemplo:

› Abra a classe `CiclosFor.java`

› Clique no botão de **Debug** 1

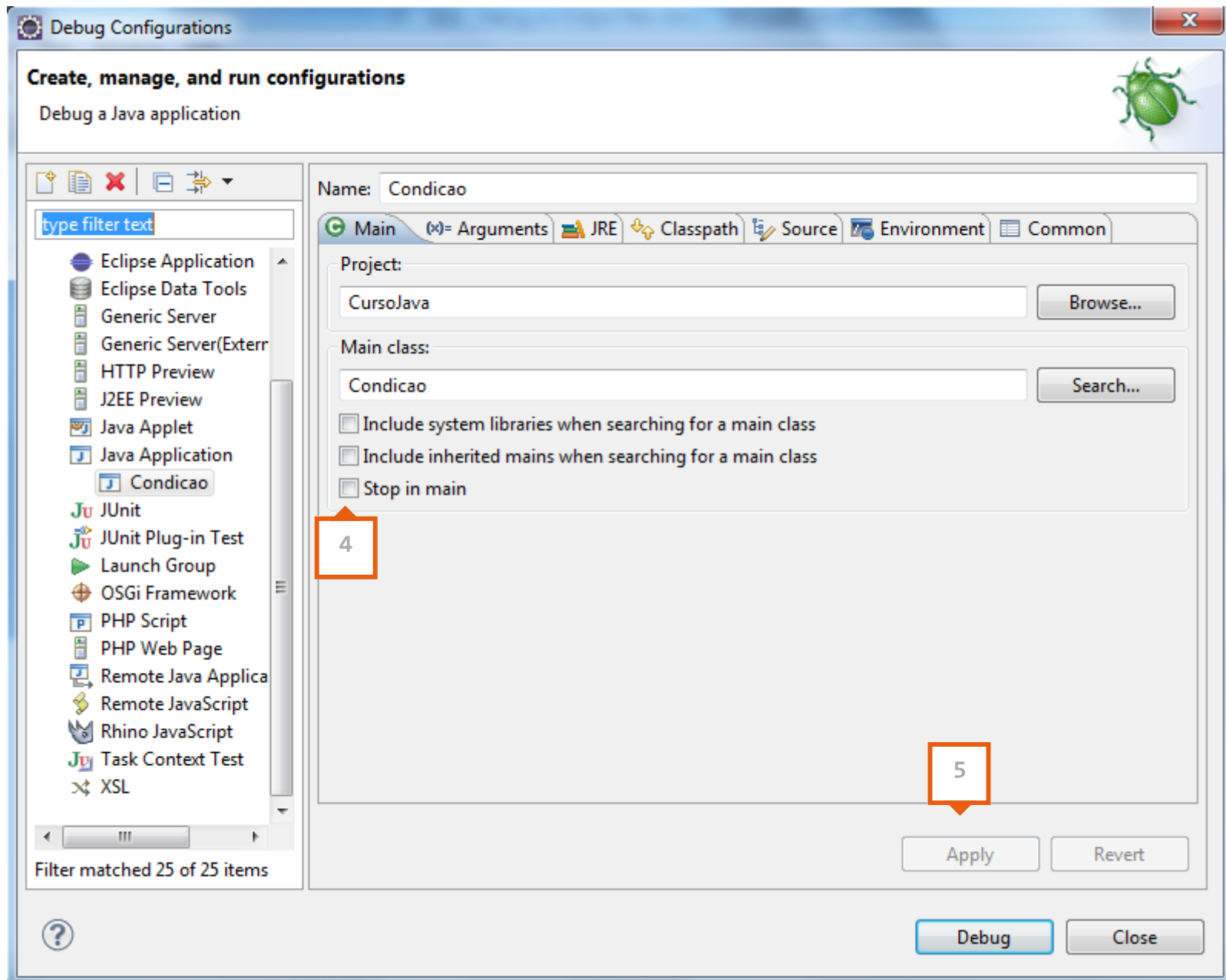


A execução do programa foi normal, da forma que costuma ver, certo? Isto acontece porque não está especificado um ponto de paragem no código, o que faz com o que seja executado normalmente.

