

INSTITUTO FEDERAL

Pernambuco

Campus Garanhuns

Teste de Software

Aula 02 - Depuração (debugging)

Prof. Elmano Ramalho Cavalcanti

<https://sites.google.com/site/elmano>

1. Tipos de erro
2. O que é debugar
3. Debugando no Eclipse
4. Perspectiva de debug
5. Debug avançado

1. Tipos de erro

- Sintaxe (compilação)
- Execução (runtime)
- Lógico

1. Tipos de erro

- Sintaxe
- Execução (runtime)
- Lógico

```
5 public static void main(String[] args) {  
6  
7     String nome = "Teste de Software";  
8     int N = nome.length();  
9  
10    for (int i = N; i > 0; i--) {  
11        System.out.print(nome.charAt(i));  
12    }  
13 }  
14 }
```

11

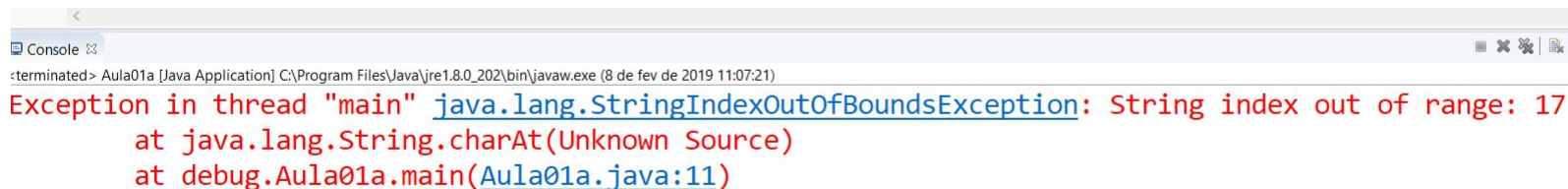
Syntax error, insert ")" to complete Expression

System

1. Tipos de erro

- Sintaxe
- Execução
- Lógico

```
3 public class Aula01a {  
4  
5     public static void main(String[] args) {  
6  
7         String nome = "Teste de Software";  
8         int N = nome.length();  
9  
10        for (int i = N; i > 0; i--) {  
11            System.out.print(nome.charAt(i));  
12        }  
13    }  
14 }
```



The screenshot shows a Java IDE window with a console output. The console title is "Console". The output text is as follows:

```
<terminated> Aula01a [Java Application] C:\Program Files\Java\jre1.8.0_202\bin\javaw.exe (8 de fev de 2019 11:07:21)  
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: String index out of range: 17  
    at java.lang.String.charAt(Unknown Source)  
    at debug.Aula01a.main(Aula01a.java:11)
```

1. Tipos de erro

- Sintaxe (compilação)
 - Execução (runtime)
 - Lógico
- ```
public static void main(String[] args) {

 String nome = "Teste de Software";
 int N = nome.length();

 for (int i = N-1; i > 0; i--) {
 System.out.print(nome.charAt(i));
 }
}
```

O programa deve imprimir o valor da variável nome de trás para frente. A lógica está correta?

Output: `erawtfoS ed etse`

## 2. O que é debugar

Erros lógicos são chamados de **bugs**. 

- ❖ Depuração (debugging) é o processo de **encontrar** e corrigir erros.

Como fazer?

- Lendo o código e buscando o erro;
- Imprimindo os valores de variáveis e do fluxo de execução;
- Utilizando a ferramenta de depuração da IDE; (best choice)

## 2. O que é depurar/debugar (*debugging*)

IDEs como Eclipse e Netbeans possuem utilitário de depuração, que permite **acompanhar o fluxo de execução do programa** (ou trecho de código). O que esses utilitários oferecem?

- Executar uma **única declaração** (statement) de cada vez;
- Entrar (**step in**) ou sair (**step over**) em/de um método;
- Inserir **breakpoints**: o programa pausa quando encontra um;
- Monitorar **variáveis**: acompanhar os valores ao longo da execução;
- Modificar variáveis: sem precisar sair do modo debug;
- Mostrar o fluxo de chamadas de métodos.



