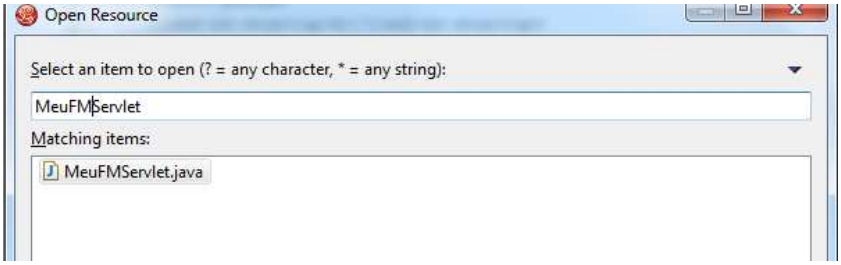
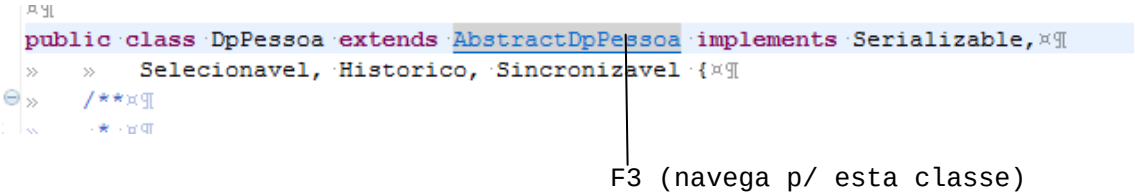
Anexo 3 - Utilizando o JBOSS Developer Studio

3.1 – Pesquisando uma classe (Ctrl+Shift+R)



Clicando em MeuFMServlet.java abre o mesmo no editor. Porém onde ele está na árvore do Package Explorer? Ver no intem 3.9 (Link Editor).

3.2 – OpenOn - Navegando nas classes - (Cursor na classe e F3 ou Ctrl+click)



OpenOn provides an easy method for switching directly from one project resource to another without navigating through the Package Explorer view. Pressing F3 or Ctrl+click when a reference to another file is highlighted will open the file in the editor. Refer to the Editors chapter of the Visual Web Tools Reference Guide for more details.

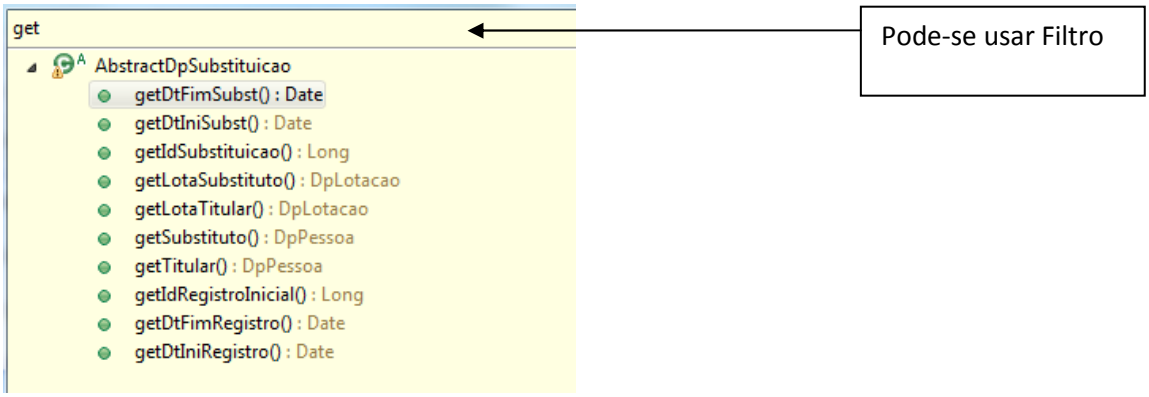
3.3 – Ajuda Comando - (Ctrl+Space)

Content Assist displays context-specific code completion suggestions while typing, speeding up development and reducing typing errors. Content Assist is supported in the following contexts: The suggestion list can be displayed by pressing Ctrl+Space, and the highlighted entry can be selected and inserted by pressing Enter.

