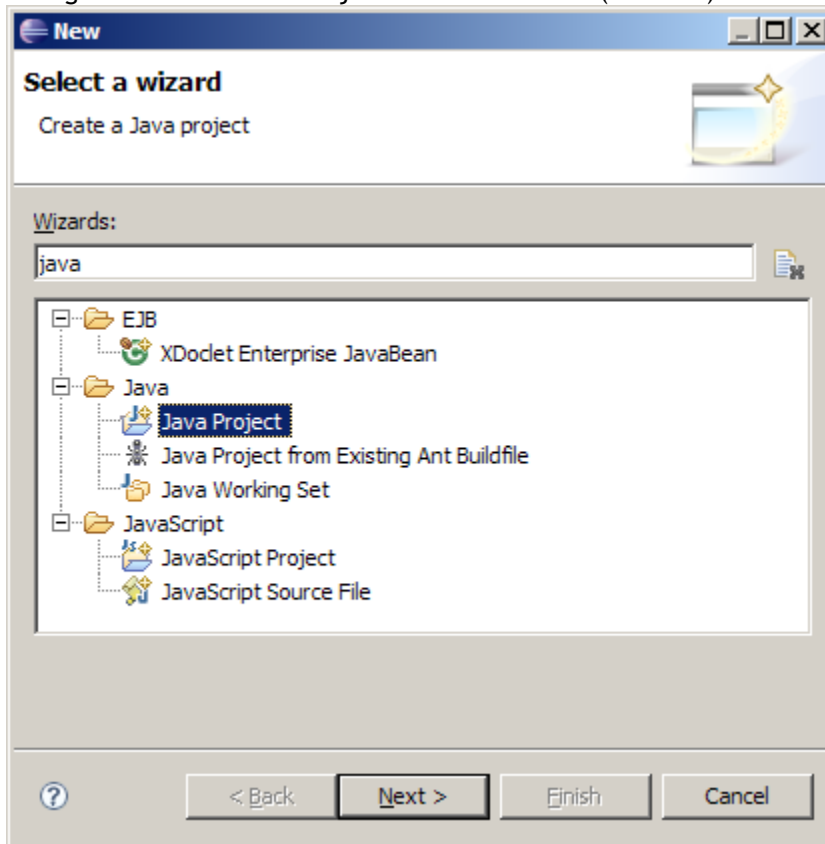

Conteúdos

1. Eclipse - Visita guiada
 2. Atalhos
 3. Referências
-

1. Eclipse - Visita guiada

Criar um novo projecto Java

Navegar até New > Java Project ou New > Other (CTRL+N) e escolher "Java Project".



Podemos usar um **filtro de procura** para filtrar os tipos que são sugeridos na lista. Introduzindo, por exemplo, "java", surgem vários recursos relacionados com Java. O filtro de procura surge em vários diálogos do Eclipse e é uma ajuda preciosa para a navegação, uma vez que há diálogos com dezenas de opções estruturadas hierarquicamente (ex. **Window > Preferences**)

O diálogo new > other permite criar inúmeros artefactos (Folder, Project, Text File, CVS Repository Location, Project from CVS, Hibernate files, Java Class, Java Interface, Java Package, JUnit Test Case,

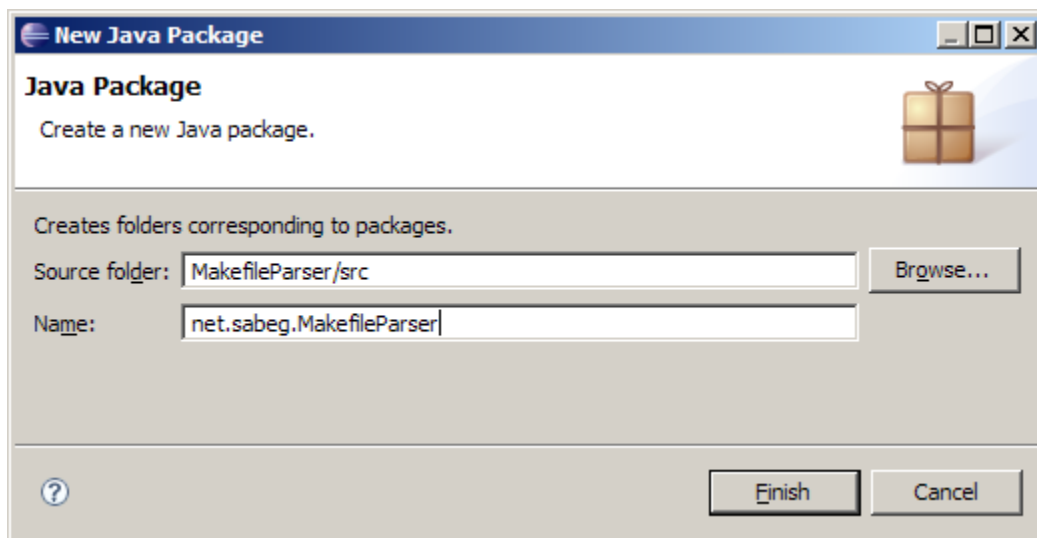
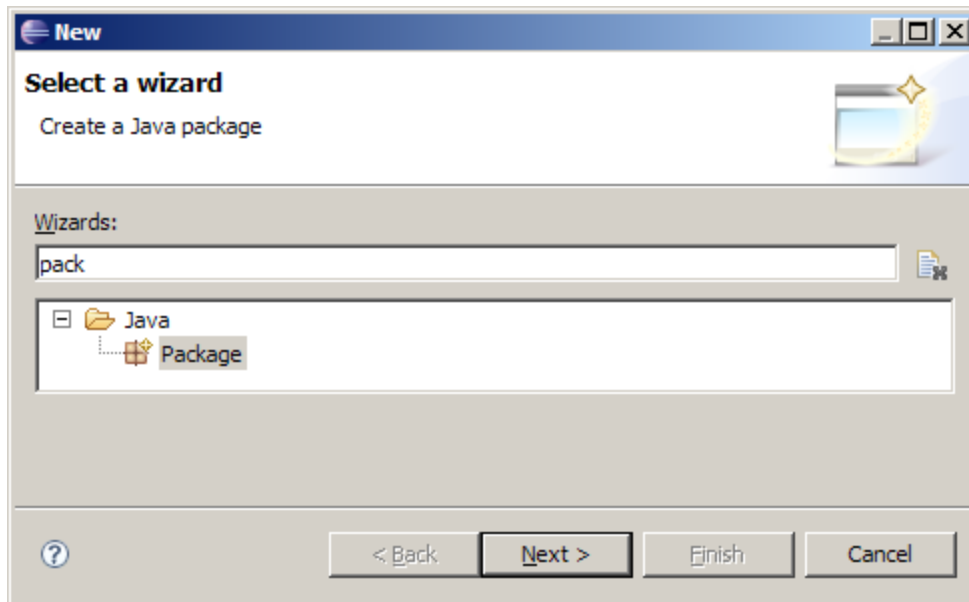
Task, CSS File, HTML, JSP, WSDL, XML, XSD, entre muitos outros)

Para tipos de ficheiros comuns o Eclipse tem editores próprios que auxiliam a codificação (Text editor, Ant editor, SQL editor, Java editor, Javascript editor, XML editor, ...) e que contêm content-specific features.

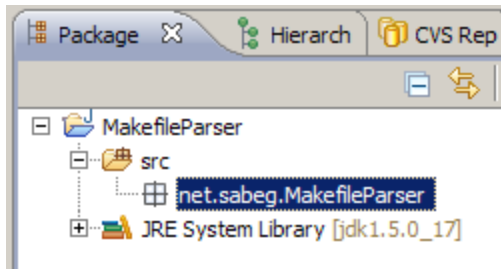
Criar uma nova package

Packages, classes, interfaces, enumerados, etc, podem ser criados usando os wizards que o Eclipse tem para facilitar o desenvolvimento.

Usando o atalho **CTRL+N** e o filtro de procura "pack" rapidamente estamos a criar o novo package:

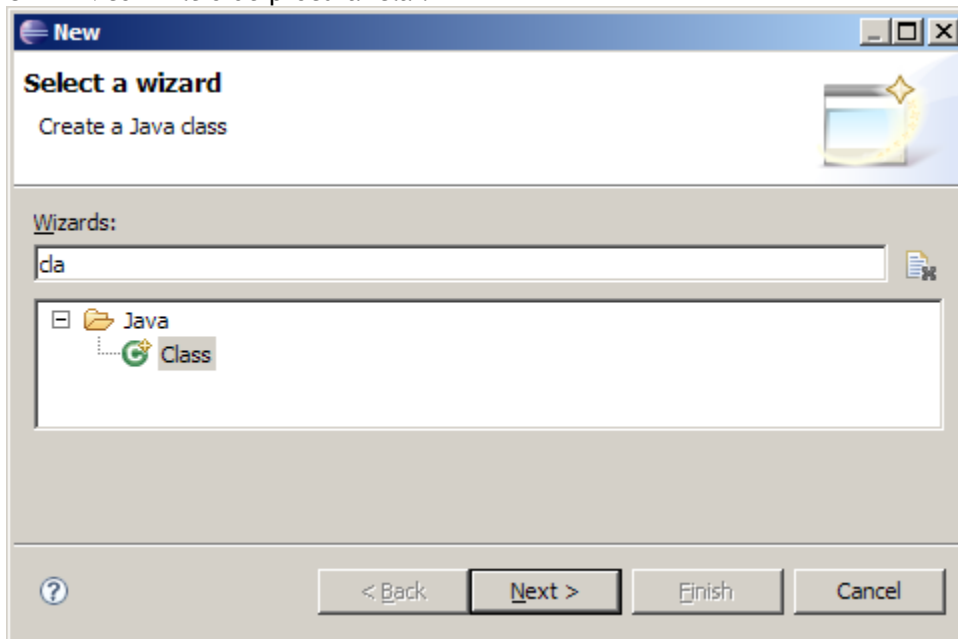


O novo package aparece no explorador de packages:



Criar uma nova classe

Podemos criar uma nova classe dentro de um package seleccionado o package pretendido e usando o CTRL+N com filtro de procura "cla":



O wizard de criação de classes é muito completo, permitido definir, para a nova classe, modificadores, superclasse, interfaces implementadas, method stubs mais comuns...

New Java Class

Java Class
Create a new Java class.

Source folder:

Package:

☐ Endorsing type:

Name:

Modifiers: ☒ public ☐ default ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

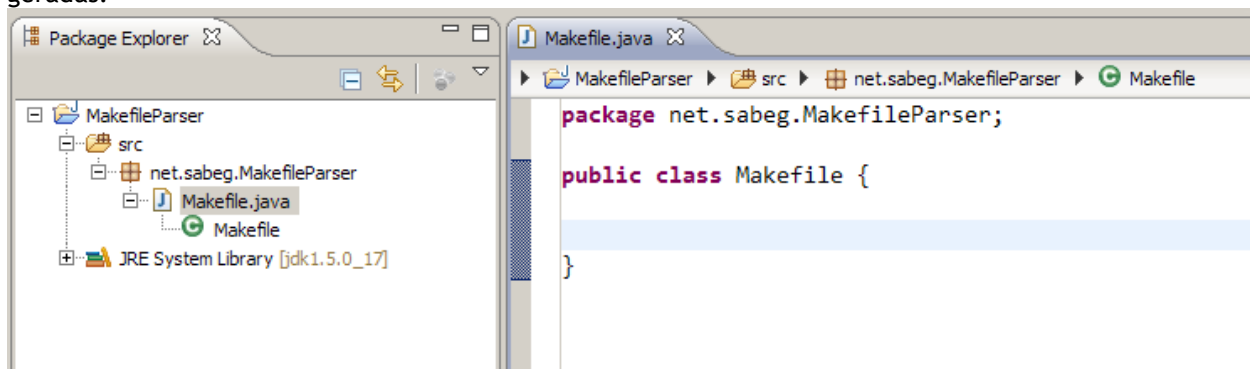
☐ Constructors from superclass

☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

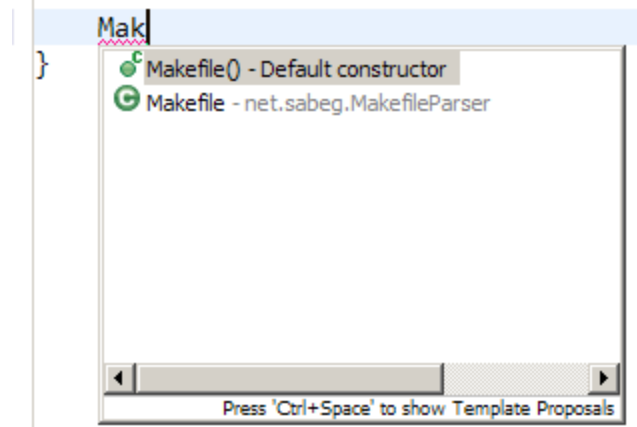
O código da classe, bem como o package e eventuais imports necessários, são gerados automaticamente. Se tiverem sido adicionados interfaces as method stubs necessárias são também geradas.