### 3. Depuração no Eclipse

Na prova de seleção para trabalhar na Google, você recebeu o desafio de corrigir o método ao lado para que ele retorne o i-ésimo número da famosa sequência de Fibonacci: 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

```
private static long fibonacciIterativo(long n) {
 long a=1;
 long b=1;
 long fib=0;
 if(n==1 || n==2) {
 return 1;
 } else {
 for(int i=2; i<=n; i++) {
 fib=a+b;
 a=b;
 b=fib;
 }
 }
 return fib;
}
```

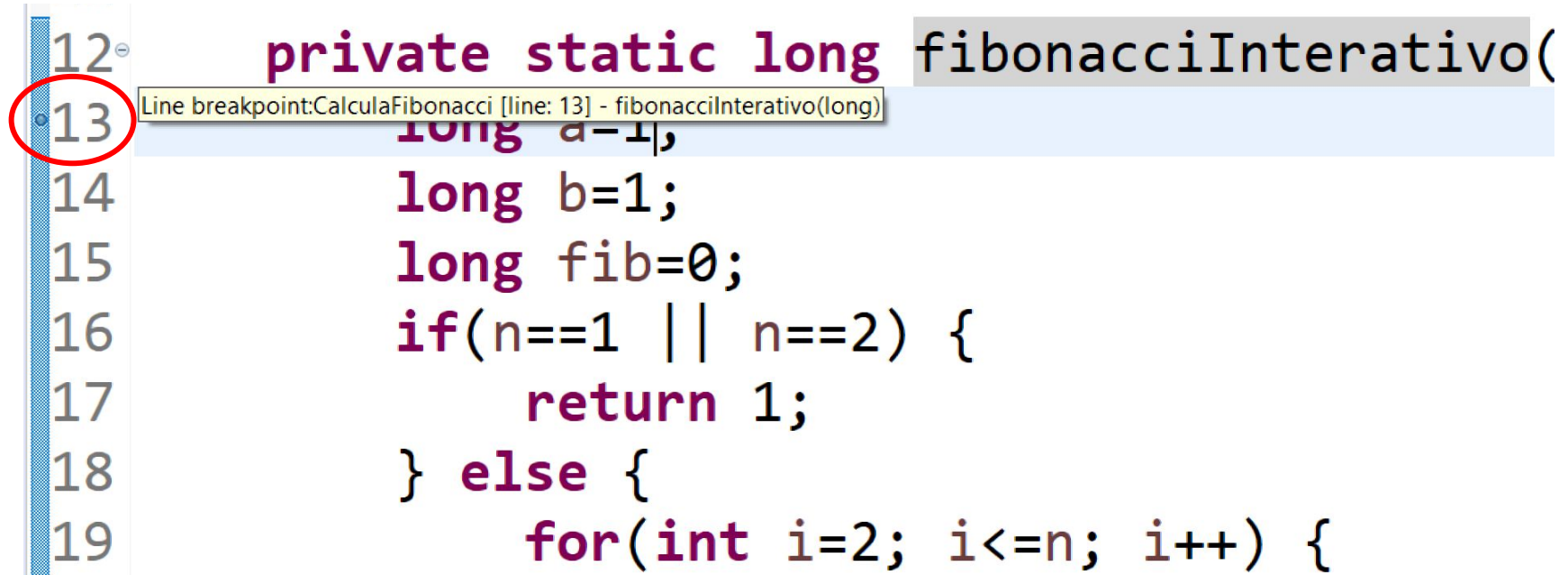
### 3. Depuração no Eclipse

1. Crie o método main, invocando o método fibonacciIterativo:

```
public static void main(String[] args) {
 Scanner sc = new Scanner(System.in);
 System.out.println("Informe a posição:");
 int n = sc.nextInt();
 System.out.println("O número Fibonacci na posição "+n+" é: "+
 fibonacciIterativo(n));
}
```

