



ECLIPSEUML STUDIO PRODUCT DOCUMENTATION

ECLIPSEUML 2.2

For

Eclipse 3.5



EclipseUML 2.2 for Eclipse 3.5 Galileo is using the latest UML 2.2 OMG specification at graphical diagram and metamodel levels and is built on the top of EclipseUML 2008 core.

Don't be confused with other tools (e.g. Topcase, Papyrus, eUML etc...) which are:

- only UML 2.2 compliant at graphical level and using GMF as background metamodel
- not integrated with Java or with Java annotations and therefore have no real modeling value in a Java project.

Contents

EclipseUML

[How to install EclipseUML..](#)

EclipseUML downloads	102
EclipseUML Packaging	103
Standalone Jee Build	125
EclipseUML Plugin	125
Zip File	126

[QuickStart with EclipseUML..](#)

EclipseUML contextual menus	134
EclipseUML Views	135
How to reverse Java Code	136

[Getting Started](#)

Class Diagram Creation	148
Classes and Interfaces	148
Attributes	149
Methods	149
Associations	150
Implementation and Inheritance	153

[Diagrams](#)

Class Diagram	154
Concept	154
Drag and Drop	154

New	157
---------------------	-------	-----

Insert	158
------------------------	-------	-----

Extend UML2	159
-----------------------------	-------	-----

Package Elements	160
----------------------------------	-------	-----

Show Hide	161
---------------------------	-------	-----

Show View	161
---------------------------	-------	-----

Property View	162
-------------------------------	-------	-----

JavaDoc	162
-------------------------	-------	-----

Refresh XMI 2.1 Editor	162
--	-------	-----

Sorting	163
-------------------------	-------	-----

Package Information	163
-------------------------------------	-------	-----

Show Hide Links	163
---------------------------------	-------	-----

Resize All Shapes	163
-----------------------------------	-------	-----

Arrange Diagram	164
---------------------------------	-------	-----

PSM – Model Driven Development	164
--	-------	-----

Presentation Mode	165
-----------------------------------	-------	-----

Navigation	167
----------------------------	-------	-----

Real-Time Synchronization	167
---	-------	-----

Refactoring	168
-----------------------------	-------	-----

Working At Package Level	168
--	-------	-----

View Selector	169
-------------------------------	-------	-----

Property Concept	169
----------------------------------	-------	-----

Association Class	169
-----------------------------------	-------	-----

Format	170
------------------------	-------	-----

Toolbar	102
Class Diagram Example	103
Sequence Diagram	125
Concept	125
4 Toolbar	126
4 Message Creation	126
4 Drag and Drop	132
5 Message Information	134
8 Reverse Engineering	135
15 Sequence Diagram extension	136
17 Sequence diagram filtering	137
17 Sequence Diagram example	139
19 Tips and tricks	140
33 Use Case Diagram	148
22 Concept	148
22 Toolbar	148
28 Extend UML2	149
33 Show Hide	149
36 Drag and Drop	150
40 UseCase Diagram Example	153
42 State Diagram	154
45 Concept	154
45 Toolbar Items	154
45 Drag and Drop	155
45 State Diagram Example	156
51 Activity Diagram	157
57 Concept	157
57 Drag and Drop	158
57 Toolbar Items	159
58 Activity Diagram Example	160
59 Communication Diagram	161
61 Concept	161
63 Drag and Drop	161
63 Toolbar Items	162
64 Communication Diagram example	162
64 Object Diagram	163
66 Concept	163
67 Toolbar Items	163
68 Component Diagram	164
68 Concept	164
71 Drag and Drop	165
71 Extend UML2	167
81 Show View	167
82 Toolbar Items	168
82 Component Diagram Example	168
85 Deployment Diagram	169
88 Concept	169
94 Toolbar Items	169
101 Drag and Drop	170

Deployment Diagram Example	171	Sequence Diagram	287
Profile Diagram	172	XMI Editor	288
Profile Diagram Example	172	Activate the Model Editor	288
Refactoring	179	Navigate in the Model Editor	289
Working with Diagrams	184	Java Metamodel Elements	291
Modeling Perspective	185	UML Metamodel Elements	297
Rearranging Diagrams	187	Multiple Projects Model	304
Font and Colors	192	Documentation Generation	309
Class Diagram Presentation	194	Model Driven Development	315
Format	198	Create AndroMDA Project	315
Show Hide Compartment	199	EJB Code Generation	325
Wires	201	Hibernate Code Generation	326
Link Thickness	204	JSF/Struts Code Generation	327
Zoom	205	JD5 – JDK 6 Support	336
Print Option	208	Enumeration	337
Export UML Diagrams as documentation	209	Generic Type	346
Customize Perspective	211	Persistence Development	351
Notes	213	Activate EclipseUML in the JPA Perspective	351
Comments	213	Create New JPA UML Classes	355
Constraint	215	Add JPA Properties	369
Select	215	Create Association between Classes	371
Label	216	Reverse an Existing Database	375
Link	217	Use Oracle 11g with EclipseUML	381
Key Bindings	218	TeamWork	387
Property View	219	Tips and Tricks	395
Documentation View	220	EclipseUML 2.2 diagrams examples	405
Reverse Engineering	223		
Reverse Engineering Java Code	226		
Merge Model	227		
Reverse Engineering JPA annotations	229		
Database Reverse	229		
Sequence Diagram Reverse	229		
Reverse Engineering Example	230		
Java Code Recognition	243		
Reverse Engineering Architecture	256		
Preferences	258		
Eclipse Preference	258		
JDT Preference	259		
Setting Preferences	261		
Global EclipseUML Preferences	262		
Class Diagram Preferences	270		
Associations	272		
Class	273		
Colors	275		
Dependency	277		
Inheritance	281		
Package	282		
PSM Code Generation	284		
Show Hide	285		
Documentation Generation	286		

How to install EclipseUML

You need to download EclipseUML from Omondo website at:

<http://www.eclipsedownload.com/download.html>

You should select either Java or JEE build when you first time install EclipseUML.

Inside the same team, you can use both Java and JEE build at the same time because the core is the same.

EclipseUML for JEE includes EclipseUML for Java + Model Driven Code generator + Dali/JPA integration

This page is composed by:

1. [Where to download the Java or the Jee EclipseUML build](#)
2. [Select your packaging](#)
3. [How to install the Stand alone JEE build jar file](#)
4. [How to install the EclipseUML plugin build jar file](#)
5. [how to install the zip file](#)

1. EclipseUML downloads:

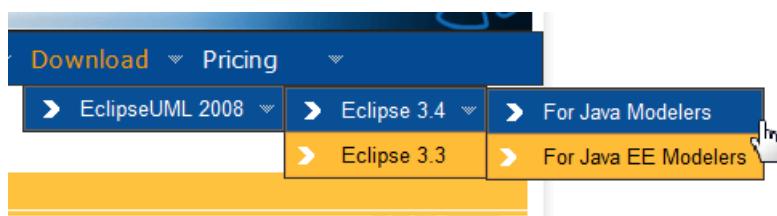
You can select Java Modelers or Java EE Modelers.

The only difference between Java and Java EE jar/zip build is that Java EE Modelers includes AndroMDA code generation templates for Eclipse.

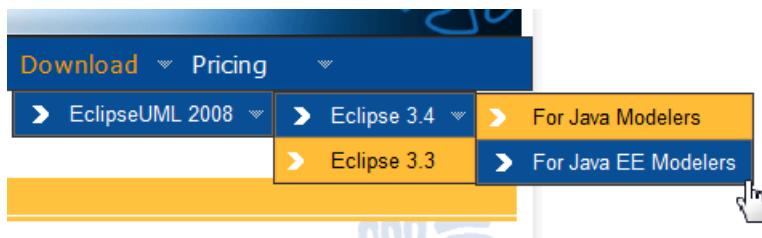
The standalone Java Jee exe file includes the Eclipse Java Jee Edition.

The Standalone Java exe film includes the Eclipse Java Edition

Select either [EclipseUML for Java Modelers](#) from our website:



Or [EclipseUML for Java EE Modelers](#):



2. EclipseUML Packaging:

EclipseUML is composed by two kinds of packaging (standalone or plugin) for two products (java or jee) which are available for download in jar or zip file format:

1. Standalone build

1. EclipseUML for Java
 1. exe file
2. EclipseUML for JEE
 1. exe file

2. EclipseUML as a plugin

1. EclipseUML for Java
 1. Jar file
 2. Zip file
2. EclipseUML for JEE
 1. Jar file
 2. Zip file

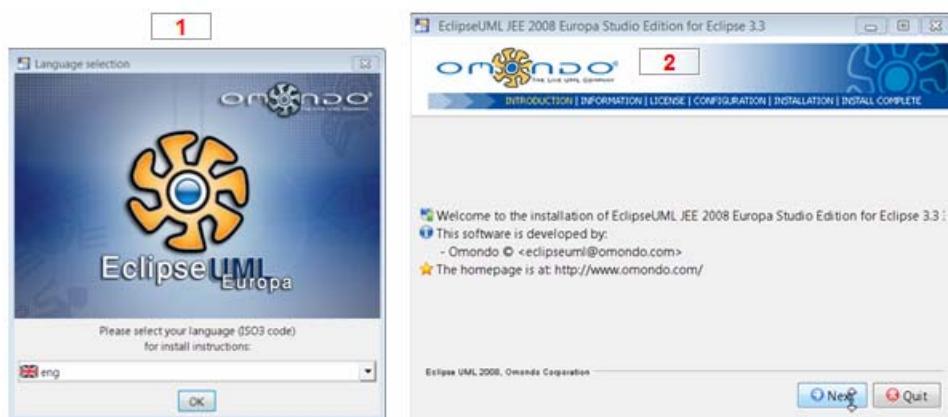
- **Standalone:** The EclipseUML download could be installed immediately. The build is composed by EclipseUML + Eclipse 3.4 + all other plugins that are needed to run EclipseUML
- **Plugin build :** EclipseUML should be installed as a plugin on an existing Eclipse 3.4
- **EclipseUML for Java:** EclipseUML is only including Europa for Java developer features
- **EclipseUML for JEE:** EclipseUML is including all Ganymede for Java EE developer features
- **Jar file:** It is an executable file which could be launched using java. It is used for XP, Vista, Linux and Mac users
- **Zip file:** It is a zip file for XP and Vista

3. How to Install the Standalone JEE build Jar file:

You need to download the EclipseUML Studio JEE standalone build from Omondo Website.