Rapidamente se implementa o construtor default. Introduzindo os primeiros caracteres do nome da classe seguidos de **CTRL+SPACE**, surge uma lista de sugestões

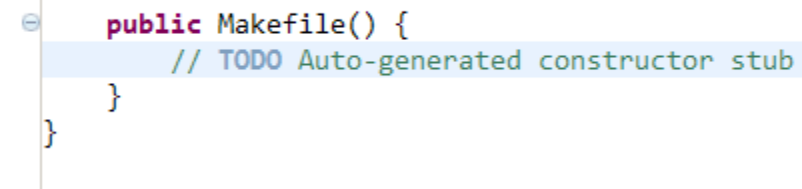
Esta funcionalidade denomina-se **Content Assist** e dá apoio à codificação, sensível ao contexto. Todos os elementos Java aplicáveis num dado contexto e reconhecidos pelo Eclipse são listados no Content Assist. Noutros tipos de conteúdo (XML, Ant buildfiles, JSP, ...) o Content Assist irá reconhecer os elementos dessas linguagens.

```
public class Makefile {
```

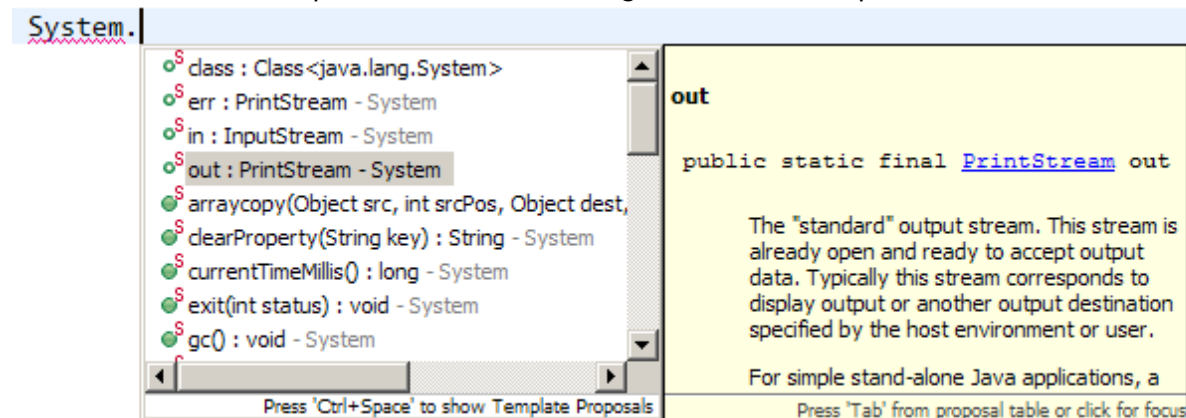


O ícone à esquerda da assinatura do método permite expandir ou colapsar o corpo do método.

```
public class Makefile {
```



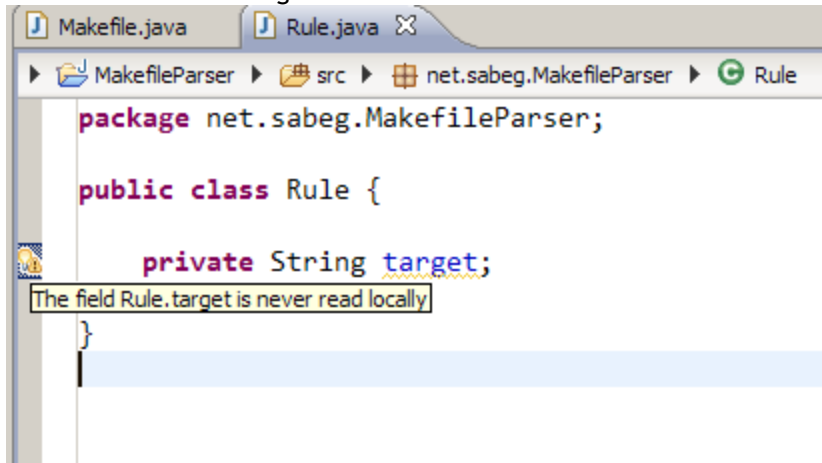
O Content Assist é automaticamente invocado quando carregamos na tecla "." no contexto de um elemento Java. Note-se que o Javadoc dos itens sugeridos é também apresentado.



Gerar getters e setters automaticamente

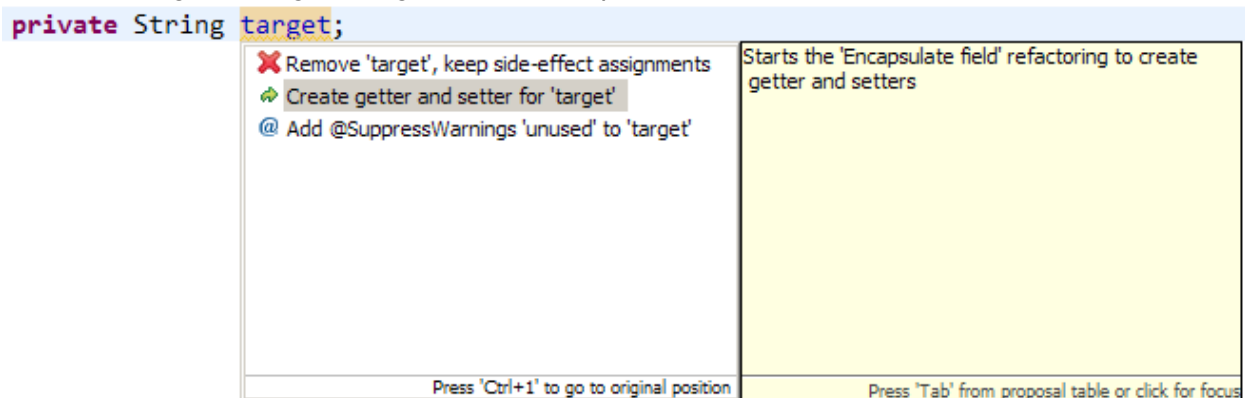
O Eclipse pode gerar automaticamente getters, setters, construtores, stubs para métodos a implementar, entre outras zonas de código comuns. No menu Source (**ALT+SHIFT+S**) encontramos estas e muitas outras funcionalidades.

Na figura seguinte vemos que o Eclipse assinala um problema (um aviso). O atributo "target" ainda não é referenciado no código.



Podemos gerar o getter e o setter para o atributo "target" recorrendo a **Source > Generate Getters and Setters** ou então à funcionalidade **Quick Fix (CTRL+1)** do Eclipse.

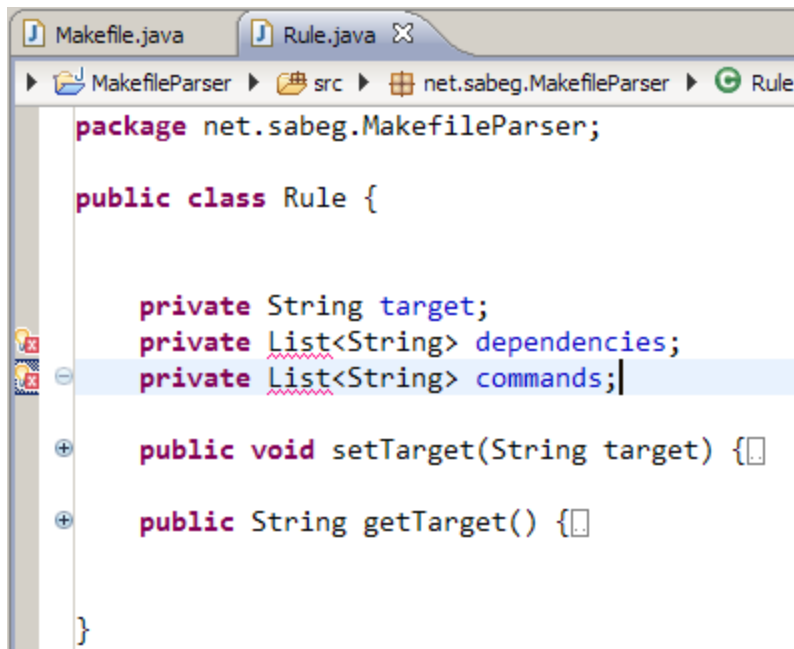
Usando o Quick Fix obtemos algumas sugestões acerca do que podemos fazer para corrigir o problema. Uma dessas sugestões é gerar os getters e setters para o atributo.



A utilização do Quick Fix pressupõe que se entende o problema e que sabemos o que queremos. Deve ser usado como um instrumento para nos facilitar a correcção de erros e não como um "oráculo" que sabe resolver todos os problemas que surjam no código.

Novos atributos que requerem imports

Adicionando novos atributos sem colocar os imports necessários levam a que o Eclipse apresente erros.



```
package net.sabeg.MakefileParser;

public class Rule {

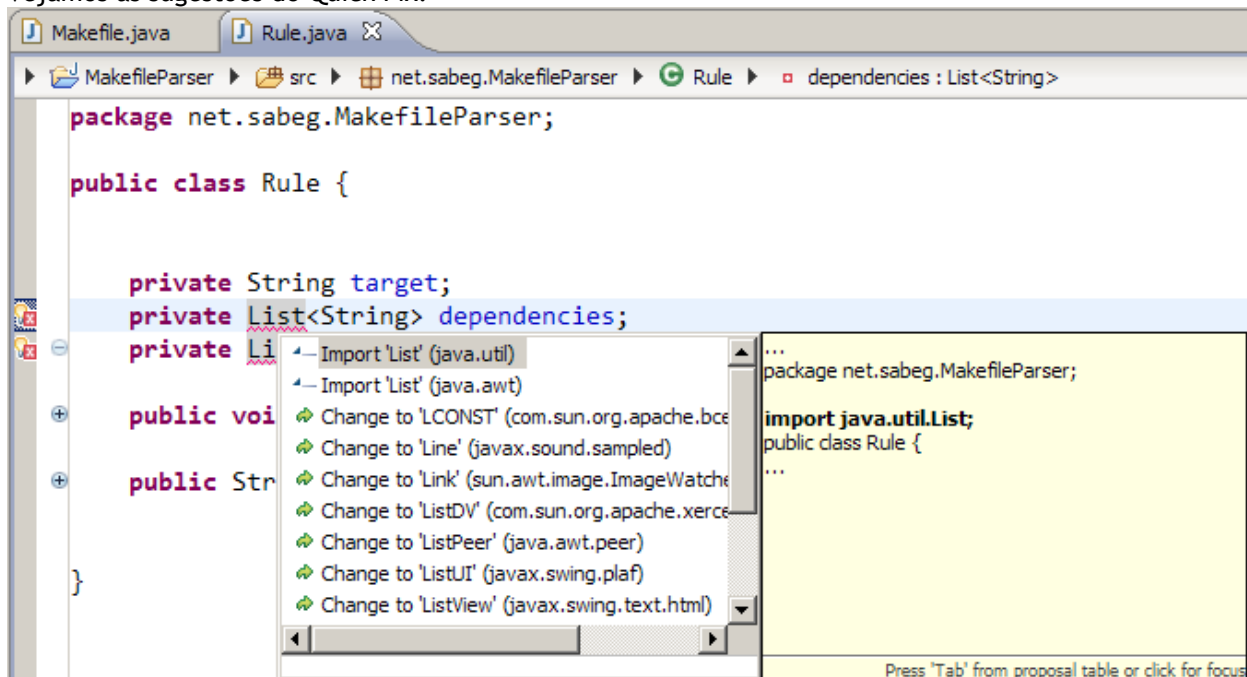
    private String target;
    private List<String> dependencies;
    private List<String> commands;

    public void setTarget(String target) {}

    public String getTarget() {}

}
```

Vejamos as sugestões do Quick Fix:



```
package net.sabeg.MakefileParser;

public class Rule {

    private String target;
    private List<String> dependencies;
    private List<String> commands;

    public void setTarget(String target) {}

    public String getTarget() {}

}
```

... package net.sabeg.MakefileParser;
import java.util.List;
public class Rule {
...
}