Note

3.4 - Encontrar os atributos / métodos dentro da classe - (Crtl+O)

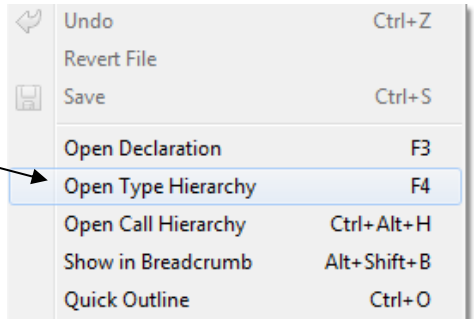
Com uma classe no editor, o Crtl+O promove a lista dos métodos e atributos que podem ser filtrados.



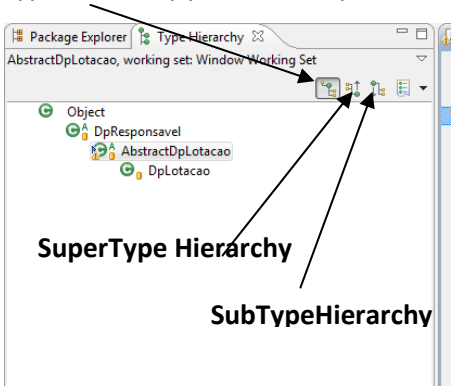
3.5 - Hierarquia das classes (Open type Hierarchy – F4)

bstractDpLotacao.java

```
63  »   @Desconsiderar¤¶
64  »   private Set<DpLotacao> dpLotacaoSubordinadosSet;¤¶
65  ¤¶
66  »   private CpTipoLotacao cpTipoLotacao;¤¶
67  »   /**¤¶
68  »    * @return the cpTipoLotacao¤¶
69  »    */¤¶
70  »   public CpTipoLotacao getCpTipoLotacao() {¤¶
71  »   »   return cpTipoLotacao;¤¶
72  »   }¤¶
73  ¤¶
74  »   /**¤¶
```



Type Hierarchy p/ AbstractDpLotacao (Dpresponsavel -> AbstractDpLoatacao -> DpLotacao)

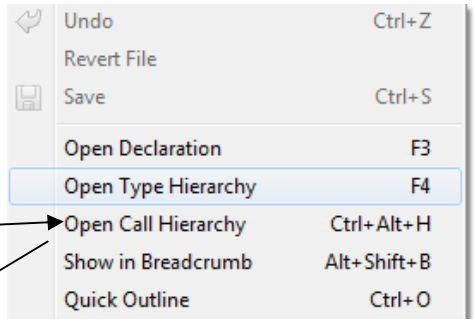


3.6 – Onde é referenciado (Open Call Hierarchy – Crtl+Alt+H)

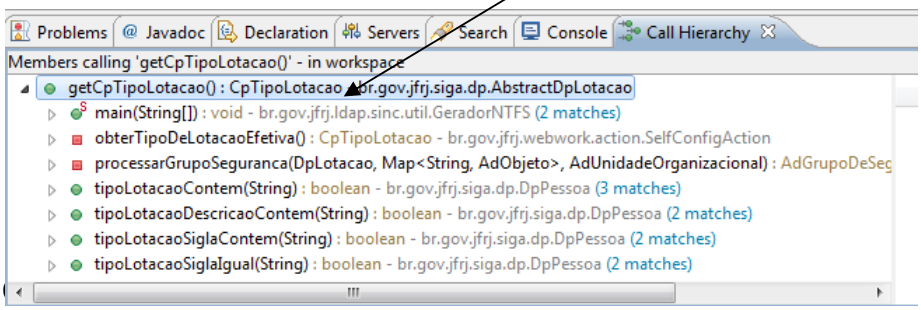
Ex: onde o método getCpTipoLotacao() é utilizado