Double click on the jar file to launch it using Java (*To launch the jar file from windows: Open the Control Panel > Folder Option > File Types > select Jar > Change > Open with Java*)

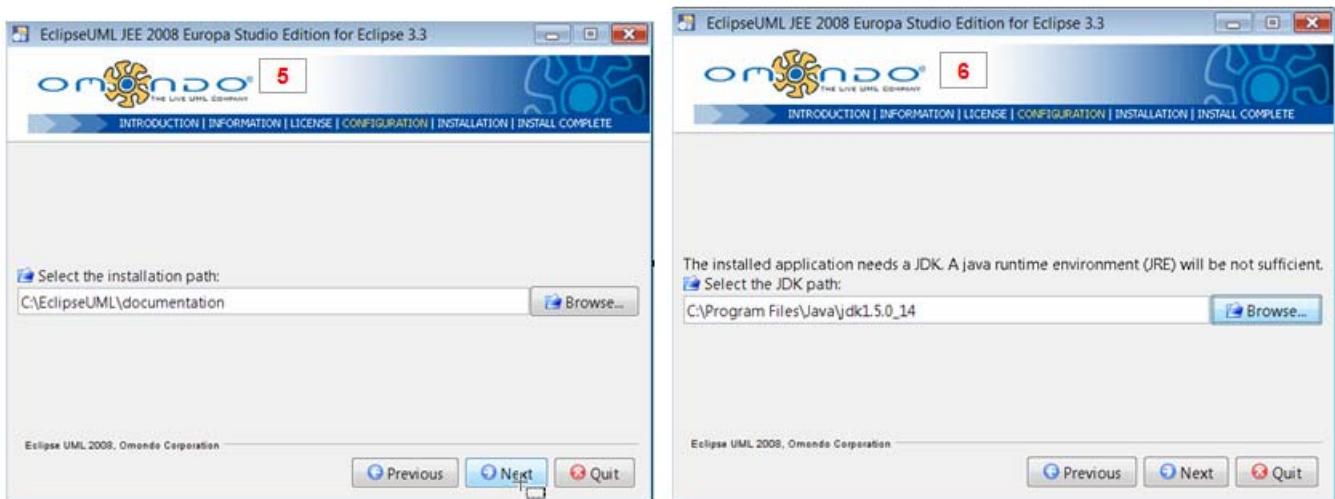
1. Select your language and click on the Ok button
2. Click on the Next button



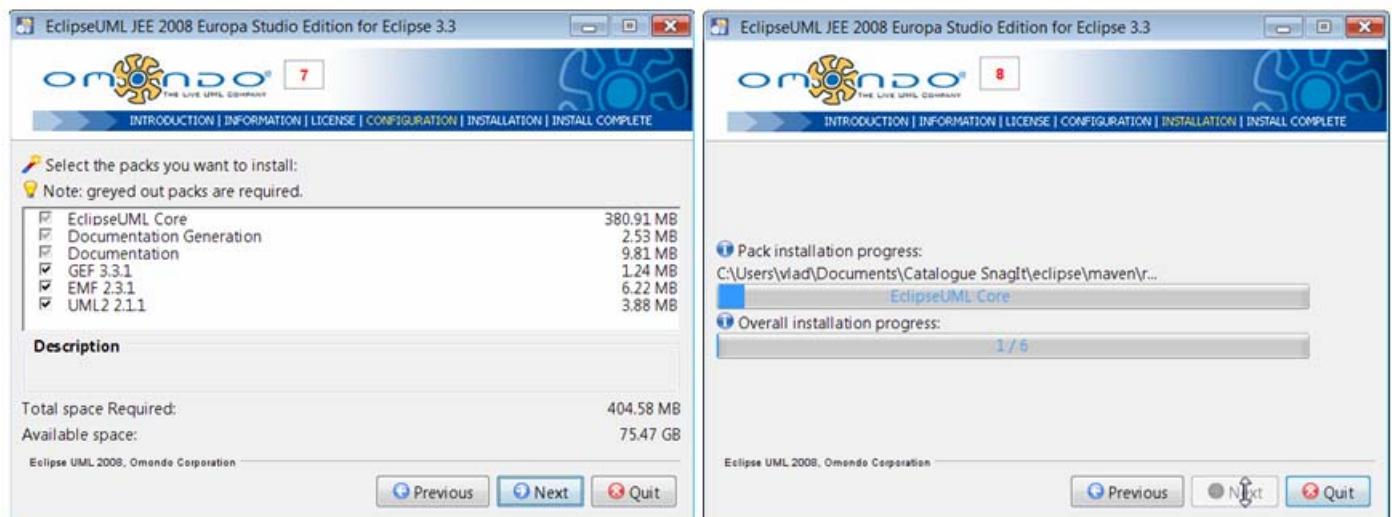
3. Read the following information and click on next
4. Select the Accept the terms of this agreement field and click on the next button



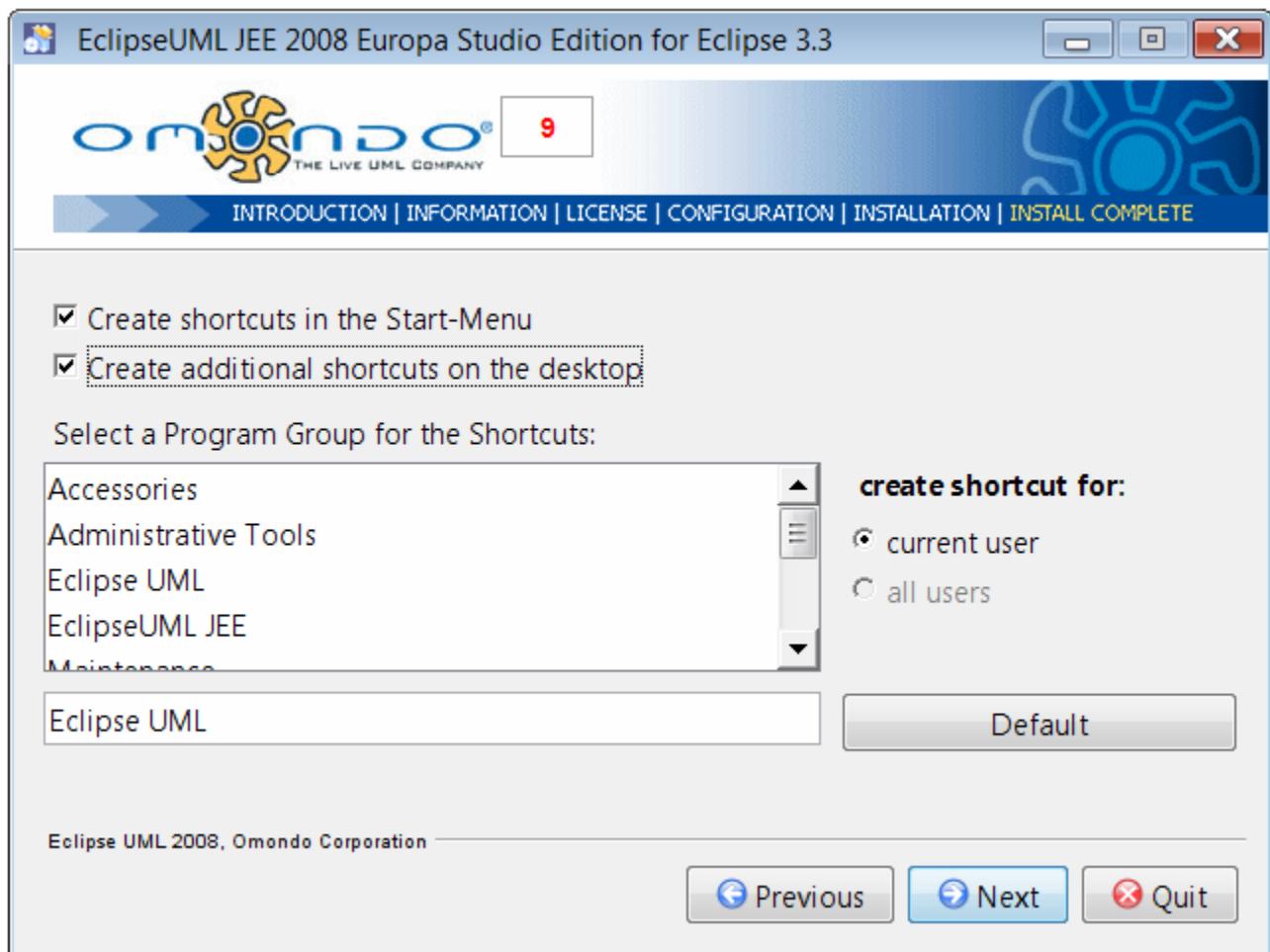
5. Create the new folder in which you will install EclipseUML
6. Select the JDK path which will only be used during the EclipseUML session (*You don't have to change your current Java configuration but only to use this JDK with EclipseUML*)



7. Select GEF, EMF and UML and click on the next button
8. EclipseUML is being installed (about 5 minutes) and click on the next button



9. Select Create a shortcuts or additional shortcuts field and click on the next button



10. The installation has completed successfully



4. How to Install the EclipseUML plugin Jar file:

You need to download the EclipseUML Installer from Omondo website or an accredited partner. For example, let's download the following beta release:

164,719 Ko Executable Jar File

 eclipseUML_E330_2007_studioEdition_3.3.0.v20070621.beta.jar

6/2

Double click on the Jar file.