### 3. Depuração no Eclipse

2. Escolha uma linha para adicionar um breakpoint:



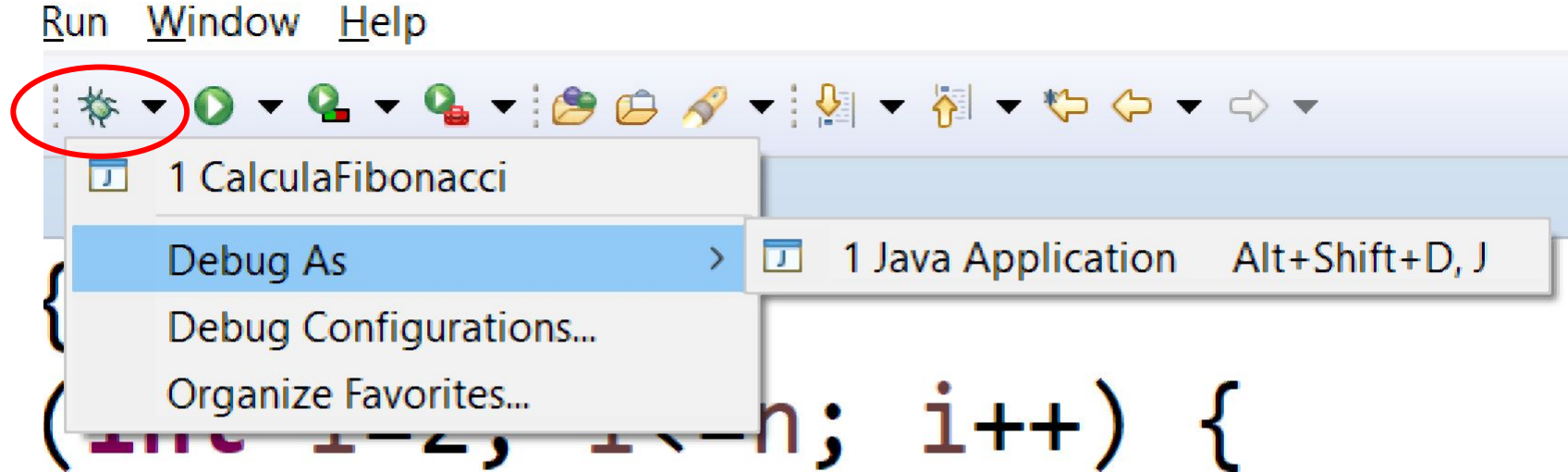
The screenshot shows the Eclipse IDE with a Java file named `CalculaFibonacci.java`. The code is as follows:

```
12 private static long fibonacciIterativo(
13 long a=1,
14 long b=1;
15 long fib=0;
16 if(n==1 || n==2) {
17 return 1;
18 } else {
19 for(int i=2; i<=n; i++) {
```

A red circle highlights line 13, and a tooltip indicates that a breakpoint has been set at this location: "Line breakpoint: CalculaFibonacci [line: 13] - fibonacciIterativo(long)".