Press 'Tab' from proposal table or click for focus

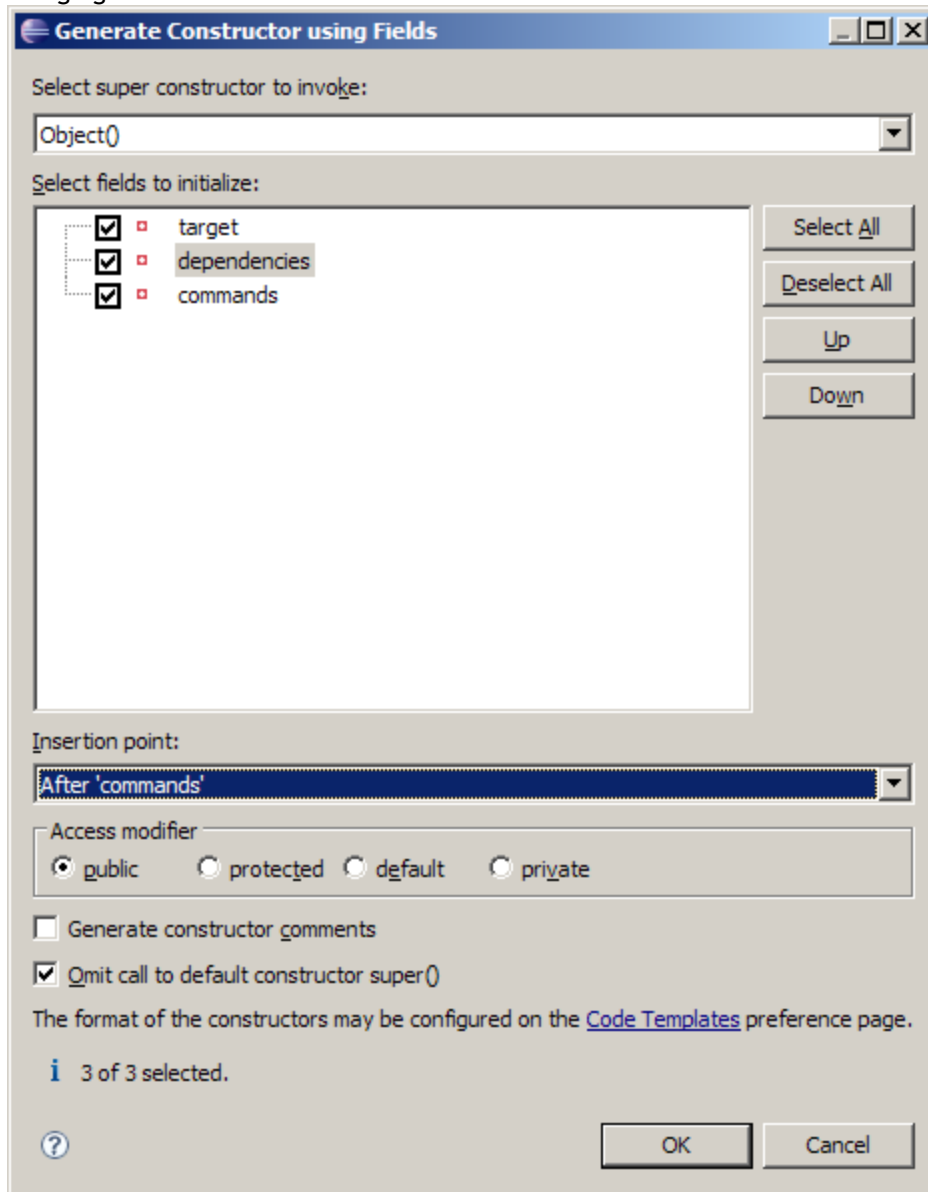
Supondo que pretendemos uma lista de Strings, onde o conceito de lista é aquele que encontramos nas Java Collections (java.util.List), apenas uma das sugestões é adequada para resolver este problema. Aqui podemos ver que a utilização do Quick Fix deve ser sempre feita com base numa escolha informada. Aceitando a sugestão "Import 'List' (java.util)" a linha de import apropriada é adicionada ao ficheiro e o Eclipse deixa de assinalar o erro.

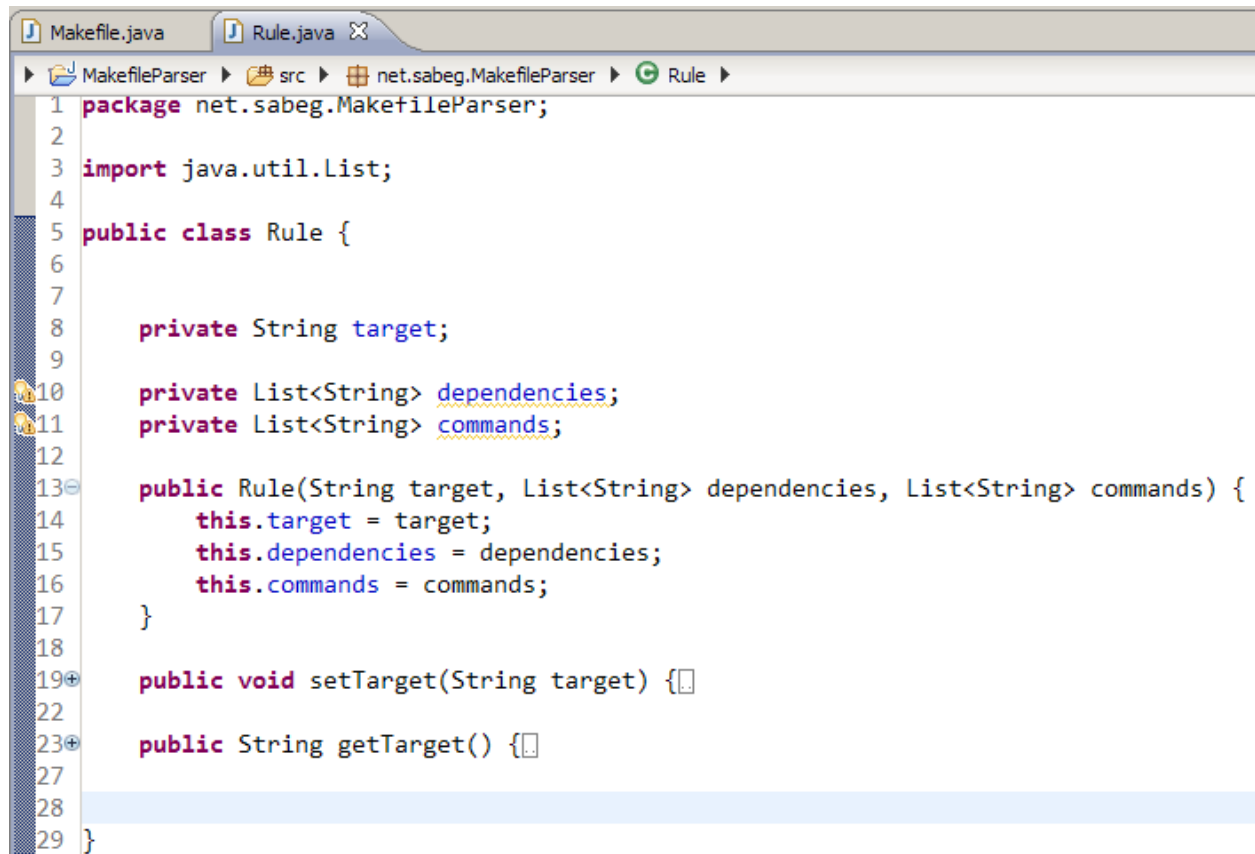
É frequente queremos ver os números de linha no código para mover o cursor para uma determinada linha. Para mostrar os números de linhas no código basta abrir o menu de contexto na margem do editor e activar a opção **Show Line Numbers**. Para mover o cursor para uma determinada existe o atalho **Goto Line (CTRL+L)**.

Gerar construtor a partir de atributos

Agora que temos definidos alguns atributos, podemos gerar um construtor que os receba como argumentos.

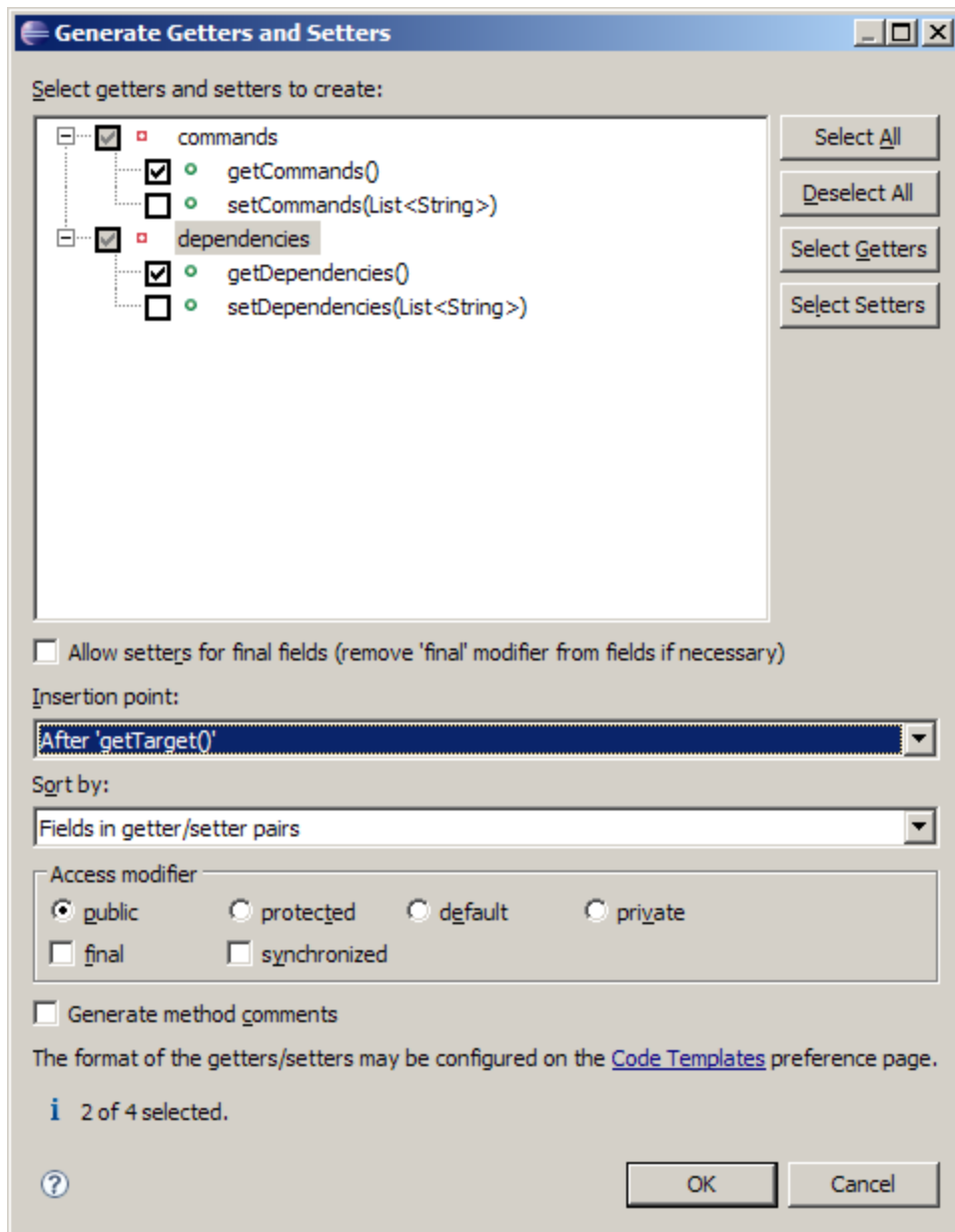
Através de **Source > Generate Constructor using Fields** podemos escolher que atributos recebe o construtor, o seu modificador de acesso, a ordem dos atributos e o ponto de inserção na classe do código gerado.





```
1 package net.sabeg.MakefileParser;
2
3 import java.util.List;
4
5 public class Rule {
6
7     private String target;
8
9     private List<String> dependencies;
10    private List<String> commands;
11
12    public Rule(String target, List<String> dependencies, List<String> commands) {
13        this.target = target;
14        this.dependencies = dependencies;
15        this.commands = commands;
16    }
17
18    public void setTarget(String target) {}
19
20    public String getTarget() {}
21
22    }
23
24    }
```

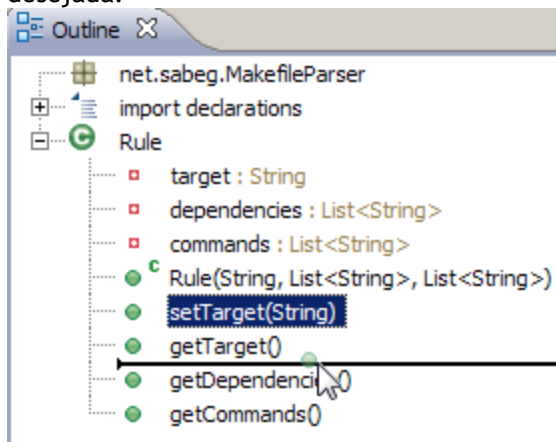
O Eclipse ainda está dar avisos sobre os atributos do tipo List<String>. Vamos criar apenas getters para estes atributos. Para isso usamos **Source > Generate Getters and Setters** pois permite seleccionar os getters e/ou setters a gerar para um ou mais atributos:



```
private List<String> dependencies;  
private List<String> commands;  
  
public Rule(String target, List<String> dependencies, List<String> commands) {  
    this.target = target;  
    this.dependencies = dependencies;  
    this.commands = commands;  
}
```

Por vezes podemos querer listar todos elementos de um ficheiro. No caso de um ficheiro Java teremos packages, imports, classe, métodos e atributos. Na vista **Outline (Window > Show View > Outline)** temos esta lista disponível bem como a possibilidade de filtrar os elementos que aparecem nessa vista. É possível também a reordenação de código usando drag and drop. Suponhamos que queríamos o

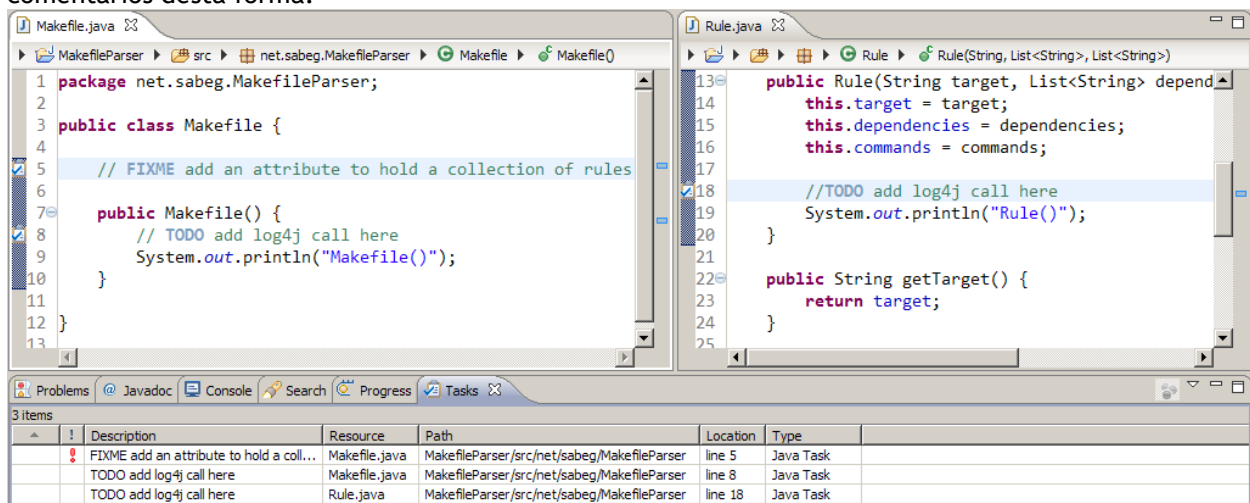
método setTarget depois do getTarget. Para tal, basta arrastar o nome do método para a localização desejada:



O código no ficheiro Java é actualizado automaticamente para se sincronizar à sua nova estrutura, definida no Outline. Não é necessário qualquer copy/paste.