AbstractDpLotacao.java

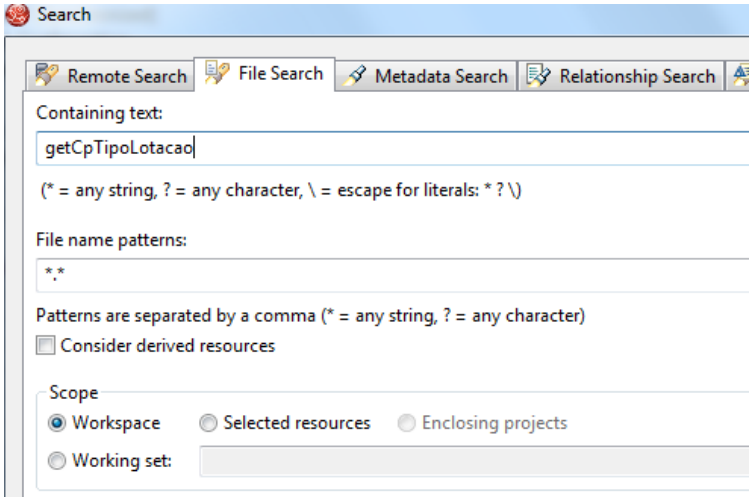
```
63  »   @Desconsiderar¤¶
64  »   private Set<DpLotacao> dpLotacaoSubordinadosSet;¤¶
65  ¤¶
66  »   private CpTipoLotacao cpTipoLotacao;¤¶
67  »   /**¤¶
68  »    * @return the cpTipoLotacao¤¶
69  »    */¤¶
70  »   public CpTipoLotacao getCpTipoLotacao() {¤¶
71  »   »   return cpTipoLotacao;¤¶
72  »   }¤¶
73  ¤¶
74  »   /**¤¶
```



Call Hierarchy p/ o getCpTipoLotacao()

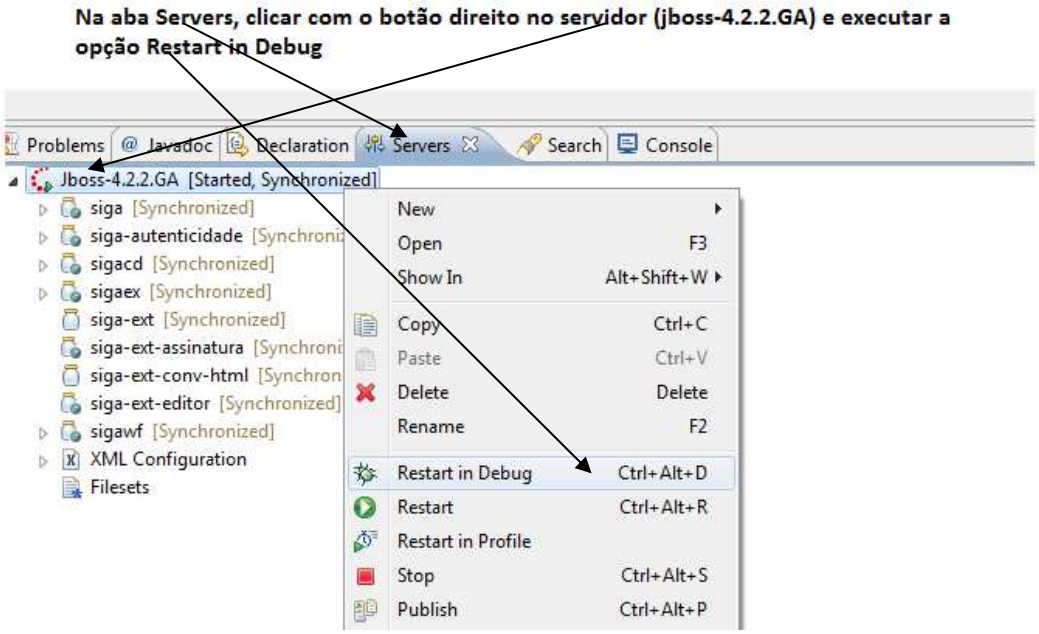


Esta forma de descobrir quem chama getCpTipoLotacao() não é 100% confiável, visto que as chamadas em código JSP ou FM não aparecerão no Call Hierarchy. Neste caso devemos utilizar o Search / File...