If it doesn't work, the two main reasons are:

- *the jar file is corrupted and you need to download it again*
- *the java.exe is not the associated program to .jar extension in your Windows file configuration*

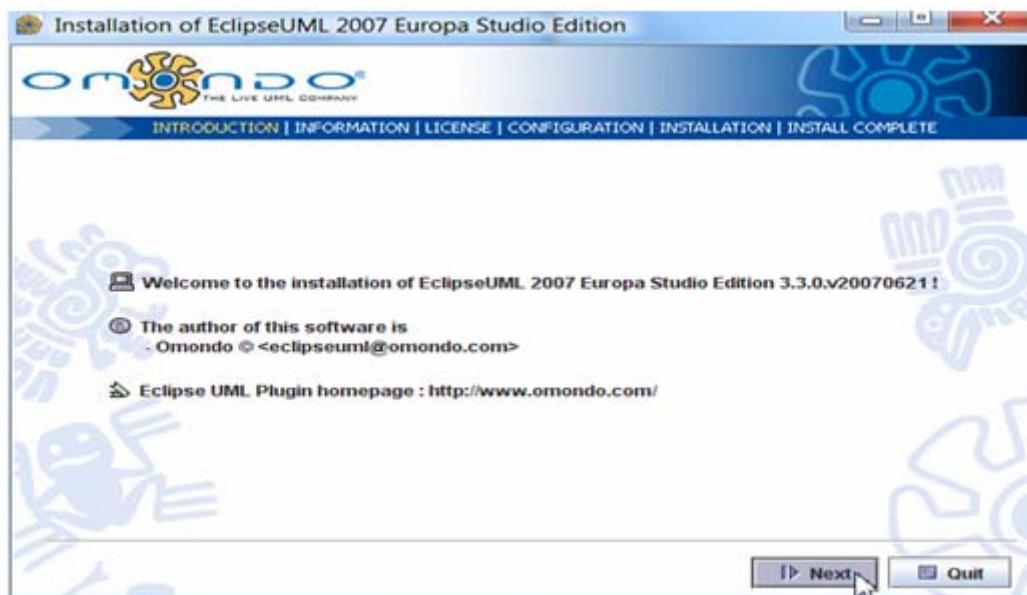
To launch the jar file from Windows:

Open the Control Panel > Folder Option > File Types > Select Jar > Open with Java

Select your language and click on the Ok button



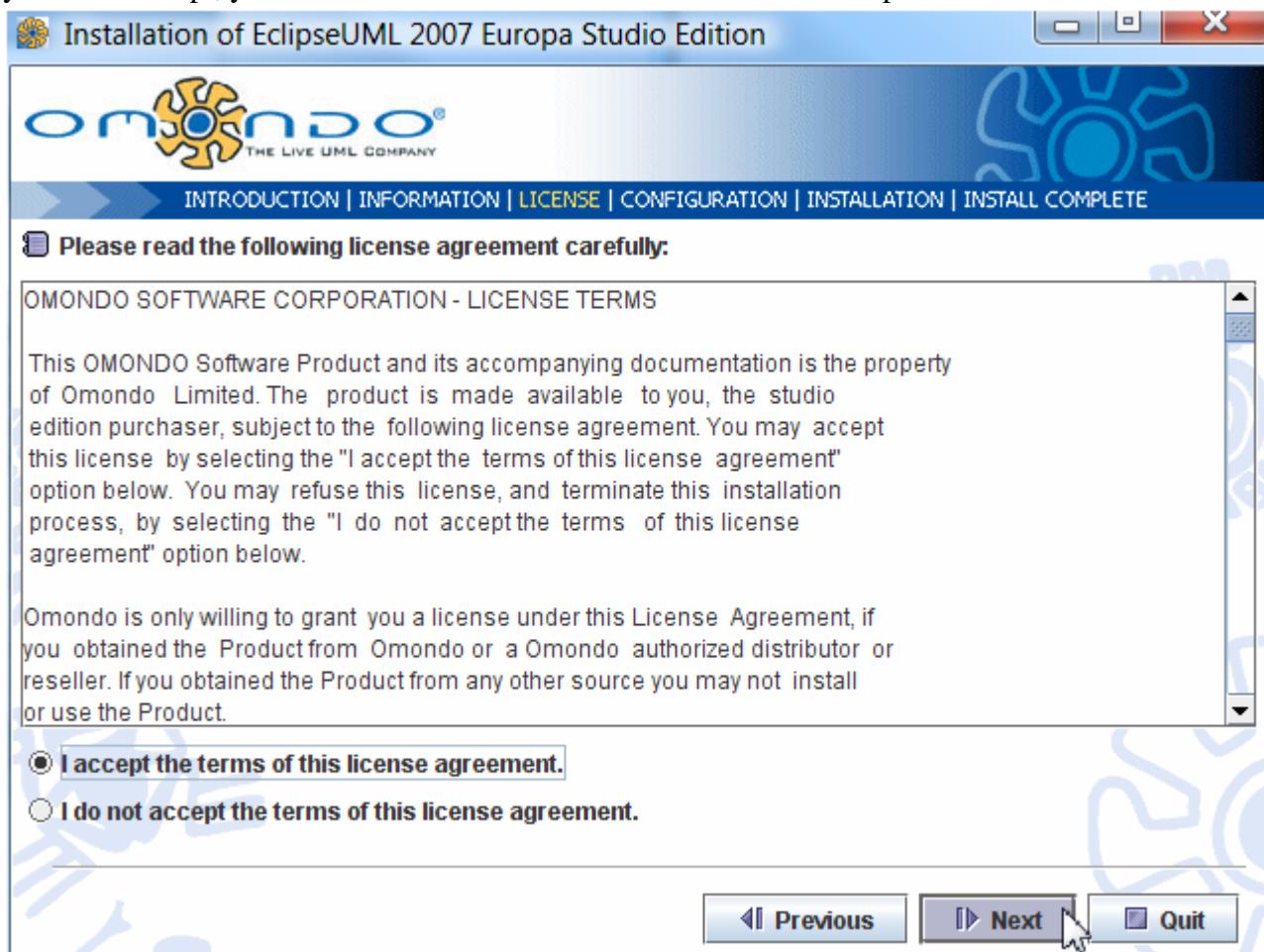
Click on the Next button



Click on the Next button

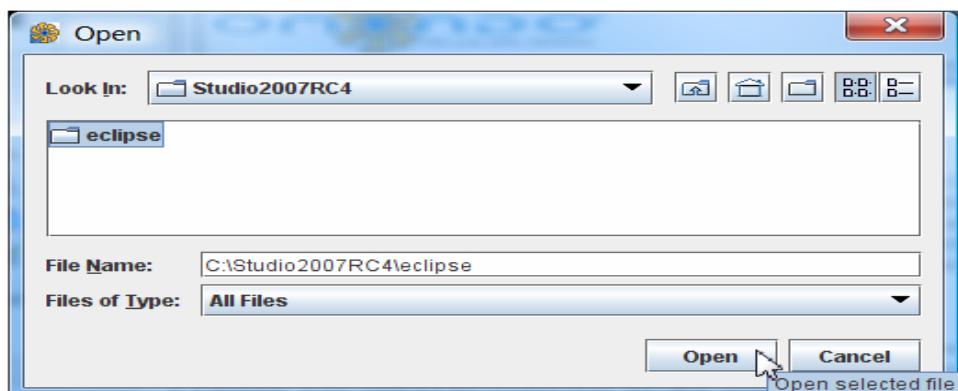
A screenshot of the EclipseUML 2007 Europa Studio Edition installation window. The title bar reads "Installation of EclipseUML 2007 Europa Studio Edition". The main area displays a section titled "Please read the following information :". It contains a blue header box with the text "About EclipseUML 2007 Europa Studio Edition for Eclipse 3.3RC4" and the date "Juin 2007". Below this, a red "About" heading is followed by a paragraph of text: "Our company was founded by Java developers for Java developers willing to spend more time on understanding and analysing business needs than on just coding. Omondo UML Plugin for Eclipse is a plugin for UML 2.1 modeling (UML2.1 notation & Metamodel XMI 2.x) in Eclipse 3.3. It can be used both for creating new models and for reverse engineering existing models. This is a beta release of EclipseUML 2007 Europa Studio Edition 3.3RC4 including :". A bulleted list follows: "● Class diagram (live synchronization between the class diagram and the java code)." At the bottom right are "Previous", "Next", and "Quit" buttons, with "Next" being the one currently highlighted.

You should accept the terms of this license agreement.
If you don't accept, you cannot click on the Next button and install EclipseUML.

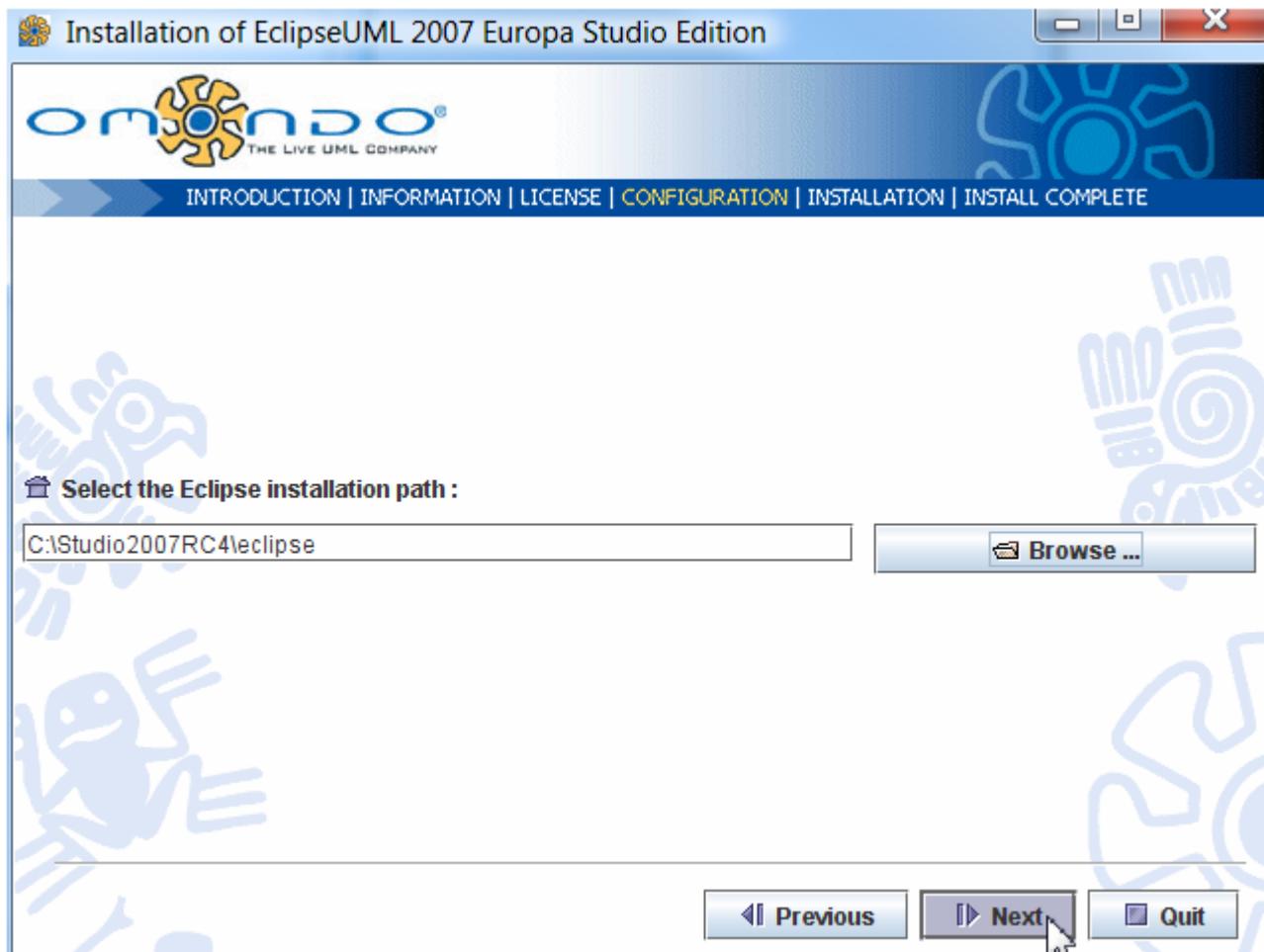


Select your Eclipse folder which contains your Eclipse 3.3 installation. You should have the following files in your folder:

.eclipseproduct	6/8/2007 8:11 PM
eclipse.exe	6/8/2007 8:11 PM
eclipse.ini	6/8/2007 8:11 PM
eclipsec.exe	6/8/2007 8:11 PM
epl-v10.html	6/8/2007 8:11 PM
notice.html	6/8/2007 8:11 PM
plugins	6/22/2007 9:21 AM



Click on the Next button after validating that you are going to install EclipseUML in the right hard disk path.

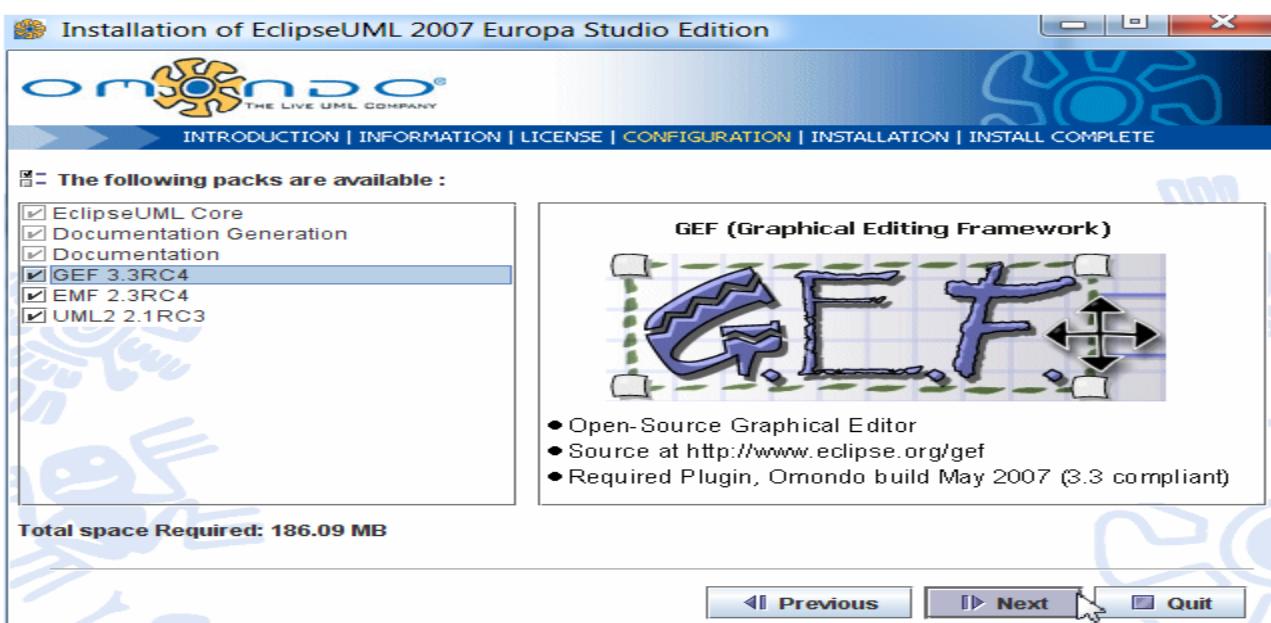


Click on the Next button.

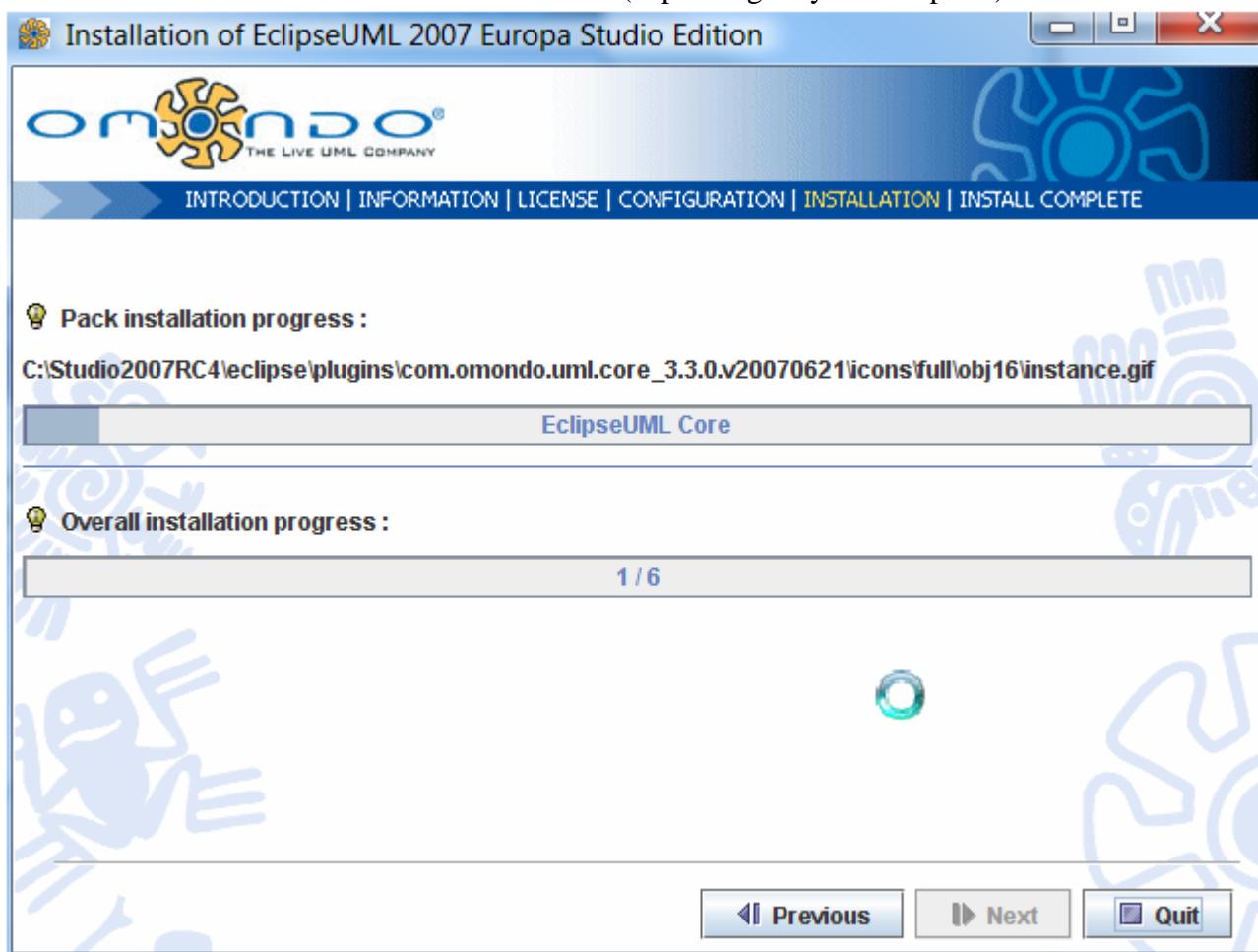
Please note that in order to avoid software instability it is no longer possible to unselect options.
Our GEF, EMF and UML 2 are always related to the lastest Eclipse 3.3 release available.

Each build has new GEF, EMF and UML 2 build configurations.

If you need more information about compatibility, you should go to <http://www.eclipse.org/projects/>



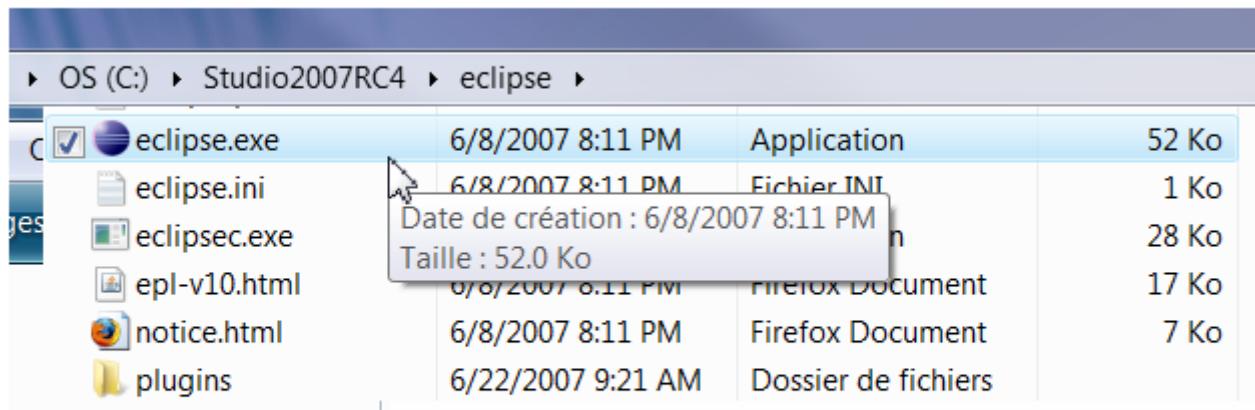
The installation will take between 2 and 7 minutes (depending on your computer)