Tarefas nos comentários

Muitas vezes um comentários contém indicações de tarefas que devem ser feitas ou problemas a ser resolvidos. O Eclipse suporta o conceito de **Task Tags** (**Window > Preferences > search filter = "task tags"**). O mais interessante é que estas tarefas aparecem na vista **Tasks** (**Window > Show View > Tasks**) e têm associadas a si uma prioridade. Podemos ver todas as tarefas do projecto que estão em comentários desta forma.



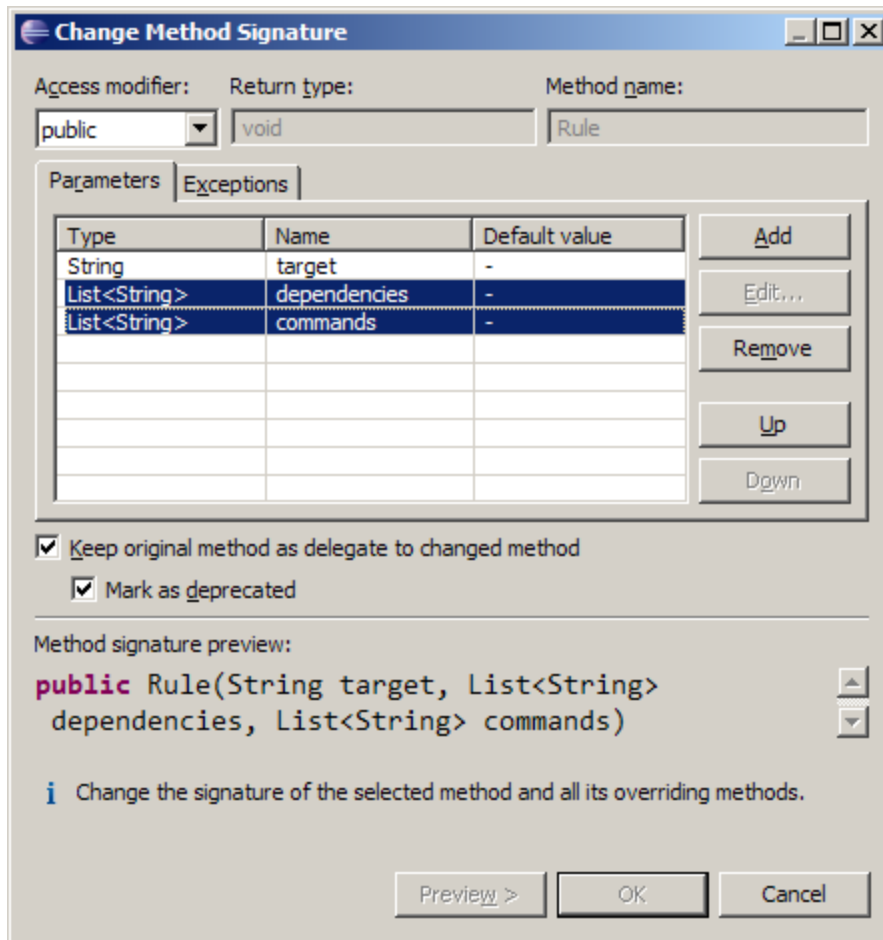
Atenção que para a lista de tarefas esteja sincronizada com as Task Tags nos comentários é necessário que estas estejam activas (**Enable searching for Task Tags** nas preferências do Eclipse) e que o projecto esteja configurado para fazer build automaticamente (**Project > Build Automatically**).

Alterar a assinatura de um método

Queremos também alterar o construtor de Rule para receber apenas o target da regra.

Para alterar a assinatura de métodos podemos usar **Refactor > Change Method Signature (ALT+SHIFT+C)**. Fazendo esta modificação através de uma funcionalidade de Refactor garantimos que o Eclipse vai tentar localizar as zonas do código que tenham que ser alteradas devido à modificação. O

Eclipse irá emitir um aviso caso estas alterações não possam ser feitas automaticamente ou surjam problemas a resolver.



The image shows the 'Change Method Signature' dialog box in the Eclipse IDE. The dialog has a title bar with the Eclipse logo and the text 'Change Method Signature'. It contains several input fields and a table for parameters.

Access modifier: **public** (dropdown menu)
Return type: **void** (text field)
Method name: **Rule** (text field)

Parameters tab is selected. The table below lists the parameters:

Type	Name	Default value
String	target	-
List<String>	dependencies	-
List<String>	commands	-

Buttons on the right side of the table: Add, Edit..., Remove, Up, Down.

Checkboxes:
☒ Keep original method as delegate to changed method
☒ Mark as deprecated

Method signature preview:
public Rule(String target, List<String> dependencies, List<String> commands)

Information icon (i): Change the signature of the selected method and all its overriding methods.

Buttons at the bottom: Preview >, OK, Cancel.

Change Method Signature

Access modifier: **public** Return type: **void** Method name: **Rule**

Parameters Exceptions

Type	Name	Default value
String	target	-

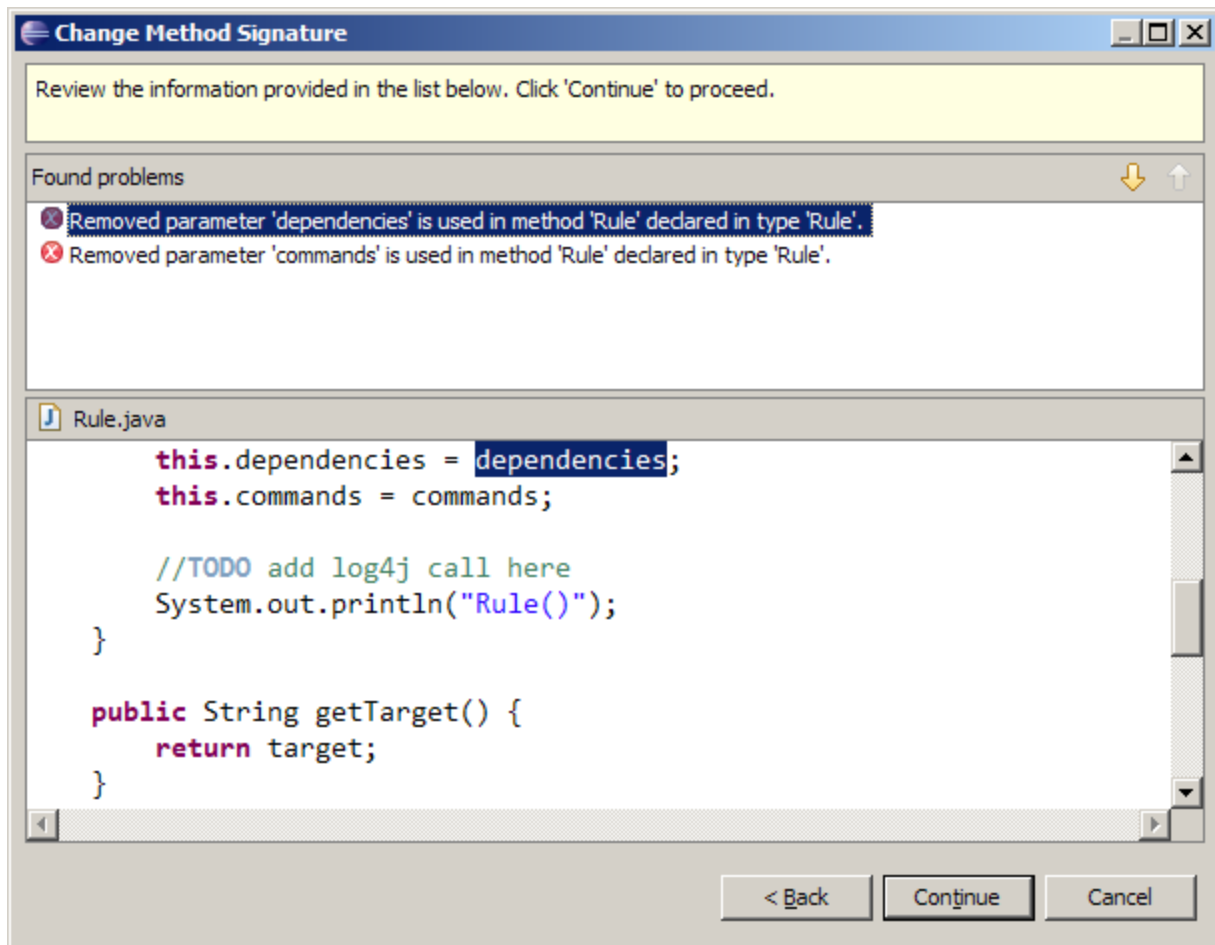
Add Edit... Remove Up Down

☒ Keep original method as delegate to changed method
☒ Mark as deprecated

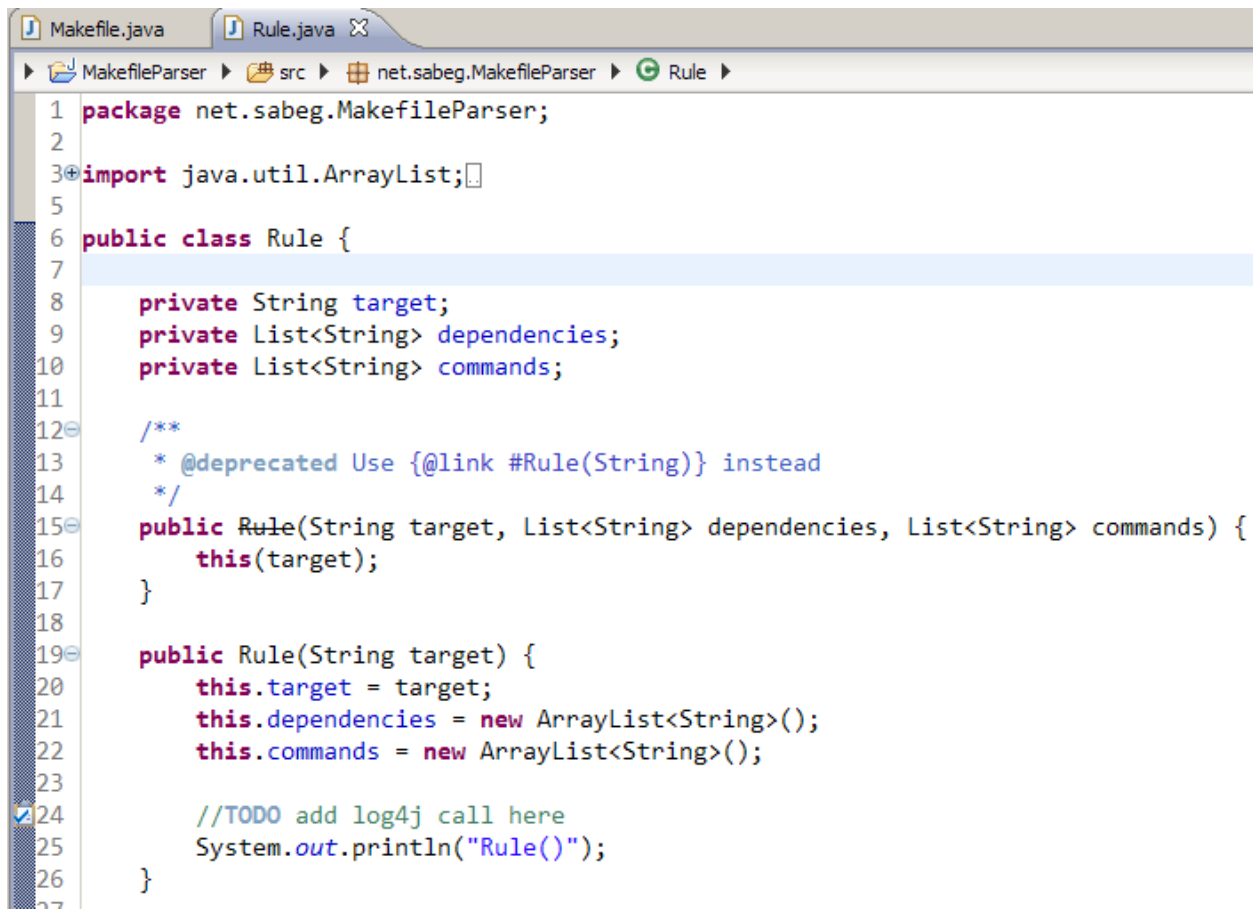
Method signature preview:
public Rule(String target)

Preview > OK Cancel

Ao clicar em OK irá surgir um aviso, pois os argumentos "dependencies" e "commands" serão removidos do construtor.