3.7 – Debugando um método JAVA

Colocando o servidor em debug Mode:



Uma vez que o servidor esteja em debug mode, selecione a classe/ método desejado.

```
[#assign suprid = func.calculaData("22","08/09/2012")/]
${suprid}
```

A aplicação FM acima executa o método calculaData() na classe FuncoesEl.java.

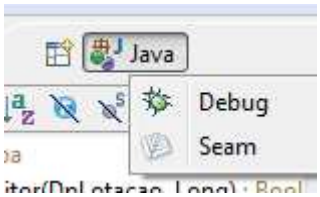
Encontrando a classe e método:
Utilizando Ctrl+Shift+R (item 1) fornecemos o nome da classe.
Utilizando Crtl+O (item 4) fornecemos o nome do método.

Selecione a linha e insira um breakpoint com duplo click.

Dê um duplo clique na linha desejada. No caso, 479. Observe o círculo cheio

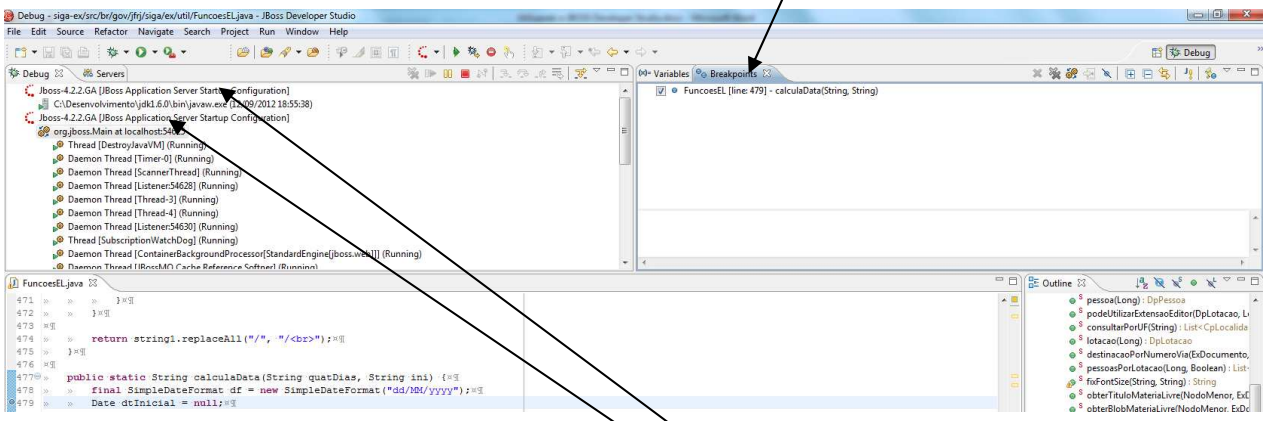
```
477  public static String calculaData(String quatDias, String ini) {
478      final SimpleDateFormat df = new SimpleDateFormat("dd/MM/yyyy");
479      Date dtInicial = null;
480      Integer dias;
481      try {
482          dias = Integer.parseInt(quatDias);
483          dtInicial = df.parse(ini);
484          Calendar c = Calendar.getInstance();
485          c.setTime(dtInicial);
486          c.add(Calendar.DAY_OF_MONTH, dias - 1);
487          dtInicial = c.getTime();
488      } catch (final ParseException e) {
489          return null;
490      } catch (final NullPointerException e) {
491          return null;
492      }
493      return df.format(dtInicial);
494  }
```

Mude a perspectiva de JAVA para Debug:



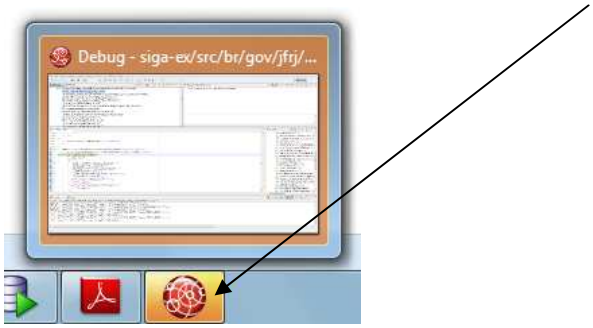
Verifique se não existem outros breakpoints indesejados na aba Breakpoints.

Verifique se não existem outras instâncias do jboss rodando.

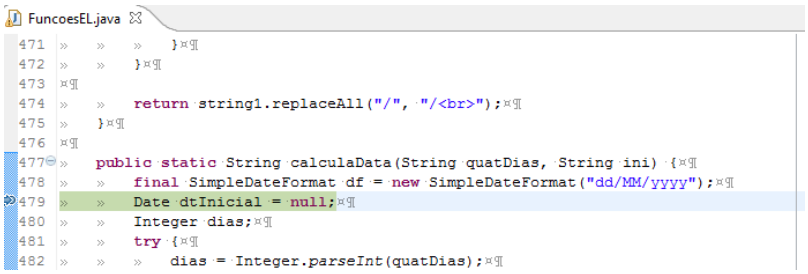


OBSERVAÇÃO: No caso acima existem 2 instâncias, e quando rodei a aplicação FM o debug não entrou. Coloquei o cursor nas instâncias, botão direito e executei a opção Terminate and Remove. Depois startei o servidor novamente em modo debug.

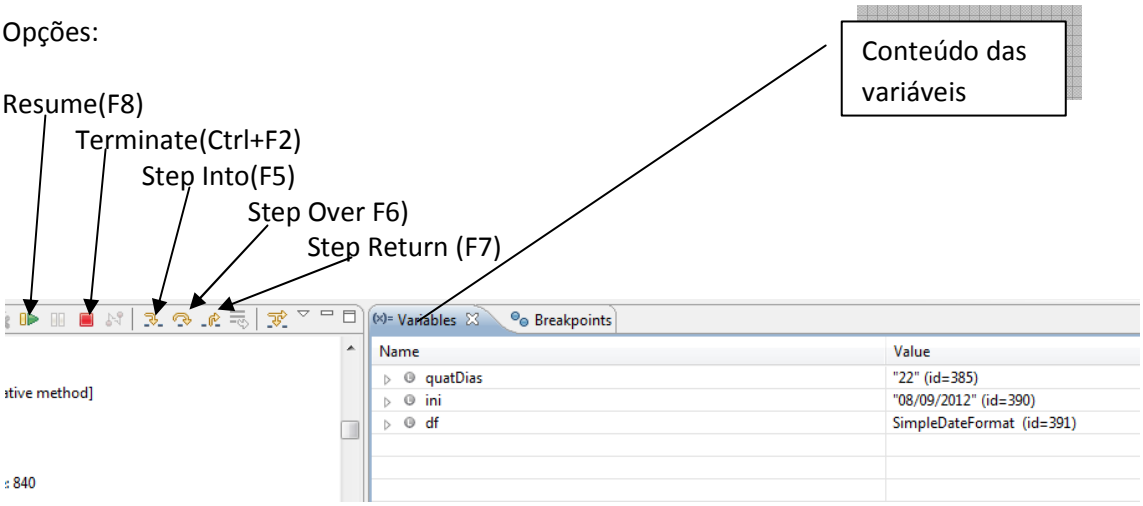
Executei novamente a aplicação FM, e o JBDS acusou (ícone piscando) o debug / breakpoint.