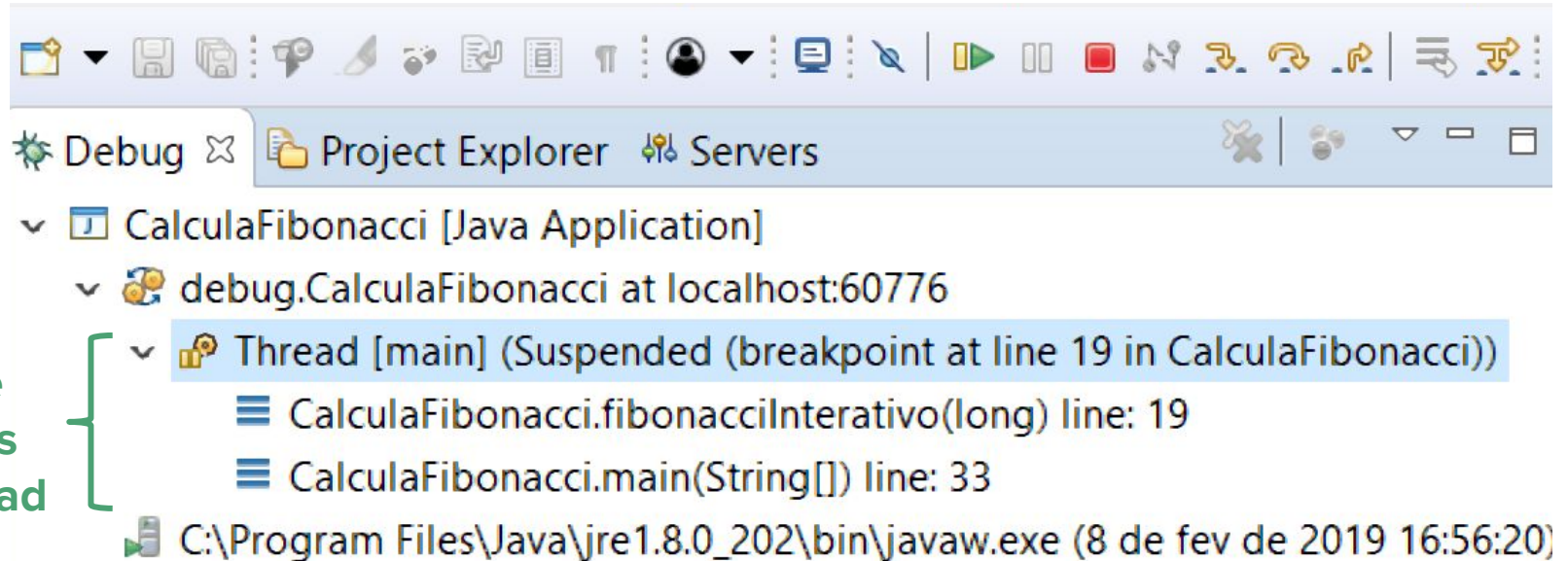
### 3. Depuração no Eclipse

3. Execute o código em modo debug:



## 4. Perspectiva de debug

Pilha de  
Métodos  
da Thread



## 4. Perspectiva de debug

**Threads** - Exibe as threads que estão sendo executadas, mostrando qual thread efetuou a chamada para o método onde está o debug. Além disso, mostra a pilha de execução, o que nos permite voltar a chamada de um método

## 4. Perspectiva de debug

Reinicia o debug a um certo nível (frame) da pilha de chamadas

**Resume (F8)**

**Step over (F6)**

**Drop to frame**



**Interrompe (ctrl+F2)**

**Step in (F5)**

**Retornar (F7)**

**Step filter**

Ex: ignorar pacotes de um certo framework (ex: JUnit) na hora do debug

## 4. Perspectiva de debug

**F5** - Vai para o próximo passo do seu programa. Se o próximo passo for um método, ele entrará no código associado;

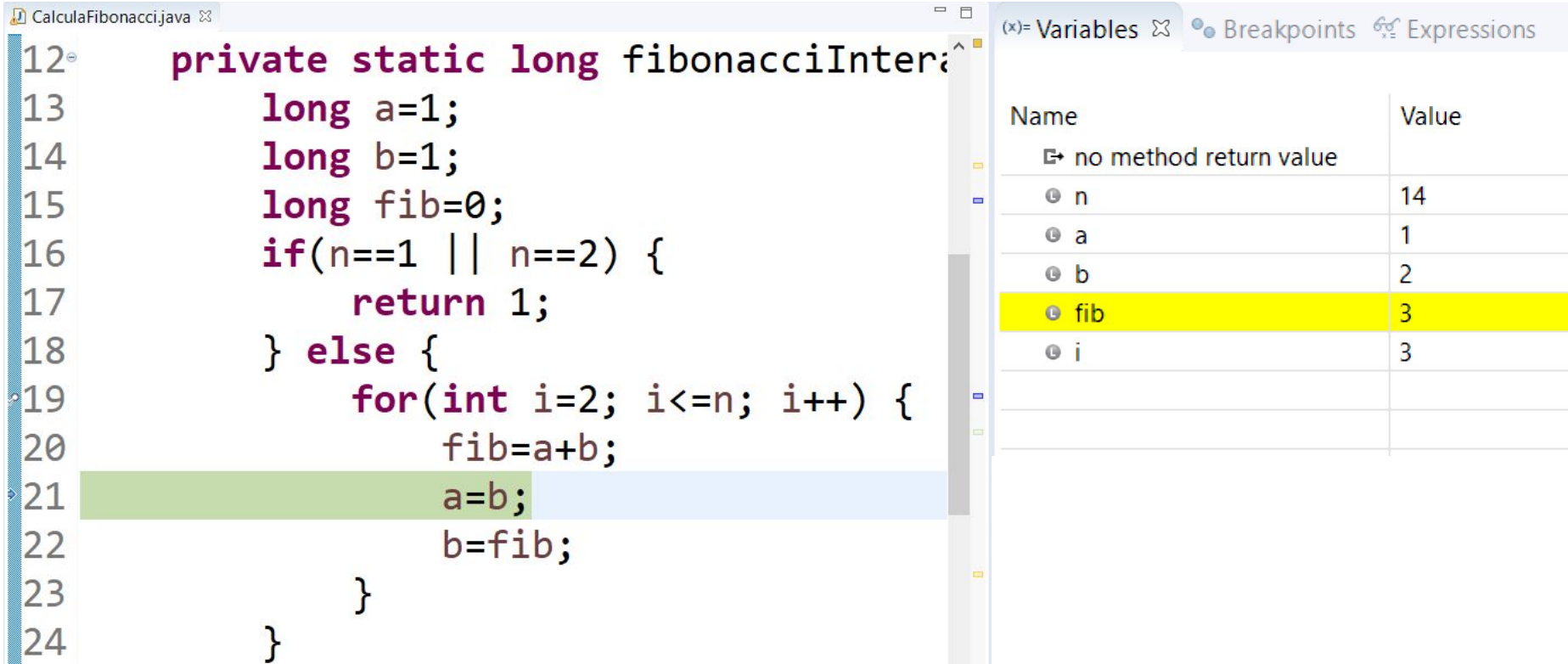
**F6** - Também vai para o próximo passo, porém se o próximo passo for um método, ele não entrará no código associado;