O construtor antigo é marcado como deprecated e passa a delegar as suas responsabilidades ao novo construtor.



```
1 package net.sabeg.MakefileParser;
2
3 import java.util.ArrayList;
4
5
6 public class Rule {
7
8     private String target;
9     private List<String> dependencies;
10    private List<String> commands;
11
12    /**
13     * @deprecated Use {@link #Rule(String)} instead
14     */
15    public Rule(String target, List<String> dependencies, List<String> commands) {
16        this(target);
17    }
18
19    public Rule(String target) {
20        this.target = target;
21        this.dependencies = new ArrayList<String>();
22        this.commands = new ArrayList<String>();
23
24        //TODO add log4j call here
25        System.out.println("Rule()");
26    }
27 }
```

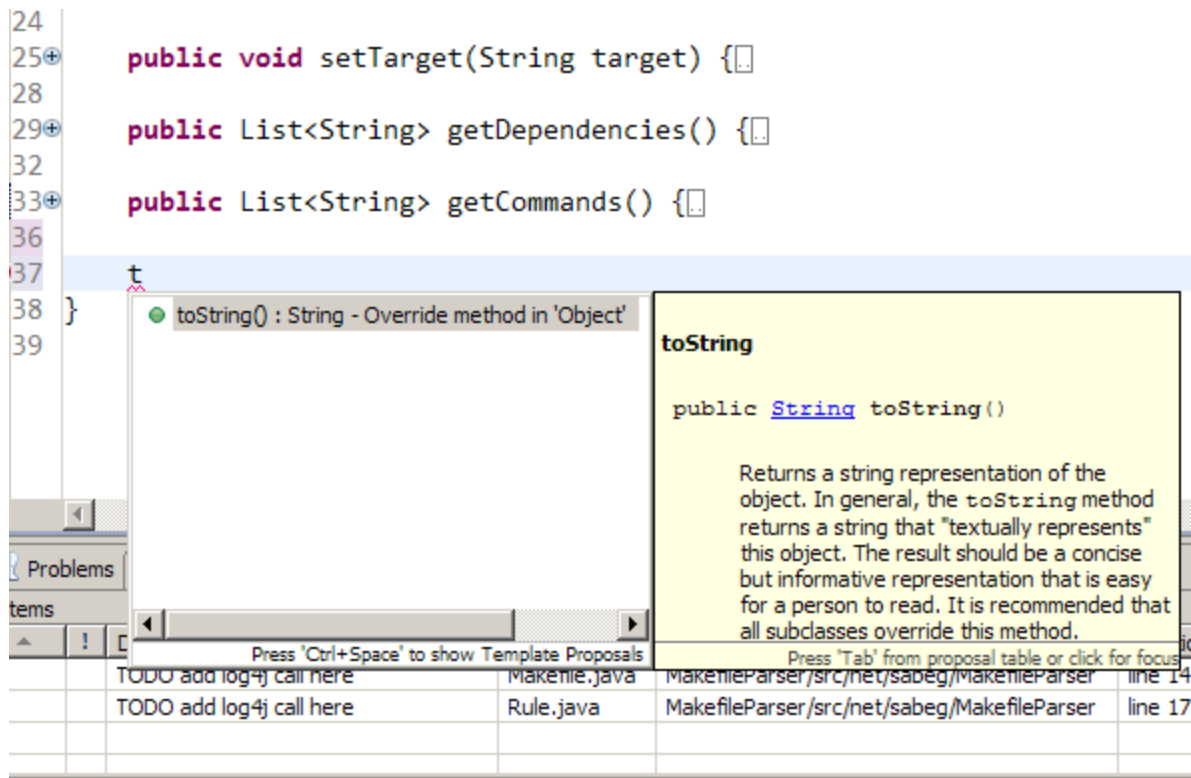
Se o construtor antigo não for mais necessário, pode ser removido.

Tal como o **Source** (**ALT+SHIFT+S**), o menu de **Refactor** (**ALT+SHIFT+T**) é muito útil e deve ser usado quando aplicável. Permite efectuar refactorização de código, isto é, alterar a estrutura do código sem modificar funcionalidade. Depois de ser feita uma refactorização o Eclipse vai tentar encontrar, no projecto, outros pontos no código que tenham que ser alterados. Ao efectuar modificações manualmente em projectos complexos podemos facilmente esquecer-nos de alterar um ou mais sítios.

Implementando mais métodos

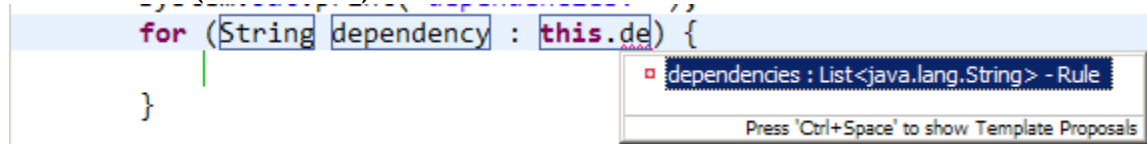
Vamos realizar a tarefa prioritária associada ao ficheiro Makefile.java, adicionando uma lista de Rules à classe Makefile. Usando o Refactor, gera-se facilmente um getter para este atributo. Em seguida, implementamos o toString nas duas classes do projecto, começando pela classe Rule. Para a classe Make o processo é análogo.

Usando o Content Assist a geração da stub para o toString é imediata.



O `toString` da `Rule` vai devolver o atributo `target` bem como as listas de dependências e comandos. Para iterar sobre as listas vamos usar um `foreach`, para o qual o Eclipse disponibiliza um **Code Template**. Um Code Template é um bloco de código, em geral parametrizável, usado frequentemente. Cada Code Template tem um nome que sintetiza o bloco de código generalizado. No caso do `foreach` é...`"foreach"`. Existem inúmeros Code Templates e podemos inclusive criar novos (**Window > Preferences > Java > Editor > Templates**).

Para introduzir o Code Template do `foreach` basta digitar alguns caracteres de `"foreach"` e invocar o **Content Assist (CTRL+SPACE)**.



Note-se que mesmo nos parâmetros do template o Content Assist está disponível!

Depois de ambos os `toString` estarem implementados, adicionamos também métodos para adicionar dependências e comandos a `Rules` bem como `Rules` a `Makefiles`. Criamos então uma classe de teste, `Test.java`, também na package das restantes classes, que contenha o método `Main`. Tudo isto pode ser feito com eficiência usando os atalhos já descritos neste documento.

Garantindo que a vista **Console** está presente (**Window > Show View > Console**), executamos a aplicação com **Run > Run (CTRL+F11)**:

The screenshot shows an IDE with three tabs: Makefile.java, Rule.java, and Test.java. The Test.java tab is active, displaying the following code:

```
1 package net.sabeg.MakefileParser;
2
3 public class Test {
4
5     public static void main(String[] args) {
6
7         Rule r1 = new Rule("hello").addDependency("hello.o").addCommand("compile hello.o -o hello");
8         Rule r2 = new Rule("hello.o").addDependency("hello.c").addCommand("compile -o hello.o -c hello.c");
9         Rule r3 = new Rule("clean").addCommand("delete hello.o").addCommand("delete hello");
10
11         Makefile m = new Makefile().addRule(r1).addRule(r2).addRule(r3);
12
13         System.out.println(m);
14     }
15 }
```

Below the code editor, the console window shows the output of the application:

```
<terminated> Test [Java Application] C:\Java\jdk1.5.0_17\bin\javaw.exe (2009/03/15 01:46:57)
Rule ()
Rule ()
Rule ()
Makefile ()
hello: hello.o
    compile hello.o -o hello

hello.o: hello.c
    compile -o hello.o -c hello.c

clean:
    delete hello.o
    delete hello
```

Renomear classes

Pretende-se renomear a classe Rule para MakefileRule, mesmo depois do projecto estar a funcionar. Para renomear a classe "Rule" para "MakefileRule" basta seleccionar o ficheiro "Rule.java", usar **Refactor > Rename (ALT+SHIFT+R)**, editar o nome e, no final, pressionar ENTER. A renomeação também pode ser feita colocando o cursor sobre o nome "Rule", em qualquer ponto do código do ficheiro "Rule.java" e usando então o Rename. São imediatamente actualizadas as referências ao nome, tanto dentro da classe como noutras classes.

```
Makefile.java
7
8 private List<MakefileRule> rules;
9
10 public Makefile() {}
11
12 public List<MakefileRule> getRules() {}
13
14 public String toString() {}
15
16 public Makefile addRule(MakefileRule rule) {}
17
18 }
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42

MakefileRule.java
29 public List<String> getDependencies() {}
30
31 public List<String> getCommands() {}
32
33 public String toString() {}
34
35 public MakefileRule addDependency(String dependency) {}
36
37 public MakefileRule addCommand(String command) {}
38
39 }
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69

Test.java
1 package net.sabeg.MakefileParser;
2
3 public class Test {
4
5     public static void main(String[] args) {
6
7         MakefileRule r1 = new MakefileRule("hello").addDependency("hello.o").addCommand("compile hello.o -o hello");
8         MakefileRule r2 = new MakefileRule("hello.o").addDependency("hello.c").addCommand("compile -o hello.o -c hello.c");
9         MakefileRule r3 = new MakefileRule("clean").addCommand("delete hello.o").addCommand("delete hello");
10
11         Makefile m = new Makefile().addRule(r1).addRule(r2).addRule(r3);
12
13     }
14 }
```

A aplicação continua a funcionar como anteriormente:

```
Problems Javadoc Console Search Progress Tasks
<terminated> Test [Java Application] C:\Java\jdk1.5.0_17\bin\javaw.exe (2009/03/15 01:59:57)
Rule ()
Rule ()
Rule ()
Makefile ()
hello: hello.o
    compile hello.o -o hello

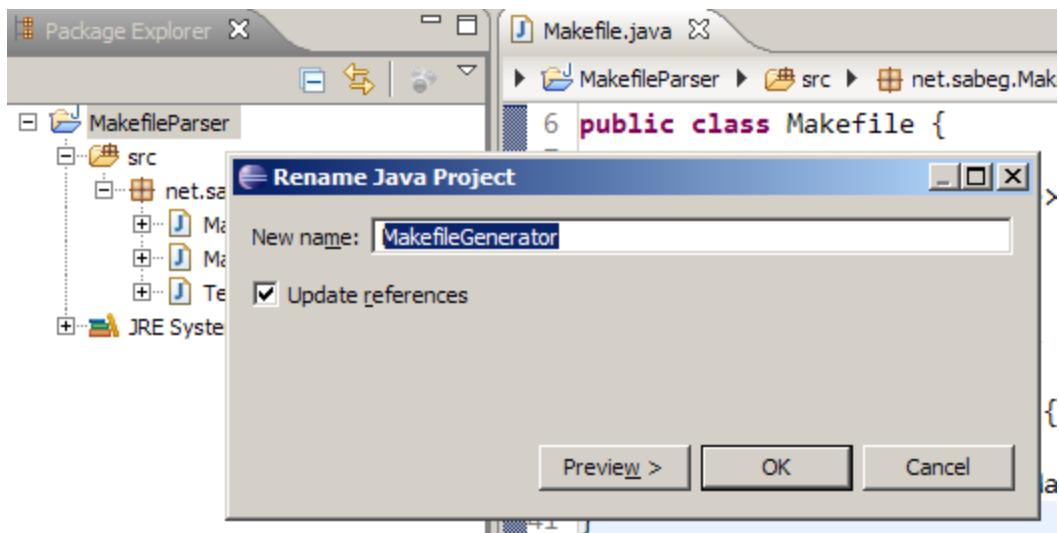
hello.o: hello.c
    compile -o hello.o -c hello.c

clean:
    delete hello.o
    delete hello
```

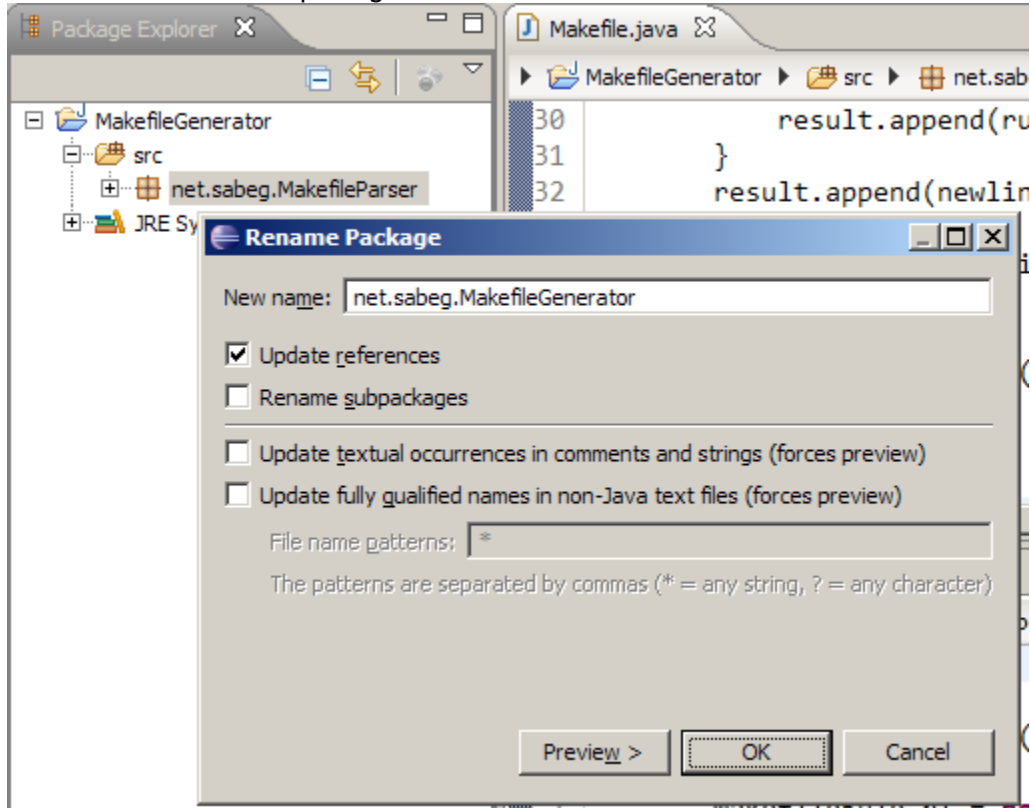
O Rename pode ser aplicado a projecto, classes, métodos, atributos, variáveis locais, entre outros elementos. O Eclipse encarrega-se de actualizar, se possível, as zonas de código onde esses elementos eram referenciados com o nome antigo, passando a usar o novo nome.