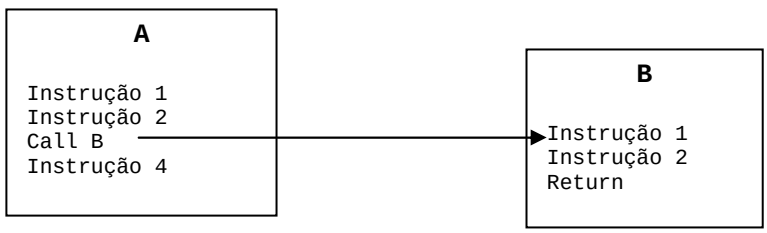


Código parado do breakpoint:



Opções:





Terminate – Interrompe o servidor em modo Debug

Resume - Executa o código até o próximo breakpoint, caso exista, ou até o final do código.

Step over – Passo sobre: vai executando linha-a-linha, porém se o seu código chama outro método / função (incluindo os métodos / funções da linguagem, tais

como: parse.int, string ...) ele não entra. Se na figura acima o debug (em A) está na instrução Call B, ele executará B, porém não entrará no código de B.

> Step **over** proceeds to the next line in your current scope (i.e. it goes to the next line), without descending into any method calls on the way. This is generally used for following the logic through a particular method without worrying about the details of its collaborators, and can be useful for finding at what point in a method the expected conditions are violated.

Step into - Passo Dentro. Idêntico ao step over, porém ele entrará nos métodos / funções chamados. Se na figura acima o debug (em A) está na instrução Call B, ele entrará em B, e continuará executando linha-a-linha. A única forma de sair de B é chamar o Step out / Step return.

> Step **into** will cause the debugger to descend into any method calls on the current line. If there are multiple method calls, they'll be visited in order of execution; if there are no method calls, this is same as step over. This is broadly equivalent to following every individual line of execution as would be seen by the interpreter.

Step Out / Step Return – termina a execução do método / função e retorna ao chamador na instrução de chamada. Se na figura acima o debug (em A) está na instrução Call B, e o usuário está modo Step into, ele entrará em B, e continuará executando linha-a-linha. A única forma de sair de B diretamente (sem executá-lo) é chamar o Step out / Step return, que fará o debug retornar para A na instrução Call B. Se usuário fornecer Step over, ele executará sem entrar em B, se der Step into, entrará novamente em B, e executará linha-a-linha.

> Step out proceeds until the next "return" or equivalent - i.e. until control has returned to the preceding stack frame. This is generally used when you've seen all you need to at thispoint/method, and want to bubble up the stack a few layers to where the value is actually used.

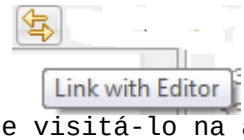
Step Into	**Step Over**	**Step Out / Step Return**
➢ Step Into will cause the debugger to go into the next function call and break there. ➢ Go into the subroutine and wait for next action. ➢ Change the debugger context to run into the function the code is stopped on. If the code cannot step into the function, this is the same as Step Over.	➢ Step Over will tell the debugger to execute the next function and break afterwards. ➢ Jump over the subroutine without waiting again. ➢ Execute the code the debugger is stopped on, but stay within the current function.	➢ Step Out will tell the debugger to finish the current function and break after it. ➢ If you are in the subroutine, you will leave it without waiting again ➢ Execute code until the end of the current function, and resume debugging once it has returned.

3.8 – Limpando Projetos Publicados

Clean projects published on the server.
You can use the clean option available in the Servers view to remove any invalid resources from the server before doing a full republish. This helps remove by-products generated as a result of the publishing process. When you find there is old code or an invalid state of code running on the server, try using this clean option to see if this helps remove these invalid states from the server.
To clean projects published on the server, complete the following steps:

1. In the Servers view, right-click a server.
2. In the pop-up menu, select Clean.
3. The following message dialog box displays:
4. Clean will discard all publish state and republish from scratch. Are you sure youwant to clean all published projects?
5. Select **OK**.

3.9 – Link do Editor com o Package Explorer (Tree)



Com um aplicativo abeto no Editor, pode-se rapidamente visitá-lo na árvore do projeto (Package Explorer) utilizando a opção Link with Editor.