**F7** - Voltará e mostrará o método que fez a chamada para o código que está sendo debugado. No nosso exemplo, voltará para o método main;

**F8** - Vai para o próximo breakpoint, se nenhum for encontrado, o programa seguirá seu fluxo de execução normal.



## 4. Perspectiva de debug



The screenshot shows an IDE with a Java file named `CalculaFibonacci.java`. The code is as follows:

```
12 private static long fibonacciInterar
13 long a=1;
14 long b=1;
15 long fib=0;
16 if(n==1 || n==2) {
17 return 1;
18 } else {
19 for(int i=2; i<=n; i++) {
20 fib=a+b;
21 a=b;
22 b=fib;
23 }
24 }
```

The line `a=b;` on line 21 is highlighted in green, indicating the current execution point. To the right, the debug console shows the `Variables` tab with the following table:

| Name                   | Value |
|------------------------|-------|
| no method return value |       |
| n                      | 14    |
| a                      | 1     |
| b                      | 2     |
| fib                    | 3     |
| i                      | 3     |

## 4. Perspectiva de debug

Visual Studio Breakpoints pane configuration:

- Variables | Breakpoints | Expressions
- CalculaFibonacci [line: 8] - fibonacciRecursivo(int)
- ☒ CalculaFibonacci [line: 19] [hit count: 10] - fibonacciIterativo()

Breakpoint settings for the selected breakpoint:

- ☐ Trigger Point
- ☒ Hit count: 10
- ☒ Conditional
- ☒ Suspend thread
- ☐ Suspend VM
- <Choose a previously entered condition>

Visual Studio Expressions pane:

| Name                 | Value |
|----------------------|-------|
| $x+y$ "a+b"          | 21    |
| $x+y$ "fib"          | 13    |
| $x+y$ ""+100*i/n+"%" | 50%   |
| + Add new expression |       |