Renomear o projecto

Tal como muitos outros elementos também o nome de um projecto podem ser renomeado. Uma vez que o nome "MakefileGenerator" não é o mais adequado para o projecto, ir-se-há mudar este nome. Seleccionado a raiz do projecto no **Package Explorer** e aplicando um **Rename** pode mudar-se facilmente o nome do projecto.



Renomeamos também a package:



Navegando pelo código

Destacam-se algumas funcionalidade do Eclipse que contribuem para maximizar a produtividade do utilizador:

- Um atalho muito útil é o **Navigate > Goto Declaration (F3)**. Colocando o cursor sobre uma chamada a um método e carregando F3, imediatamente é aberto um editor na linha que contém a declaração do método, mesmo que seja noutra ficheiro, noutra package.
- Em qualquer editor, o **Java Breadcrumb** mostra-nos o caminho do elemento Java actualmente em edição, desde a raiz do projecto, bem como o nome e tipo de cada parte desse caminho:

```

MakefileGenerator ▸ src ▸ net.sabeg.MakefileGenerator ▸ Test ▸ main(String[])
1 package net.sabeg.MakefileGenerator;
2
3 public class Test {
4
5     public static void main(String[] args) {
6
7         MakefileRule r1 = new MakefileRule("hello").addDependency
8         MakefileRule r2 = new MakefileRule("hello.o").addDepender

```

- Caso a vista Javadoc esteja activa (**Window > Show View > Javadoc**) podemos ver em tempo real a documentação Javadoc para o elemento de código actualmente sob o cursor:

```

42
43     StringBuilder result = new StringBuilder();
44
45     result.append(target).append(":");
46
47     for (String dependency : this.dependencies) {

```

Problems @ Javadoc Console Search Progress Tasks

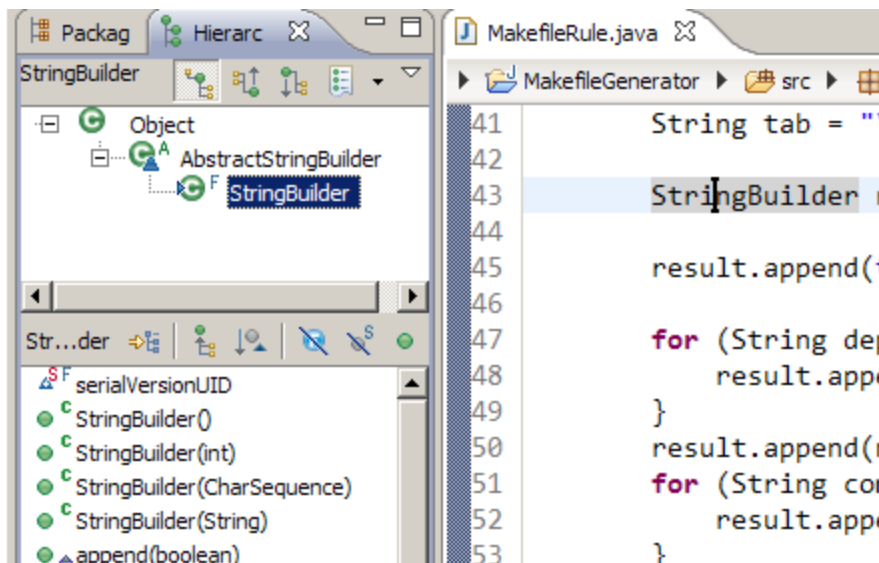
java.lang.StringBuilder

A mutable sequence of characters. This class provides an API compatible with `StringBuffer`, but `StringBuffer` in places where the string buffer was being used by a single thread (as is generally the case) as it will be faster under most implementations.

The principal operations on a `StringBuilder` are the `append` and `insert` methods, which append or insert the characters of that string to the string builder. The `append` method appends the characters at the specified point.

Em alternativa, basta clicar em **Edit > Show Tooltip Description (F2)** quando o cursor está sobre a expressão pretendida.

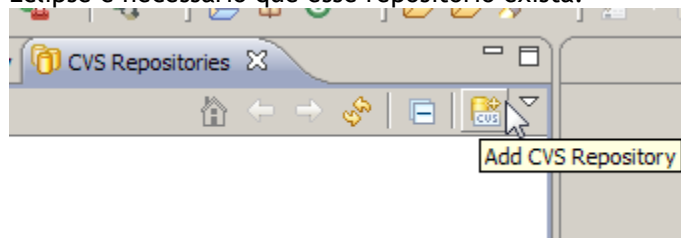
- Maximizar um editor ou vista pode ser feito com o atalho **CTRL+M**. Para regressar ao tamanho normal, premir **CTRL+M** novamente.
- Muitas vezes é preciso conhecer a hierarquia de classes de uma classe particular. A vista **Hierarchy (Window > Show View > Hierarchy)** permite fazer isso. Selecciona-se um nome no código e, através do atalho **Open Type Hierarchy (F4)**, podem-se explorar as superclasses e subclasses da classe a que pertence essa classe. É possível filtrar a hierarquia de modo a mostrar apenas as superclasses ou subclasses.



Existe uma vista semelhante, a **Call Hierarchy**, que funciona de modo análogo para as invocações de métodos.

Adicionar o projecto a um repositório CVS

Para realizar operações com CVS é necessário dar ao Eclipse a conhecer a localização de um ou mais repositórios. A vista **CVS Repositories** (Window > Show View > Other > **CVS Repositories**) permite gerir os repositórios conhecidos pelo Eclipse. Antes de adicionar a localização de um repositório CVS ao Eclipse é necessário que esse repositório exista.



Esta vista está incluída numa perspectiva (**Perspective**) dedicada a trabalhar com CVS, **CVS Repository Exploring**. Uma perspectiva agrupa várias vistas e editores, em geral relacionados entre si. A perspectiva natural para codificação em Java é a **Java Perspective**.

Ao adicionar um repositório é importante escolher o tipo de conexão apropriada:

Add CVS Repository

Add a new CVS Repository to the CVS Repositories view

Location

Host: sigma.ist.utl.pt

Repository path: /afs/ist.utl.pt/users/X/Y/istIDXY/essd-cvs

Authentication

User: istIDXY

Password:

Connection

Connection type: extssh

☒ Use default port

☐ Use port:

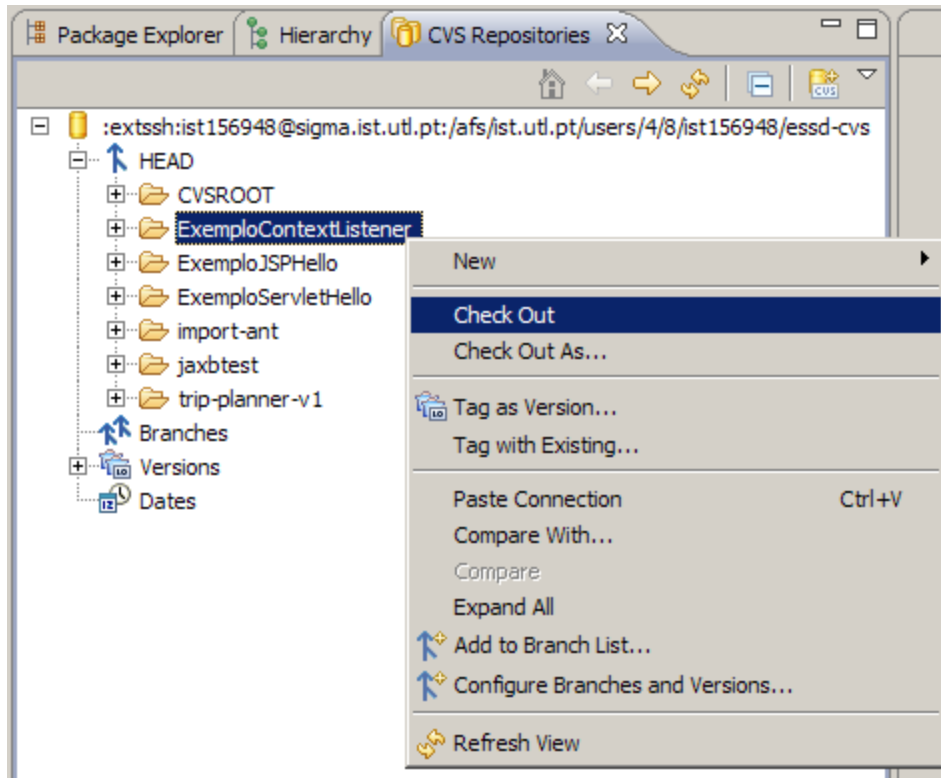
☒ Validate connection on finish

☐ Save password (could trigger secure storage login)

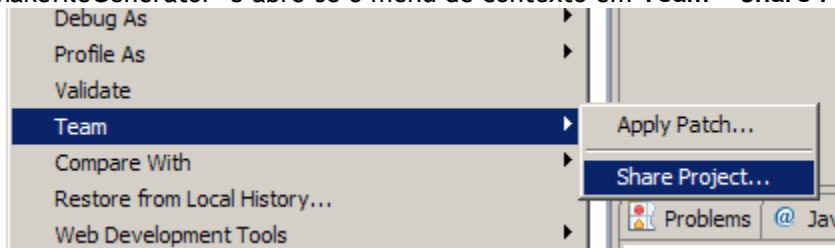
[Configure connection preferences...](#)

Finish Cancel

Se o repositório for adicionado com sucesso, aparece na vista CVS Repositories, podendo ser explorados os seus módulos (tanto na HEAD como por Branches, Versions ou Dates) antes de fazer checkout de um deles:

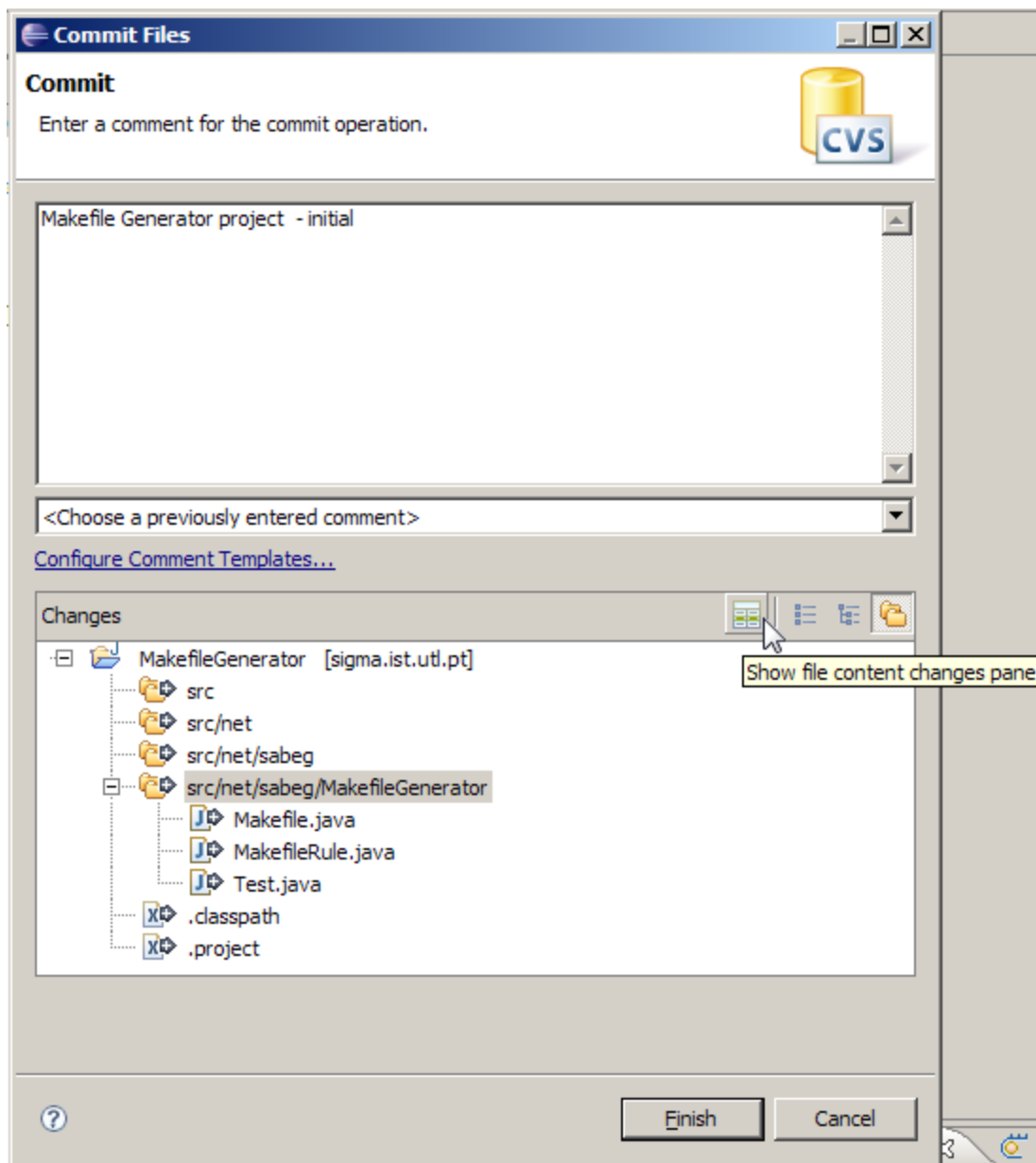


Neste caso não nos interessa fazer o checkout de um módulo mas sim adicionar o projecto local ao repositório. Para tal volta-se ao Package Explorer, selecciona-se a raiz do projecto "MakefileGenerator" e abre-se o menu de contexto em **Team > Share Project**:



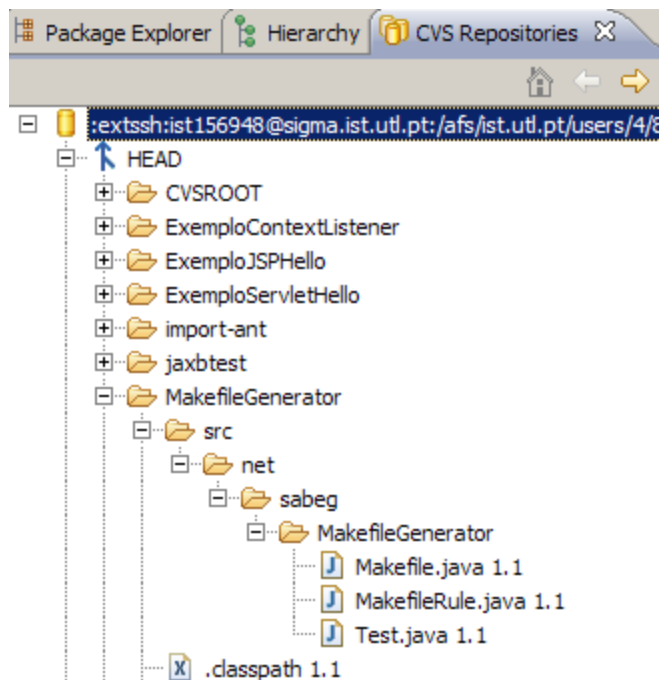
Selecciona-se o repositório previamente adicionado ao Eclipse (**Use existing repository location**) e em seguida escolhemos o nome para o módulo onde ficará localizado o projecto. Podemos usar o nome do projecto como nome do módulo, definir um nome diferente para o módulo ou usar um módulo já existente. Decidiu-se usar o nome de projecto como nome do módulo.

Surge em seguida o **Commit Wizard** que auxilia na selecção dos recursos a serem enviados para o repositório, finalizando-se a operação com a introdução da mensagem associada ao **Commit**:



Clicando em **Show file content changes pane** surge o editor **Java Source Compare**, que permite ver as diferenças entre um ficheiro que estamos prestes a enviar para o repositório e a sua versão no repositório.

Após clicar em **Finish**, o Eclipse executa o commit do projecto. Caso a operação tenha sucesso voltamos à vista **CVS Repositories**, seleccionamos o nome do repositório e carregamos em **Refresh (F5)**. Serão reanalisados os módulos do repositório e aparecerá lá o novo módulo "MakefileGenerator".

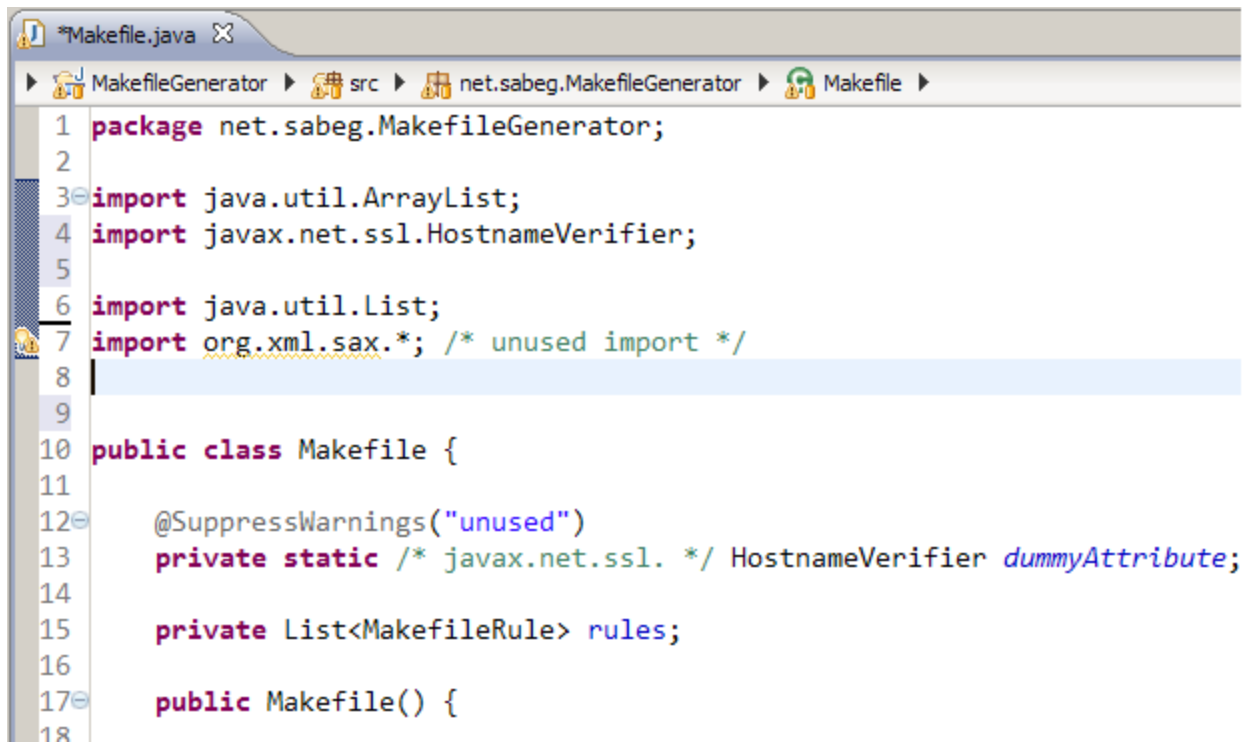


Trabalho em equipa

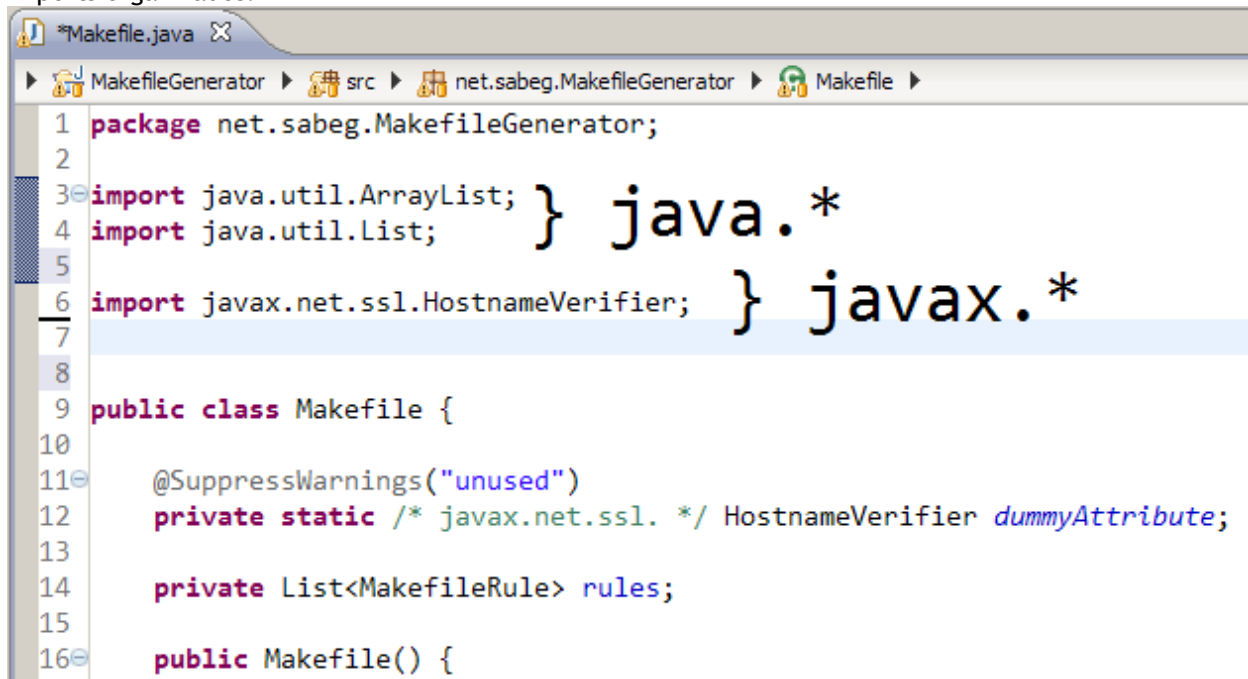
Quando um projecto é partilhado por várias pessoas é ainda mais importante seguir um conjunto comum de regras de codificação e formatação do código fonte.

O Eclipse ajuda nesta tarefa, permitindo definir o estilo de codificação em **Window > Preferences > Java > Code Style**. Existem várias funcionalidades que podem ser configuradas: **Clean Up**, **generated Code Templates**, **Formatter** (perfis de formatação automática de código) e **Organize Imports**.

O atalho **Source > Organize Imports (CTRL+SHIFT+O)** permite aceder organizar os imports, ordenando-os segundo um determinado critério (preferências de Code Style > Organize Imports) e remove imports não utilizados:



Imports organizados:

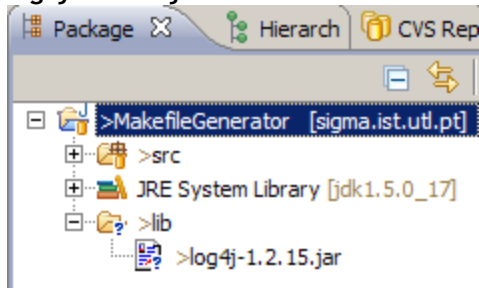


Manter uma formatação do código consistente é também um aspecto ao qual o Eclipse dá suporte. Depois de se definir um perfil de formatação (preferências de Code Style > Formatter), pode usar-se o atalho **Source > Format (CTRL+SHIFT+F)** para formatar o ficheiro fonte segundo esse perfil. Se apenas pretendermos corrigir a indentação (pois a formatação não está disponível para todos os tipos de ficheiro) existe o atalho **Source > Correct Indentation (CTRL+I)**.

Adicionar logging ao projecto

Depois de adicionar o projecto ao CVS e de conhecer algumas funcionalidades para manter o código organizado, vamos executar as duas tarefas ainda pendentes no projecto "MakefileGenerator", substituir as chamadas a System.out por métodos de logging da biblioteca [log4j](#).

Cria-se uma pasta **lib** na raiz do projecto e coloca-se lá o Jar da biblioteca que vamos usar, **log4j-1.2.15.jar**.



Neste momento o Eclipse ainda não reconhece as classes dentro do Jar, pois é necessário dar-lhe a conhecer essa informação.

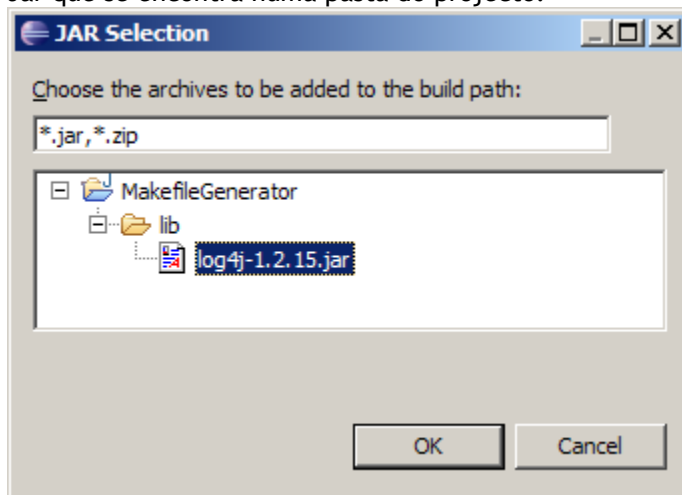
Seleccionamos a raiz do projecto no Package Explorer e navegamos até **File > Properties (ALT+ENTER)**.

No diálogo que surge, **Properties for MakefileGenerator**, encontramos todas as propriedades do projecto.