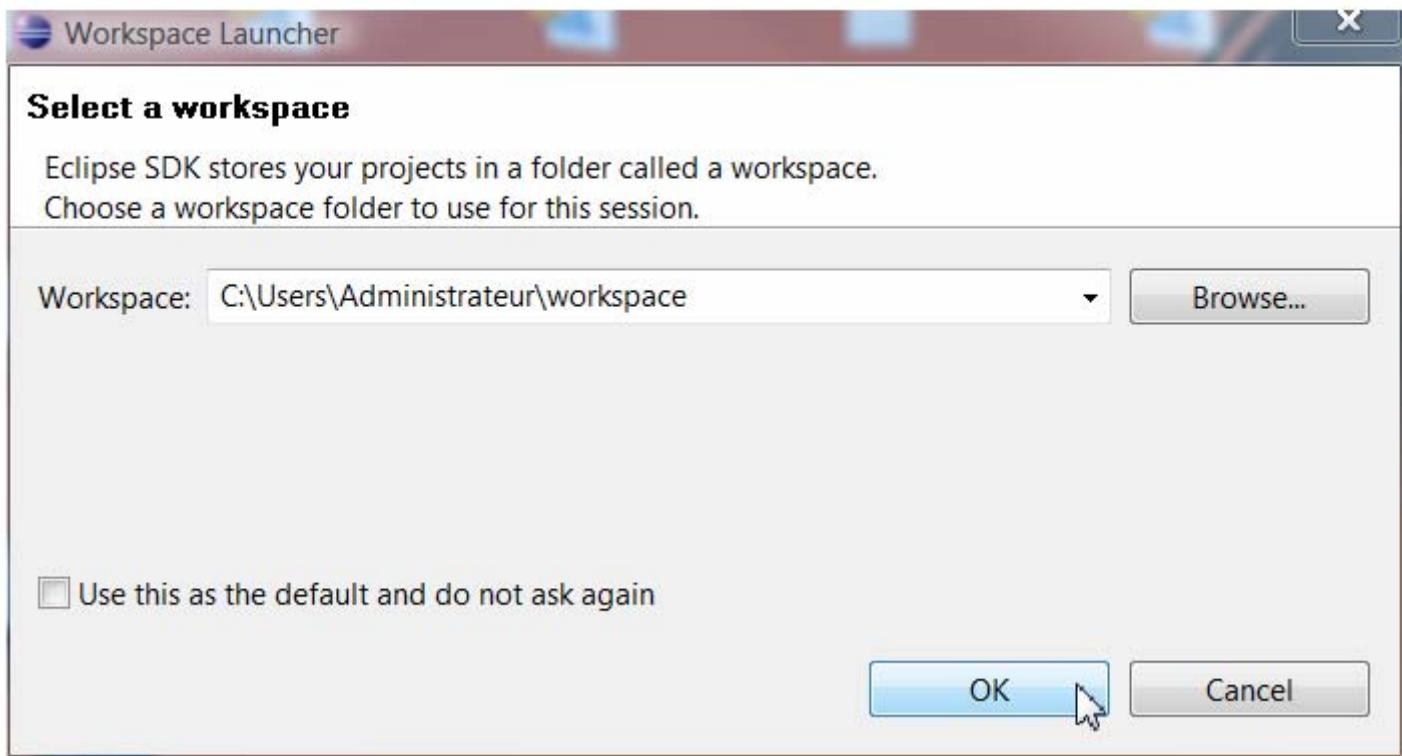
If the installation has been successfully completed, then the last wizard will appear.
Click on the Quit button to leave the installer.



To launch EclipseUML, you need to manually launch Eclipse from your hard disk.
Double Click on the Eclipse Icon



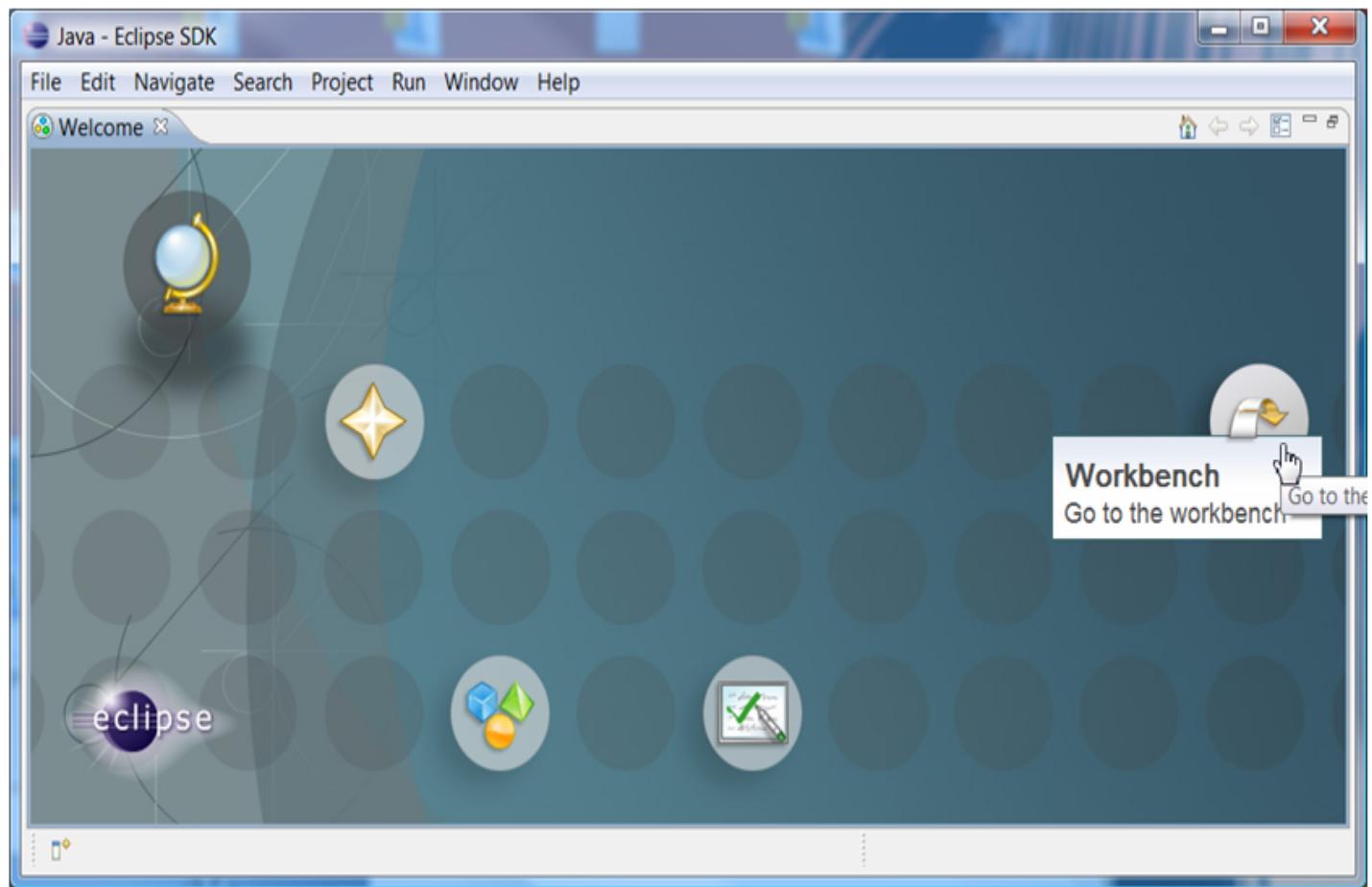
You need to select your Eclipse workspace.
Please note that EclipseUML will use the same workspace as Eclipse.



If EclipseUML is correctly installed, then the EclipseUML Europa splash screen will appear and activate the plugin.



Launch the Workbench to enter inside Eclipse 3.4



5. How to Install the Zip file:

You have two kinds of zip files:

1. EclipseUML zip file contains EclipseUML Studio Edition, EMF, GEF and UML2
2. Eclipse 3.4 + Eclipse UML zip file, which is a Stand Alone Eclipse zip file which contains Eclipse 3.3, EclipseUML Edition, EMF, GEF and UML2.

1. To install the EclipseUML zip file, you need to select your Eclipse workspace.

Select your Eclipse folder which contains your Eclipse 3.4 installation.

You should have these files in your folder:

 .eclipseproduct	6/8/2007 8:11 PM
 eclipse.exe	6/8/2007 8:11 PM
 eclipse.ini	6/8/2007 8:11 PM
 eclipsc.exe	6/8/2007 8:11 PM
 epl-v10.html	6/8/2007 8:11 PM
 notice.html	6/8/2007 8:11 PM
 plugins	6/22/2007 9:21 AM

3. To install Eclipse 3.4 + Eclipse UML, you just need to unzip it on your hard disk.

PS: This documentation has been updated on April 3rd, 2009.

It is valid for Eclipse Europa, Ganymede and Galileo.

The screen shots have been made with Eclipse Europa in 2007.

QuickStart with EclipseUML

This quickstart should help you to start using EclipseUML plugin for Eclipse Ganymede in less than 15 minutes.

1.EclipseUML contextual menus

1. [Create a new UML diagram](#)
 2. [Create a class inside your Class Diagram](#)
 3. [Diagram contextual menus](#)
-
2. [EclipseUML views](#)
 3. [How to reverse a Java code into a Class diagram](#)

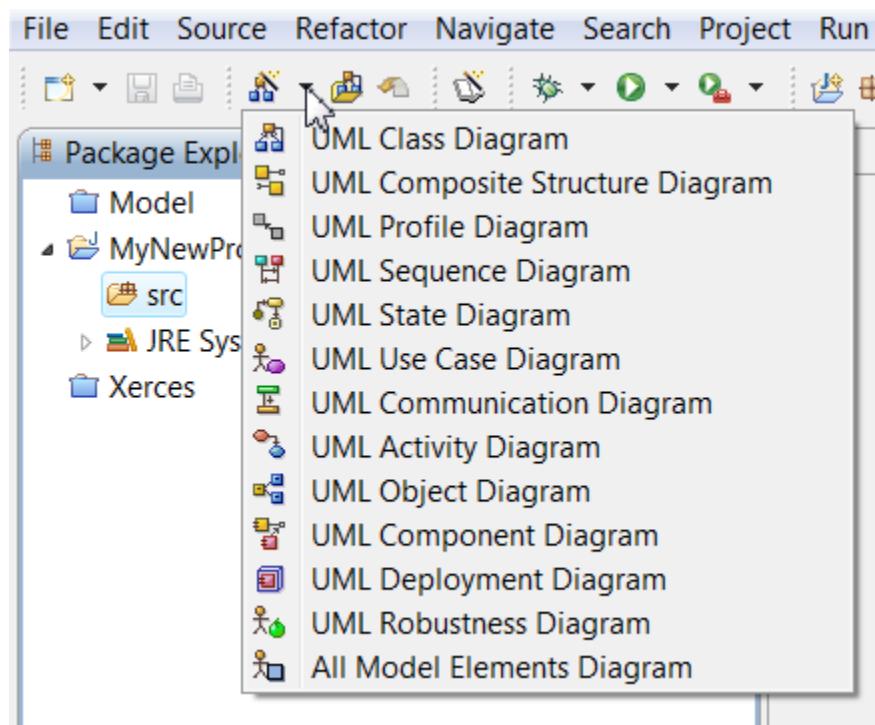
1. Where are EclipseUML Menus ?