Vamos alterar este comportamento:

› Clique na **seta** 2 ao lado do botão de **Debug**

› De seguida, clique em **Debug Configurations...** 3

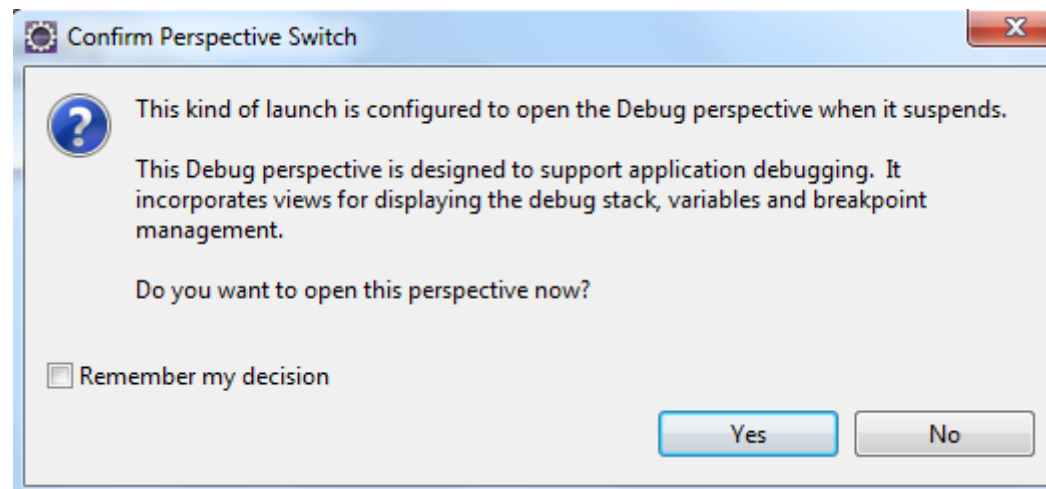


› **Clique** na opção **Stop in main** 4 4, de seguida, em **Apply** 5

📄 Acabámos de adicionar um ponto de paragem, que corresponde ao início do código que temos nesta classe.

› **Fech**e a janela e **clique** no botão de **Debug** 1

Quando o Eclipse chega ao início do código e deteta um ponto de paragem, pergunta-lhe se pretende mudar de perspetiva.



Esta mudança refere-se apenas às janelas visíveis e o seu arranjo. Isto significa que as janelas que fazem sentido ver em modo de **Debug**, podem ser diferentes das janelas que pretendemos ver em modo de desenvolvimento. A alteração da perspetiva pode ser feita, mesmo não estando em modo