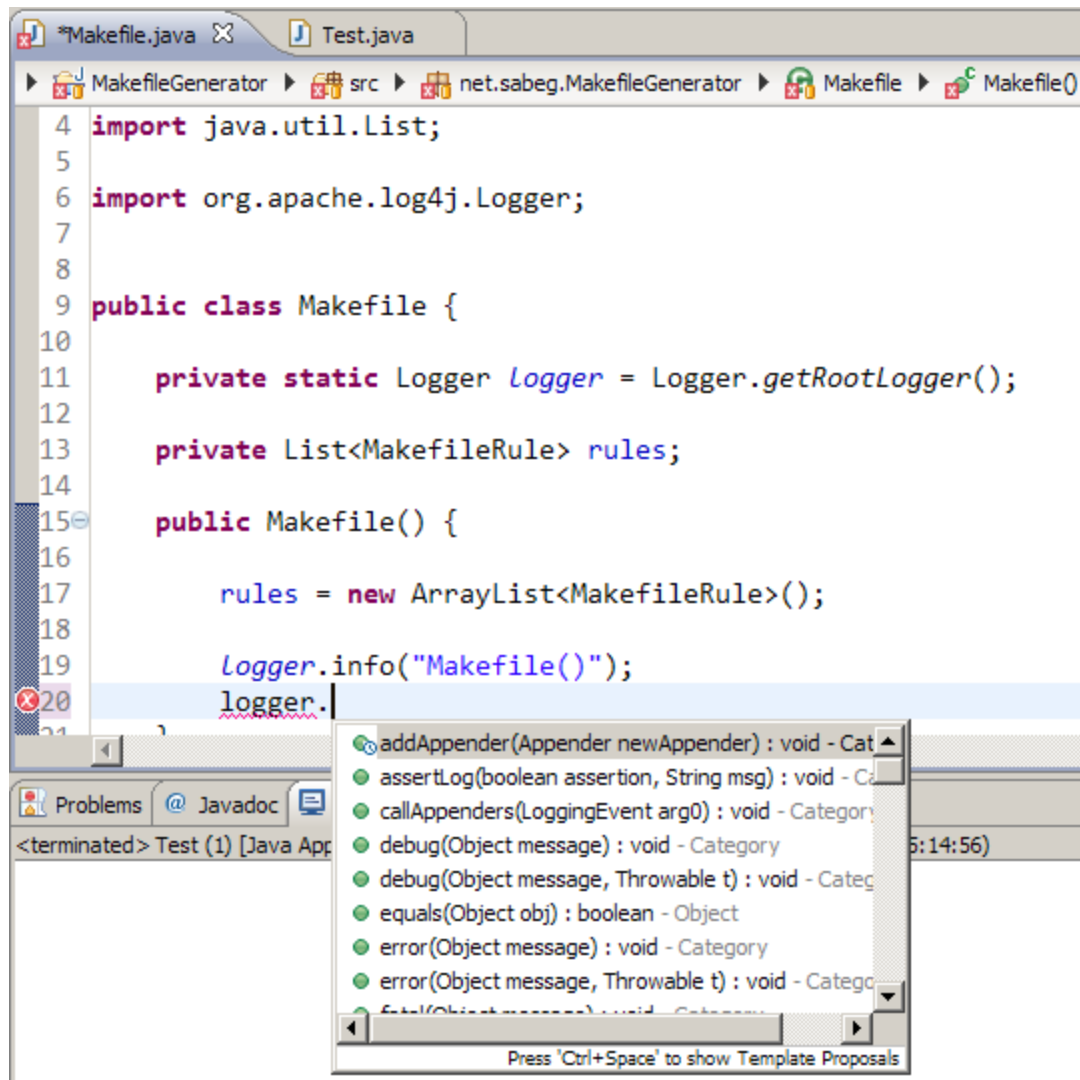
Estamos interessados em particular nas opções de **Java Build Path**.

Aí podemos configurar, entre outros aspectos, todas as pastas que contém código fonte e bibliotecas (localizadas em várias fontes).

Para adicionar o log4j-1.2.15.jar ao projecto basta ir para a tab **Libraries** e clicar **Add JARs**, pois é um Jar que se encontra numa pasta do projecto:



O código de logging log4j é reconhecido pelo Eclipse, em particular pelo Content Assist:



Adicionando também código de logging à classe MakefileRule, bem como código para configurar o logger na classe Test, terminamos as duas tarefas.

Executando o projecto, obtemos:

The screenshot shows the Eclipse IDE with two Java files open: `Makefile.java` and `MakefileRule.java`. The `Makefile.java` file contains a `Test` class with a `main` method that configures a `BasicConfigurator` and creates three `MakefileRule` objects. The `MakefileRule.java` file contains a `MakefileRule` class with private fields for `target`, `dependencies`, and `commands`, and a `MakefileRule` constructor that initializes these fields. The console output shows the execution of the `Test` class, which prints the output of the `MakefileRule` objects.

```
3 import org.apache.log4j.BasicConfigurator;
4
5 public class Test {
6
7     public static void main(String[] args) {
8
9         BasicConfigurator.configure();
10
11         MakefileRule r1 = new MakefileRule("h
12         MakefileRule r2 = new MakefileRule("h
13         MakefileRule r3 = new MakefileRule("c
14
10 private static Logger logger = Logger.getRootLogger();
11
12 private String target;
13 private List<String> dependencies;
14 private List<String> commands;
15
16 public MakefileRule(String target) {
17     this.target = target;
18     this.dependencies = new ArrayList<String>();
19     this.commands = new ArrayList<String>();
20
21     logger.info("MakefileRule()");
```

```
<terminated> Test (1) [Java Application] C:\Java\jdk1.5.0_17\bin\javaw.exe (2009/03/15 15:28:07)
0 [main] INFO root - MakefileRule()
0 [main] INFO root - MakefileRule()
0 [main] INFO root - MakefileRule()
0 [main] INFO root - Makefile()
hello: hello.o
    compile hello.o -o hello

hello.o: hello.c
    compile -o hello.o -c hello.c

clean:
    delete hello.o
    delete hello
```

Se outras bibliotecas forem necessárias, qualquer que seja o seu tipo, é importante configurar correctamente o build path para que o Eclipse reconheça as novas classes. Quando uma classe não é reconhecida ou não surge no Content Assist é porque, em geral, o build path não está bem configurado.

Procurar texto e não só

No Eclipse, a funcionalidade básica de procura e substituição de texto invoca-se usando `Edit > Find/Replace` (CTRL+F).

No entanto o IDE tem uma funcionalidade de procura muito mais completa e poderosa em `Search > Search` (CTRL+H).

As procuras efectuadas usando a `Search` podem extender-se não só ao ficheiro corrente mas a todo o projecto.

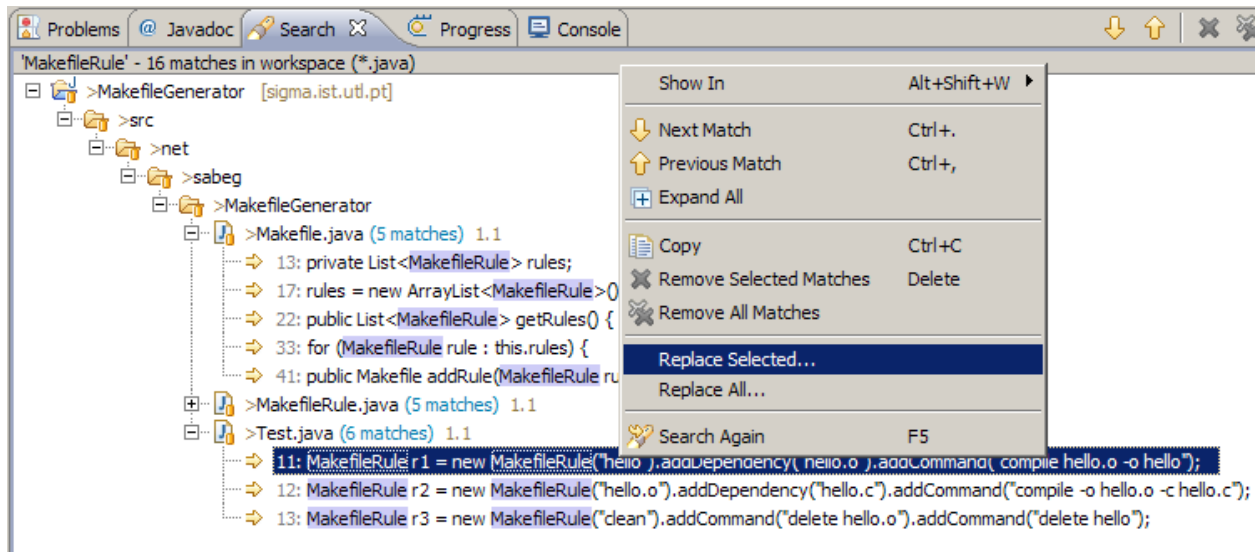
Um dos tipos de procura é a `Java Search`, que procura por elementos Java como tipos, métodos, construtores, packages e atributos.

Outro é a `File Search`, que permite definir os padrões de nomes de ficheiros onde procurar. Existem inúmeras opções para as procuras pelo que não é possível descrevê-las aqui em detalhe.

São aceites expressões regulares nas procuras.

Os resultados de uma `Search` são apresentados na vista `Window > Show View > Search` (ALT+SHIFT+Q, S).

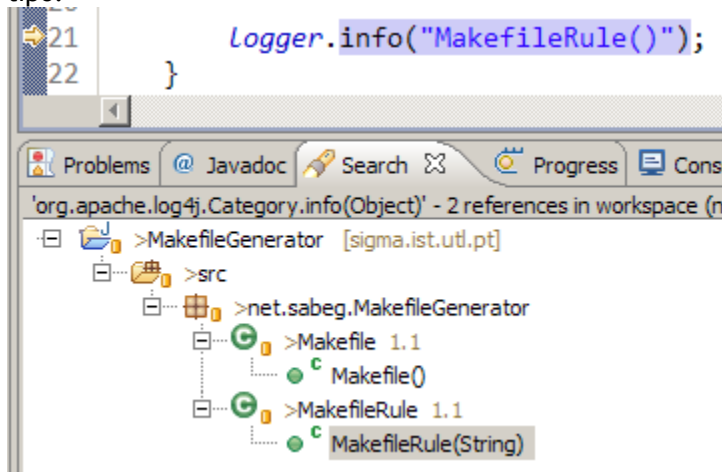
Podemos explorar os resultados e para cada ocorrência particular estão disponíveis acções como mostrar, copiar, remover ocorrências, substituir. Um duplo clique numa ocorrência leva-nos para a linha, também indicada, onde se esta se encontra.



Pode também ser útil encontrar todas as referências a um dado nome no workspace.

Usando **Search > References > Workspace (CTRL+SHIFT+G)** obtemos, na vista Search, todas as referências ao nome sob actualmente o cursor.

Na vista encontramos funcionalidades para agrupar os resultados por projecto, package, ficheiro ou tipo.



2. Atalhos

CTRL+N (New)

Criar um novo artefacto usando os wizards

CTRL+SPACE (Content Assist)

Invocar o Content Assist no contexto onde se encontra cursor

ALT+SHIFT+S (Source)

Abrir o menu Source

ALT+SHIFT+T (Refactor)

Abrir o menu Refactor

CTRL+1 (Quick Fix)

Invocar a funcionalidade Quick Fix na linha actual

CTRL+L

Ir para uma determinada linha no ficheiro

CTRL+M (Maximize)

Alterna o estado de Maximizado da vista ou editor

ALT+SHIFT+Q , O (Outline)

Abre a vista de Outline

ALT+SHIFT+Q , T (Hierarchy)

Abre a vista de hierarquias de classes

ALT+SHIFT+Q , P (Package Explorer)

Abre a vista do explorador de packages

ALT+SHIFT+Q , S (Search)

Abre a vista de resultados de procura

CTRL+F7 (Next View)

CTRL+SHIFT+F7 (Previous View)

CTRL+F8 (Next Perspective)

CTRL+SHIFT+F8 (Previous Perspective)

F3 (Goto Declaration)

F2 (Show Tooltip Description)

F5 (Refresh contents)

Permite sincronizar uma vista com o seu conteúdo.

CTRL+SHIFT+O (Organize Imports)

Organiza os imports segundo uma certa order e remove imports não usados

CTRL+SHIFT+F (Format)

Formata o ficheiro fonte automaticamente

CTRL+SHIFT+I (Correct Indentation)

Corrige a indentação do ficheiro

ALT+ENTER (Properties)

CTRL+F (Find/Replace)

CTRL+K (Find Next)

CTRL+SHIFT+K (Find Previous)

CTRL+H (Search)

ALT+/ (Word Completion)

3. Referências

Eclipse 3.4 New Features

(part 1) <http://download.eclipse.org/eclipse/downloads/drops/R-3.4-200806172000/whatsnew3.4/eclipse-news-part1.html>

(part 2) <http://download.eclipse.org/eclipse/downloads/drops/R-3.4-200806172000/whatsnew3.4/eclipse-news-part2.html>

(part 3) <http://download.eclipse.org/eclipse/downloads/drops/R-3.4-200806172000/whatsnew3.4/eclipse-news-part3.html>

Tips for using Eclipse effectively

http://www.aspectprogrammer.org/blogs/adrian/2006/02/tips_for_using.html

Using Perspectives in the Eclipse UI

<http://www.eclipse.org/articles/using-perspectives/PerspectiveArticle.html>

Eclipsepedia

<http://wiki.eclipse.org>

Eclipse Cheat Sheet

<http://weblogs.goshaky.com/weblogs/lars/resource/eclipse.pdf>