1. Create a new UML Diagram

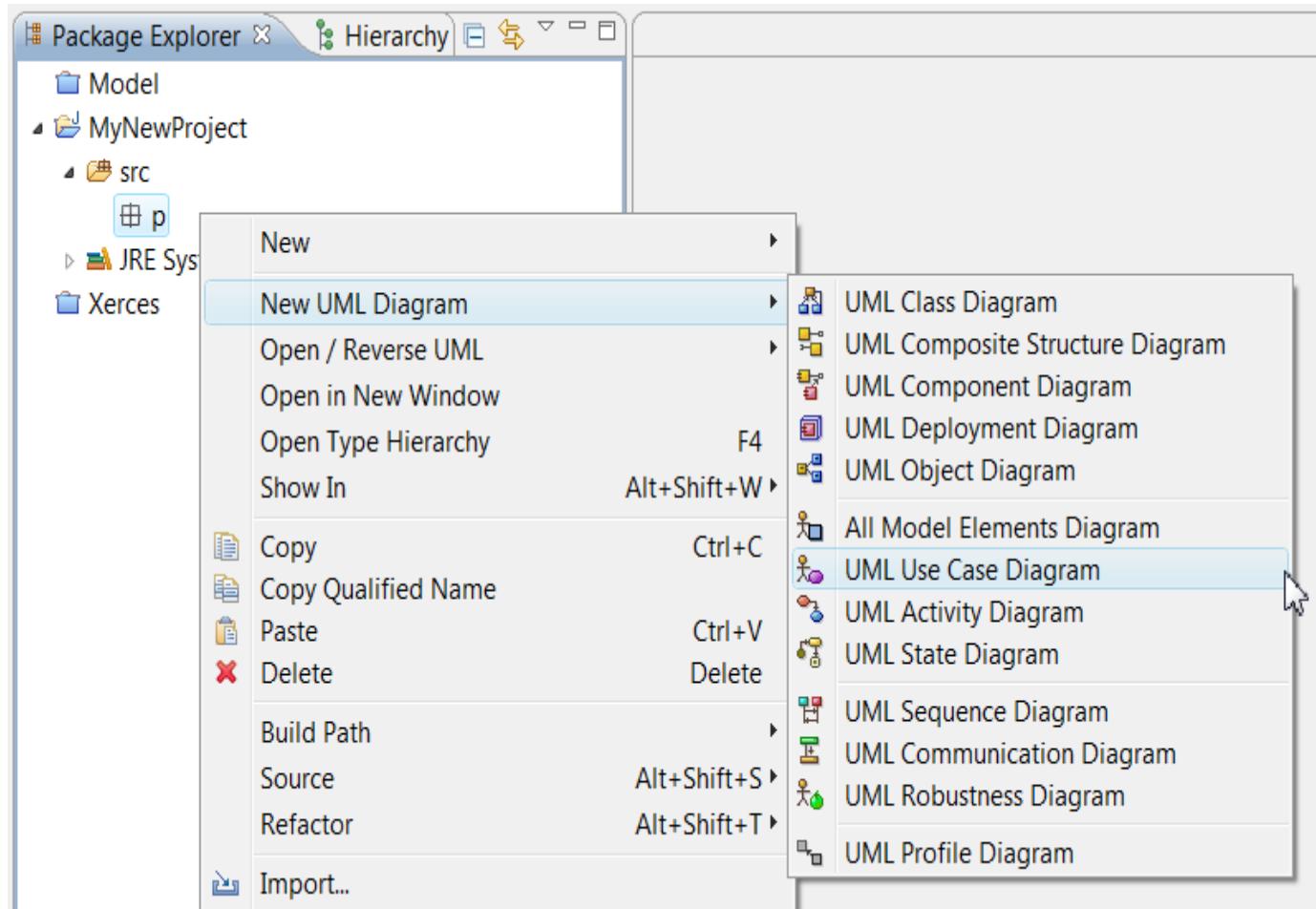
You first need to create a project inside your Package Explorer.

This project should have a Java or Jee nature.

You can create diagrams at the **src root using the top down arrow icon**



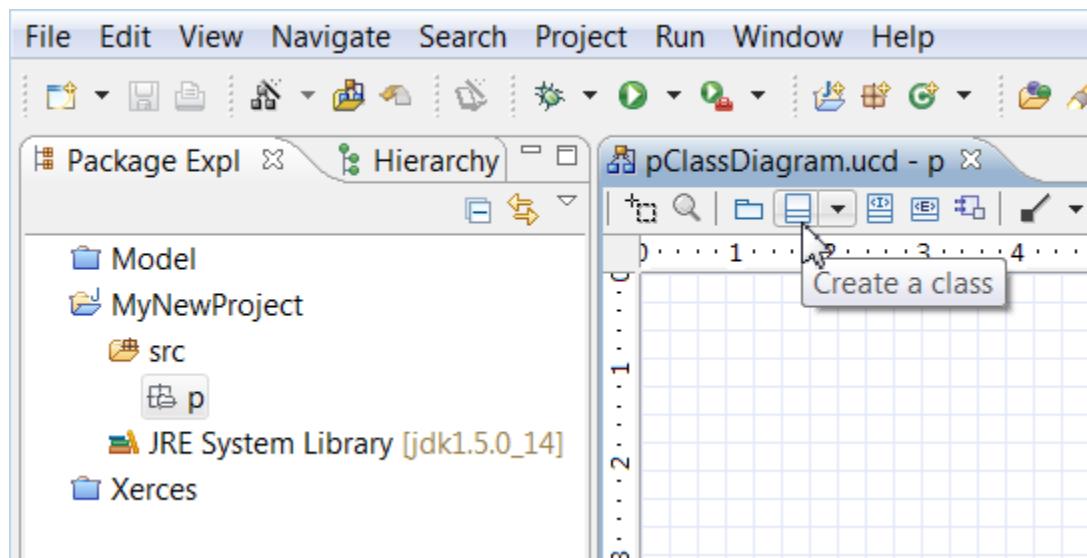
Or at the package root by selecting the **package > New UML Diagram >...**



Please note that if you don't create any package then your model elements will be created in a default package [inside the UML superstructure](#). It is therefore better to always give an appropriate name to each package.

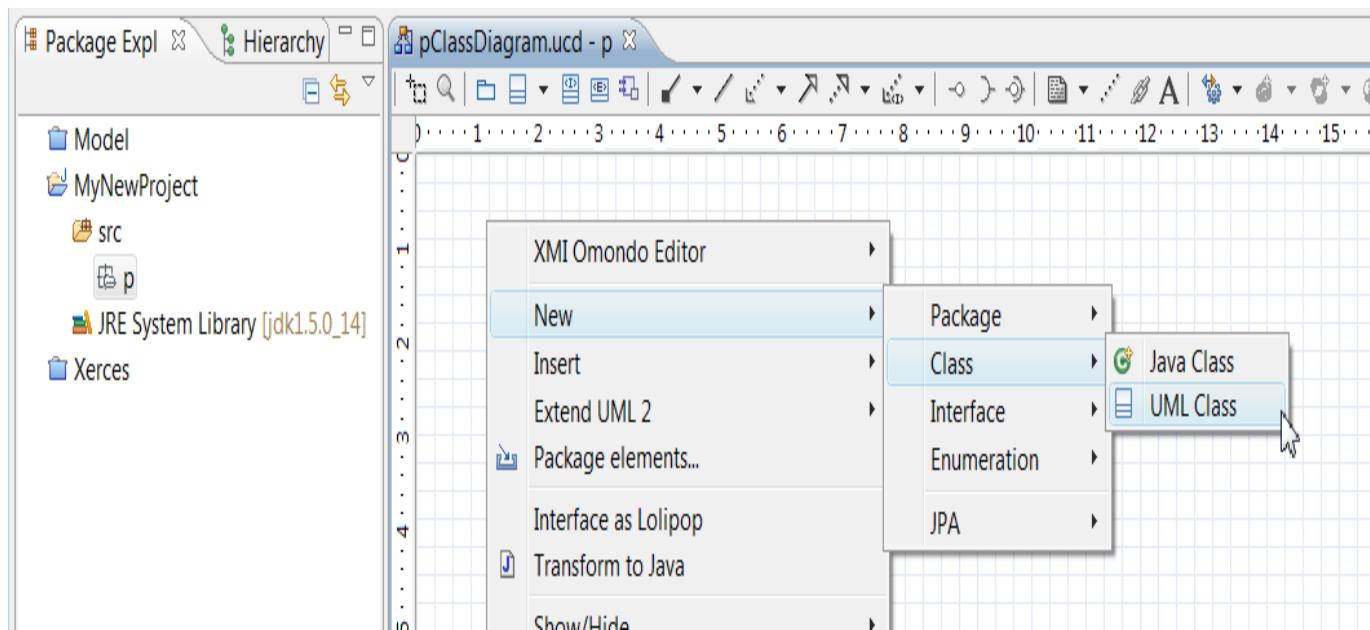
2. Create a Class inside your Class Diagram

You can create either **Java Classes** using the toolbar



or create UML classes (e.g. no java code is created at this stage) using the Class Diagram contextual menu.

The Class Diagram contextual menu can be selected by a left mouse click on the diagram background and a right click to open the menu.



3. Diagrams contextual menus

Each time you **click on anything inside the diagram**, you will always open a contextual menu. Using the mouse left button will allow you to select graphical model elements and the right click will help you to customize the graphical information and update the model.