## 5. Debug avançado

1. Watchpoint
2. Hit count
3. Exception breakpoint
4. Method breakpoint
5. Breakpoints for loading classes

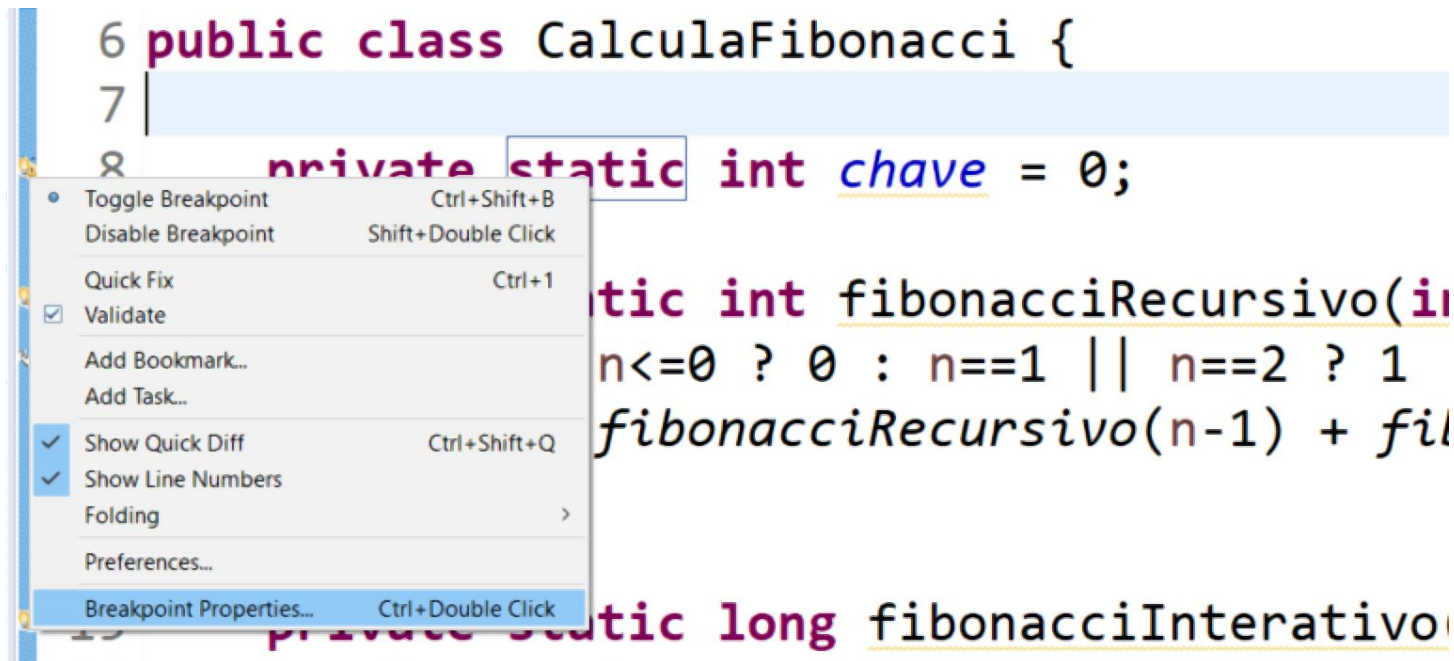
## 5. Debug avançado

### 1) Watchpoint

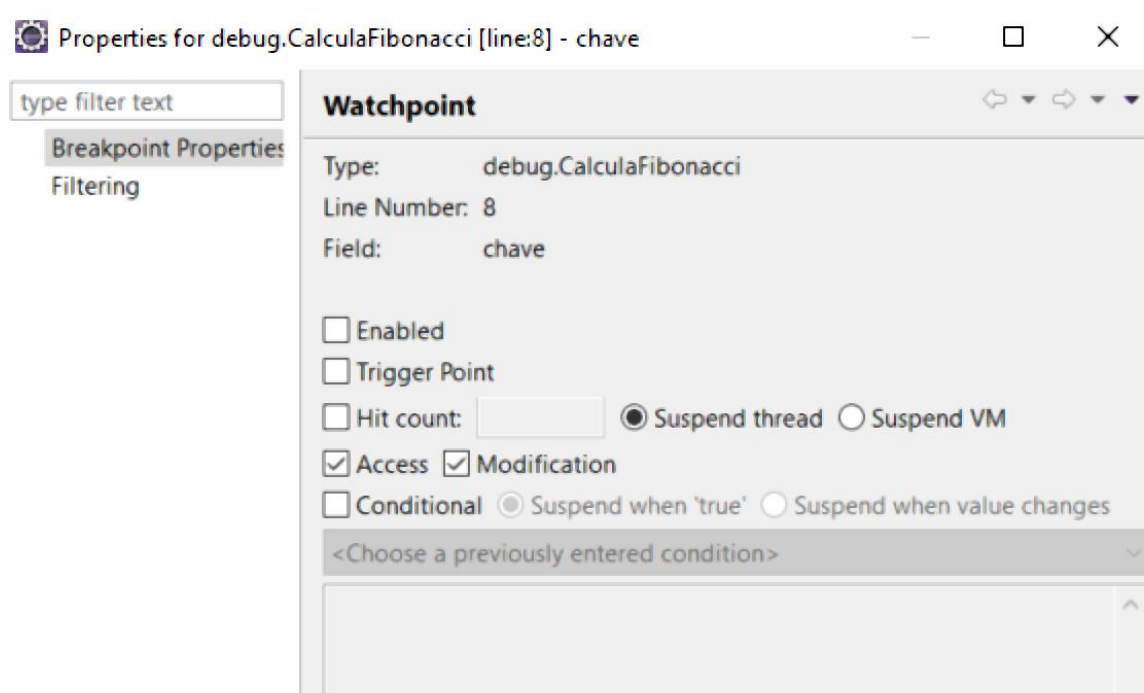
Um watchpoint é um ponto de interrupção especial que interrompe a execução de um programa sempre que **o valor de uma expressão é alterado**, independente de quando ou onde isso ocorre.