**Debug**, através dos **botões de perspectiva** 6 .

› **Clique** em **Yes** na janela de mudança de perspectiva

7

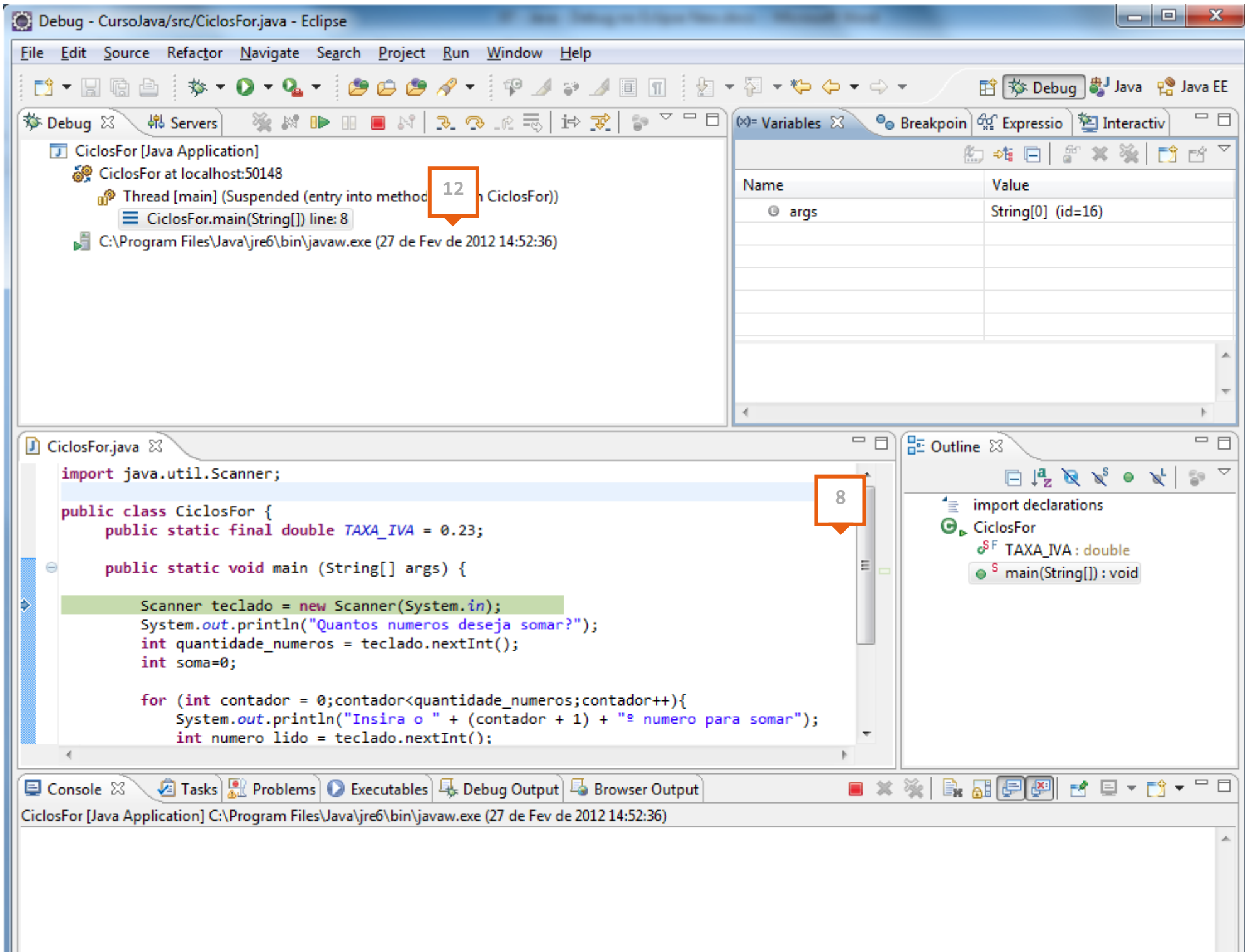
9

8

6

10

11



**Resume 7** – Corre o programa até ao próximo ponto de paragem ou até ao fim, caso não haja mais nenhum ponto de paragem.

**Terminate 8** – Para a execução do programa.

**Step Over 9** – Executa a linha em questão, avançando o cursor da linha para a linha seguinte.

**Área 10** – Área com os programas Java que estão de momento a ser executados. Isto significa que podemos estar a executar vários programas Java simultaneamente.

**Área 11** – Área das variáveis correntes. Esta zona mostra as variáveis que estão a ser utilizadas no ponto em que o programa está a executar, assim como os seus valores.

**Linha 12** – A linha verde indica a linha em que o compilador está a executar. Ou seja, este encontra-se parado à espera das suas instruções para continuar a executar.

› **Execute** o programa passo a passo, **clicando** no **Step Over 9** até chegar ao fim

📄 Repare como após executar a instrução `int quantidade numeros = teclado.nextInt()` não nos é permitido executar mais. Isto porque, o programa fica à espera de dados para o `System.in()`.

› **Escreva** um **número** na consola e `continue` a executar passo a passo até ao fim

Com instruções condicionais no código, é possível através de debug, ver diretamente quais os blocos que são executados.