You can click on:

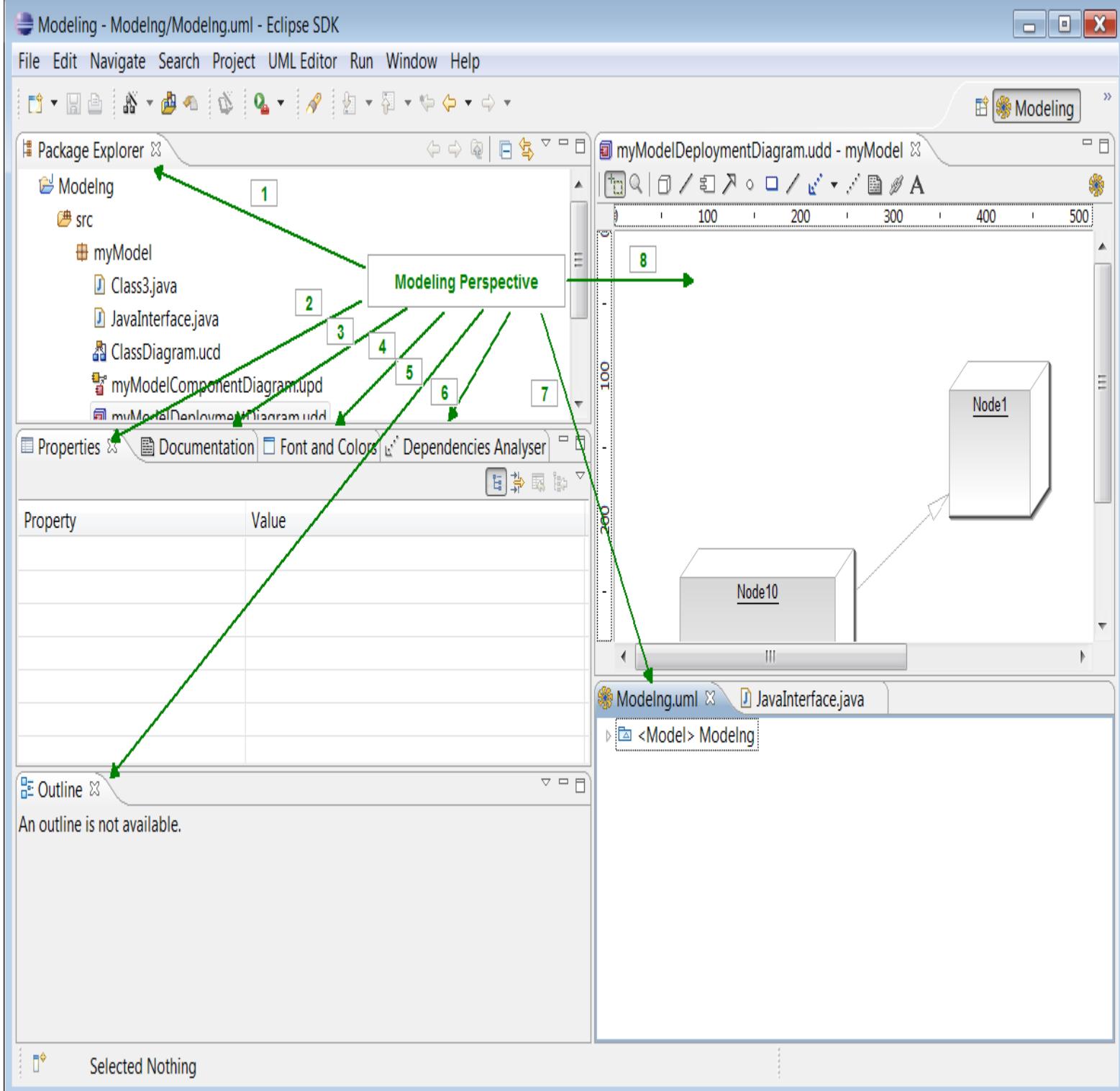
- [The diagram background](#) and open diagram contextual menu
- [Select an element](#) and open the element contextual menu
- [Select an attribute or method](#) and open the attribute/method contextual menu

2. EclipseUML views

You can use the [Omundo Modeling perspective](#) or use any of the Omundo views [inside another perspective](#) (e.g. Java, JPA, EJB, Web Services etc....)

You can model using any of the following views:

1. Package explorer
2. Properties View
3. Documentation View
4. Font and Colors View
5. Outline
6. Dependency Analyzer View
7. Model Editor
8. UML Editor

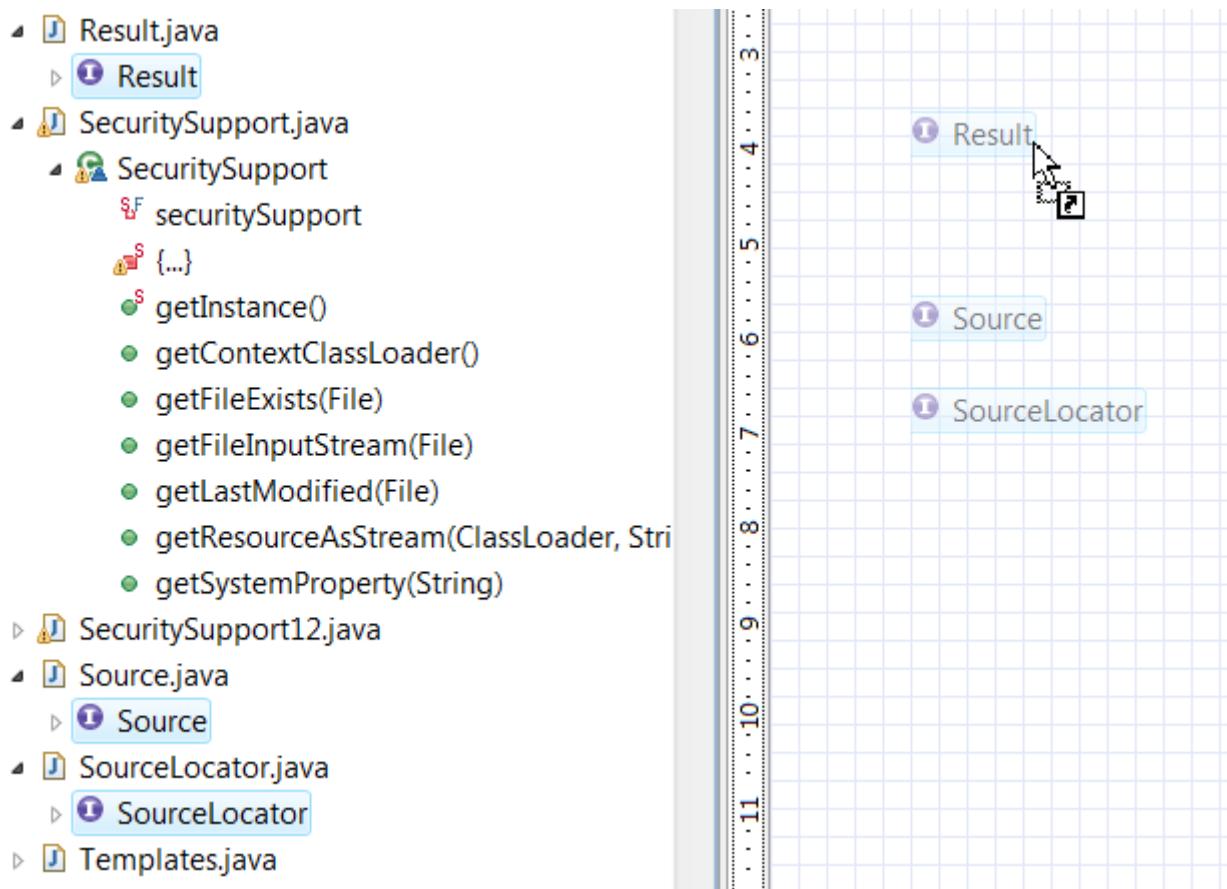


3. How to reverse a Java code into a Class diagram

The easiest way to reverse a Java code is to drag and drop a group of Classes and Interfaces from the Package Explorer to a Class Diagram. You can [create a class diagram at the root of the package](#) (e.g. the Class Diagram classifiers will be considered to belong [to this package](#)) or to give a name to the class diagram and [save it anywhere in the project](#).

Select a group of classifiers with the mouse in the Package Explorer and drop them inside the diagram.

Note that you can mix Classes and Interfaces coming from different packages.



For advanced reverse engineering need please read the [reverse engineering product documentation](#) or [the reverse engineering example of the Xerces project](#).

How to start:

- How to install EclipseUML (*5min*): [Flash demo](#) / [.exe file](#)
- Dynamic Modeling Concept (*1min*) [Flash demo](#) / [.exe file](#)
- Class Diagram Creation (*40s*) : [Flash demo](#) / [.exe file](#)
- My first class diagram (*3min*): [Flash demo](#) / [.exe file](#)

Getting Started

This tutorial is designed to get you started on using the Eclipse UML plugin for Eclipse. The following areas are covered:

[Creating a First Diagram](#)

- [Class Diagram Creation](#)
- [Classes and Interfaces](#)
- [Attributes](#)
- [Methods](#)
- [Associations](#)
- [Implementation and Inheritance](#)

How to start:

- How to install EclipseUML (*5min*): [Flash demo](#) / [.exe file](#)
- Dynamic Modeling Concept (*1min*) [Flash demo](#) / [.exe file](#)
- Class Diagram Creation (*40s*) : [Flash demo](#) / [.exe file](#)
- My first class diagram (*3min*): [Flash demo](#) / [.exe file](#)

Class Diagram Creation

In this section, you will learn how to start using EclipseUML inside Eclipse and make a new class diagram.

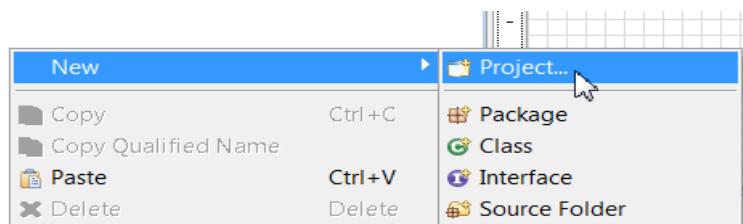
It is important to always create a src directory.