# Adicionando um watchpoint

Clique duplo no ícone da variável (à esquerda do número da linha)



# Configurando um watchpoint



## 5. Debug avançado

### 2) Hit count

Para cada breakpoint, você pode especificar uma contagem de ocorrências em suas propriedades. O aplicativo é interrompido quando **o breakpoint for atingido o número de vezes definido** na contagem de ocorrências.

## 5. Debug avançado

### 3) Exception breakpoints

Você pode definir breakpoint para exceções lançadas. O programa entrará em modo debug automaticamente se uma determinada exceção (ex: NullPointerException) for lançada.

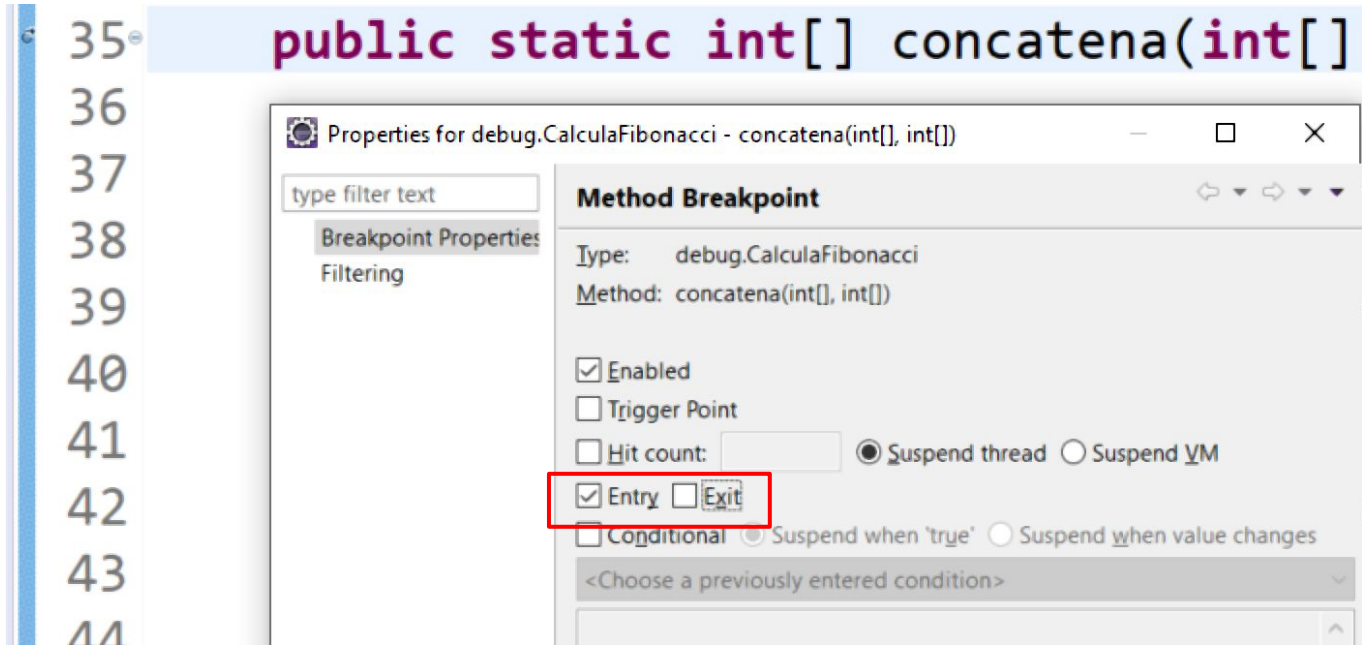




## 5. Debug avançado

### 4) Method breakpoint

Pode-se definir um breakpoint na entrada ou saída do método, por exemplo.