- › Abra a classe `Condicao.Java`

- › Através da mesma técnica, **faça debug** a esta classe

- › Introduza **valores** diferentes quando pedido. Um acima de 19 e outro abaixo de 19

-  Repare que mediante o número inserido, é executada uma parte diferente do bloco `if`.

## 2. Breakpoints

Os **breakpoints** são os pontos de paragem do código. Sempre que o código é executado em modo debug, este tem que parar obrigatoriamente em todos os **breakpoints** definidos. Isto é bastante útil, quando o código é extenso e apenas se está interessado no funcionamento de uma parte específica do código. Nesse caso, executa-se o código em modo debug e utiliza-se o **resume**  fazendo com que o código pare apenas no próximo **breakpoint**.

- › Dê **duplo clique** na **barra azul** à esquerda do código, na linha `if (idade<19){`

-  É assim inserido um breakpoint, nessa linha de código.



📄 Para remover um breakpoint, basta fazer duplo clique sobre ele.

📄 Em alternativa, para inserir ou remover um breakpoint, pode posicionar-se na linha correta, aceder ao menu **run** e escolher a opção **Toggle breakpoint**.

› **Execute** agora o programa em **modo Debug**

📄 Este começa a executar no `main`.

› **Clique** no botão **Resume** 7

📄 O programa continuou a executar. Este vai agora parar no próximo breakpoint.

› **Introduza** a **idade** quando o programa pedir

📄 Repare como agora o código parou na linha onde definiu o breakpoint.

› **Termine** a execução do programa **clcando** em **Terminate** 8

### 3. Valores de variáveis e expressões

› **Abra** a classe `Ciclos.java`

› **Altere** a instrução:

```
int contador = 11;
```

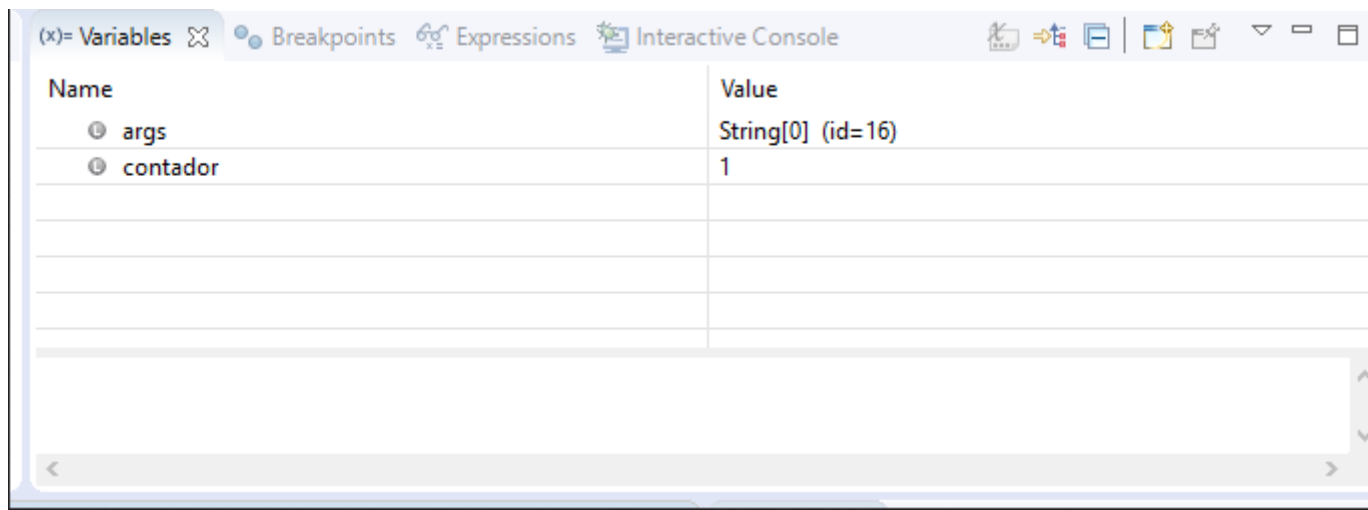
Para:

```
int contador = 1;
```

› **Insira** um breakpoint nessa mesma instrução e **execute** o programa em modo **Debug**

› **Execute** essa instrução

Repare como agora na janela das variáveis, aparece a variável contador, pois a última instrução acabou de criá-la. No seu lado direito, está o seu valor na linha em que o programa está a executar.

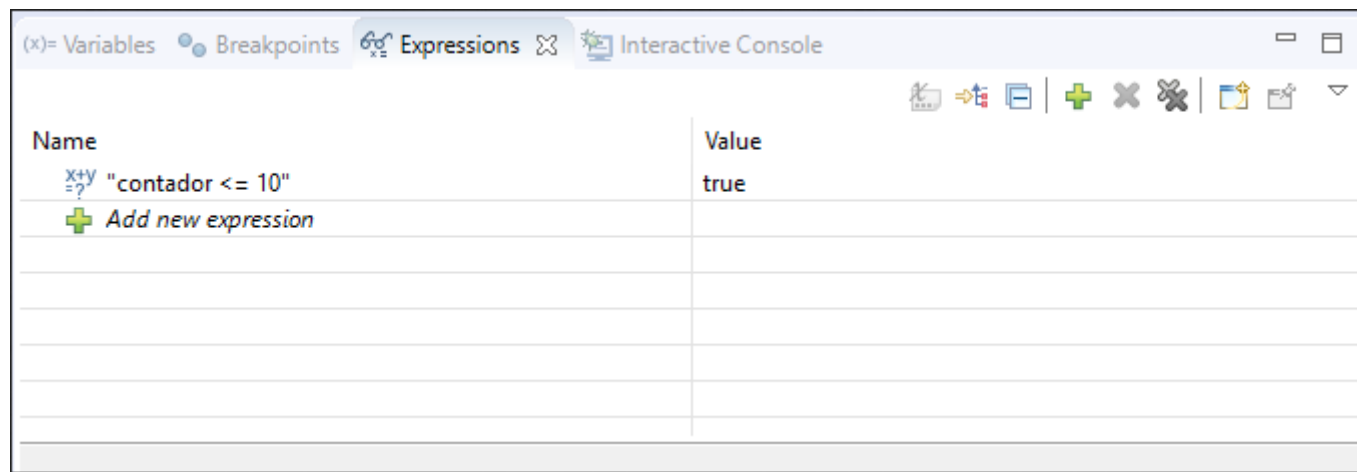


- › **Execute** mais alguns passos até observar a alteração do valor da variável

À medida que são criadas e utilizadas mais variáveis, estas serão todas representadas nesta janela. Também podemos analisar expressões, funcionalidade especialmente útil quando temos condições complexas.

- › **Selecione** o texto `contador <= 10` que está dentro do `while`
- › **Clique** com o **botão direito** do rato e, de seguida, na opção **Watch**

📄 Repare como se abriu a janela **Expressions** 13



Aqui conseguimos visualizar o valor de qualquer expressão, o que em várias situações poderá ser um indicador de que temos um erro no código.

A qualquer altura podemos apagar expressões desta janela.

- › **Clique** com o **botão direito** por cima da expressão e na opção **Remove**
- › **Continue** a executar passo a passo o programa até ao fim

## 4. Block Comments

Outra funcionalidade útil do Eclipse é a conversão de um bloco de código em comentários e vice-versa.

› **Selecione** um bloco de **5** ou **6 linhas** de código

› **Selecione** a opção **Toggle comment** no menu **Source**

📄 Observe o resultado.

› **Volte a selecionar** o mesmo bloco de **5** ou **6 linhas** de código

› **Selecione** a opção **Toggle comment** no menu **Source**

📄 Analise o resultado.

📄 **Faça** o mesmo exercício, mas agora utilizando a opção **Add block comment** do menu **Source**.