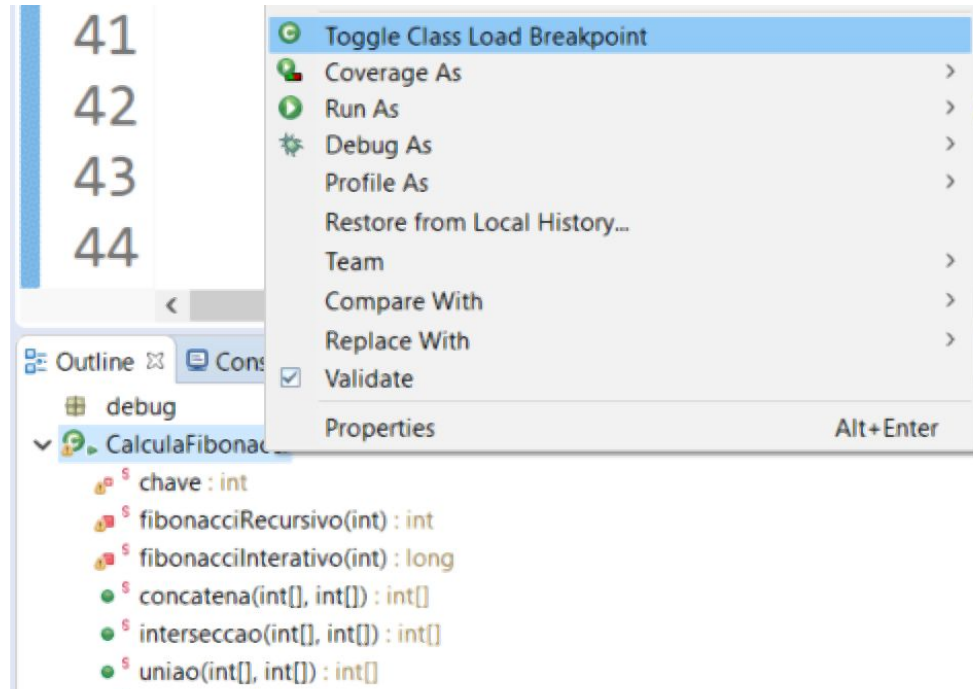


## 5. Debug avançado

### 5) Breakpoints for loading classes

Pode-se definir um breakpoint para quando uma classe é carregada.

View Outline > botão direito em cima da classe



# Verificação da Aprendizagem Teórica

## Questões de Concurso

**1) Ano: 2011 Banca: CESPE Órgão: BRB Prova: CESPE - 2011 - BRB - Analista de Tecnologia da Informação**

No Eclipse, a perspectiva Debug possui várias views para realizar a depuração de um programa Java: uma delas é a view Debug, que exibe os servidores configurados para executar o projeto e a lista de processos Java em execução.

( ) Certo

( ) Errado

**2) Ano: 2012 Banca: CESPE Órgão: Banco da Amazônia Prova:**  
**CESPE - 2012 - Banco da Amazônia - Técnico Científico - Análise de**  
**Sistemas**

Um depurador é definido como um ambiente especializado para controlar e monitorar a execução de um programa. A sua funcionalidade básica consiste na inserção de pontos de parada no código, de forma que, quando o programa esteja parado, o valor corrente das variáveis possa ser verificado.

( ) Certo

( ) Errado

**3) Ano: 2012 Banca: PaqTcPB Órgão: UEPB Prova: PaqTcPB - 2012 - UEPB - Técnico em Informática - Programador**

O erro que ocorre quando tentamos armazenar mais bits do que uma capacidade estabelecida para uma variável é conhecido como:

- A) Exceção
- B) Overflow
- C) Warning
- D) Bug
- E) Interrupção

# The 5 Stages of Debugging

At some point in each of our lives, we must face errors in our code. Debugging is a natural healing process to help us through these times. It is important to recognize these common stages and realize that debugging will eventually come to an end.



## Denial

This stage is often characterized by such phrases as "What? That's impossible," or "I know this is right." A strong sign of denial is recompiling without changing any code, "just in case."



## Bargaining/Self-Blame

Several programming errors are uncovered and the programmer feels stupid and guilty for having made them. Bargaining is common: "If I fix this, will you please compile?" Also, "I only have 14 errors to go!"



## Anger

Cryptic error messages send the programmer into a rage. This stage is accompanied by an hours-long and profanity-filled diatribe about the limitations of the language directed at whomever will listen.



## Depression

Following the outburst, the programmer becomes aware that hours have gone by unproductively and there is still no solution in sight. The programmer becomes listless. Posture often deteriorates.



## Acceptance

The programmer finally accepts the situation, declares the bug a "feature", and goes to play some Quake.

# Referências

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

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

[https://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.cdt.doc.user%2Ftasks%2Fcdt\\_t\\_add\\_watch.htm](https://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.cdt.doc.user%2Ftasks%2Fcdt_t_add_watch.htm)

Y. Daniel Liang. Introduction to Java Programming. Pearson, 10th edition, 2015.