This section uses the following elements:

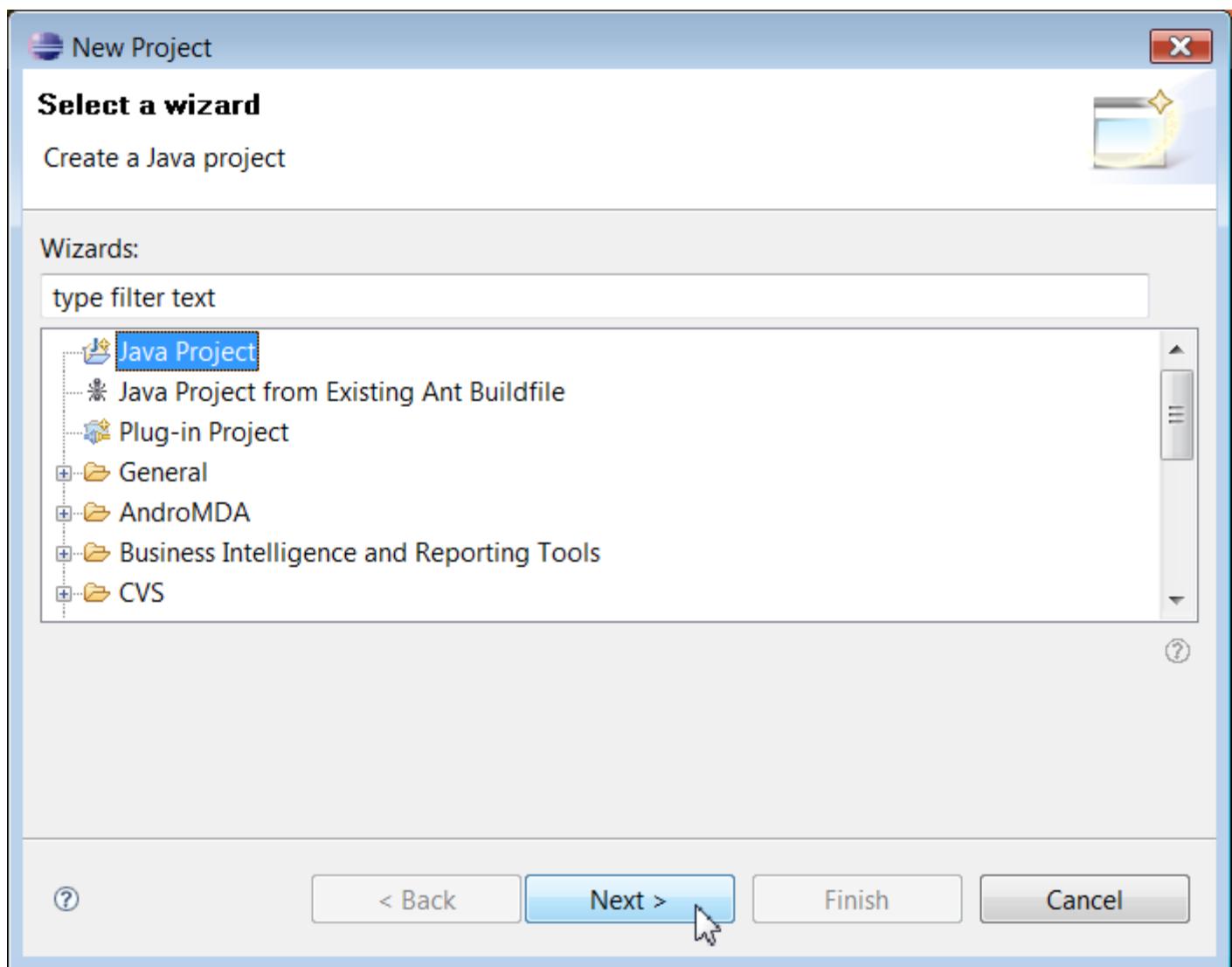
1. [Create a Java Project](#)
2. [Create a Package](#)
3. [Create a Class Diagram](#)

1. Create a Java Project

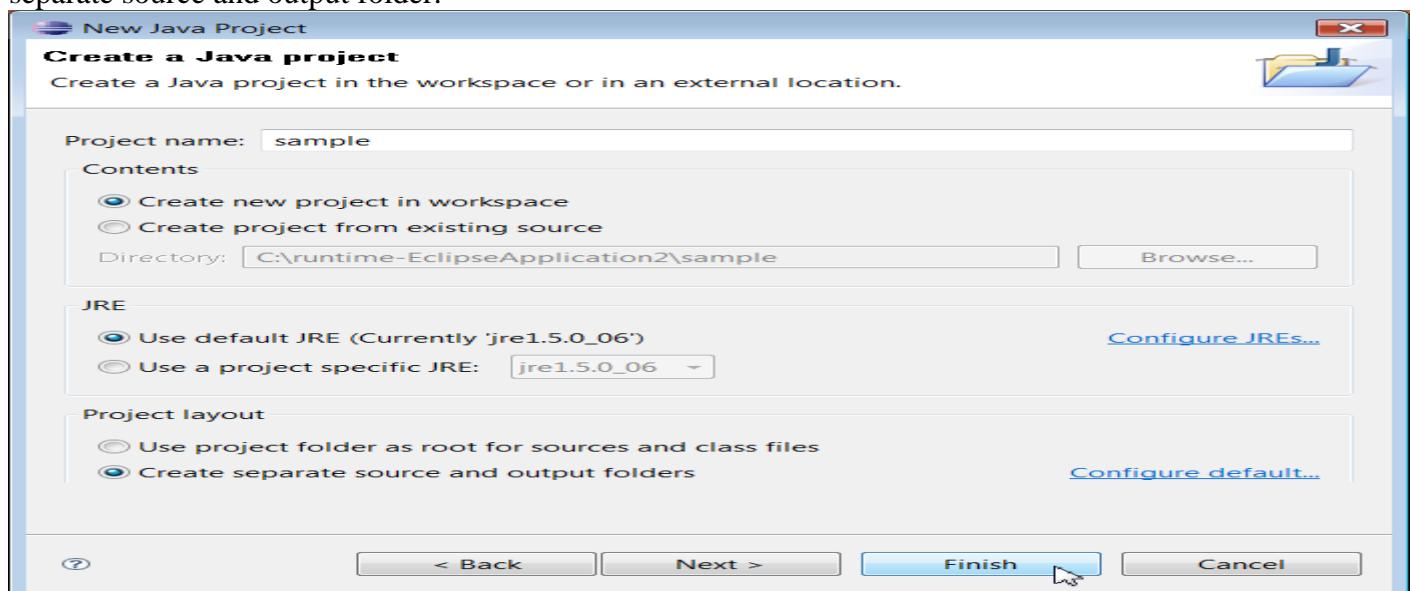
When starting your class diagram creation, you should create a Java Project.
Right click in the Package Explorer select **New > Project**



Select **Java Project** and click on the Next button.

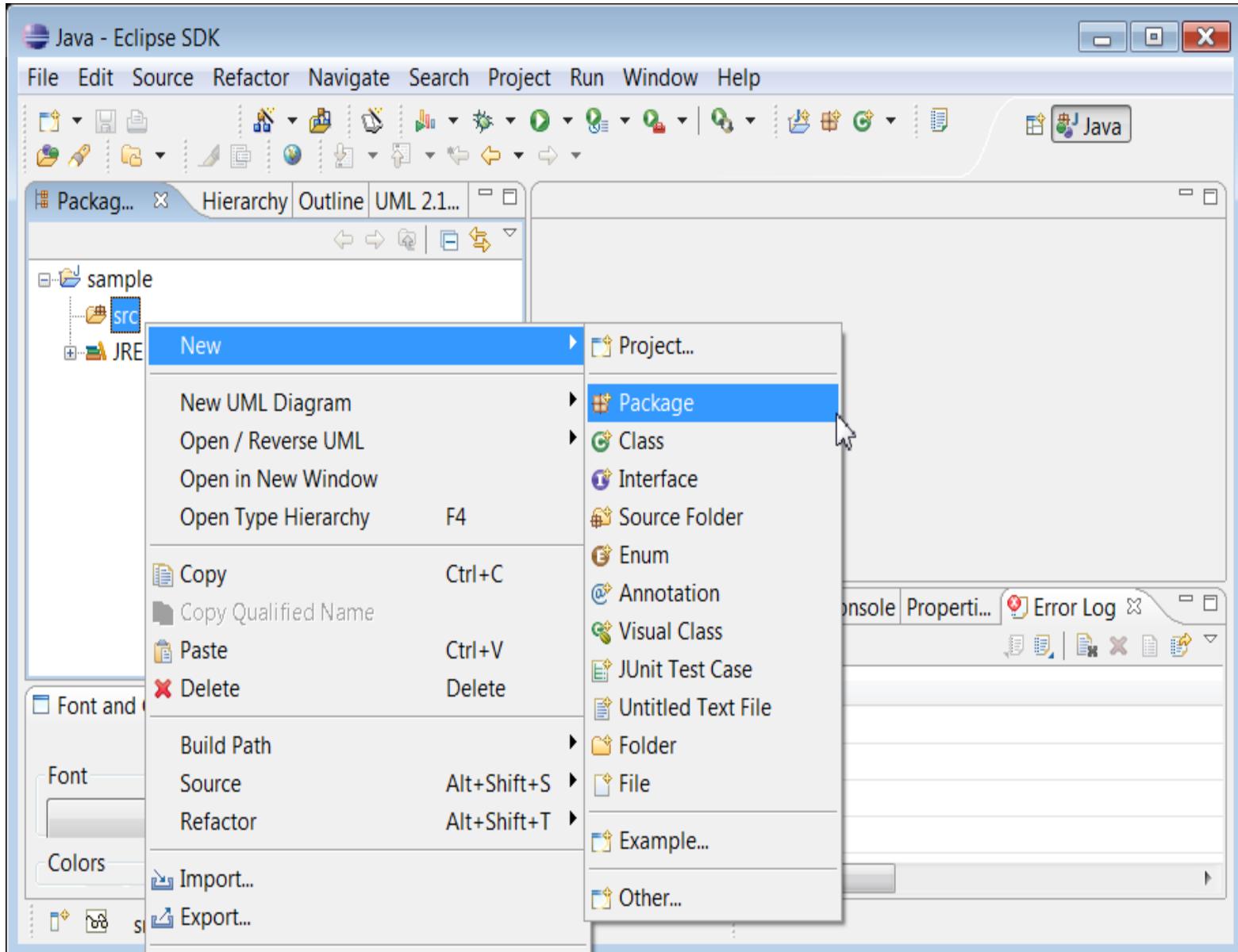


Enter the name of the project in the Project name field. Select your workspace location and create separate source and output folder.

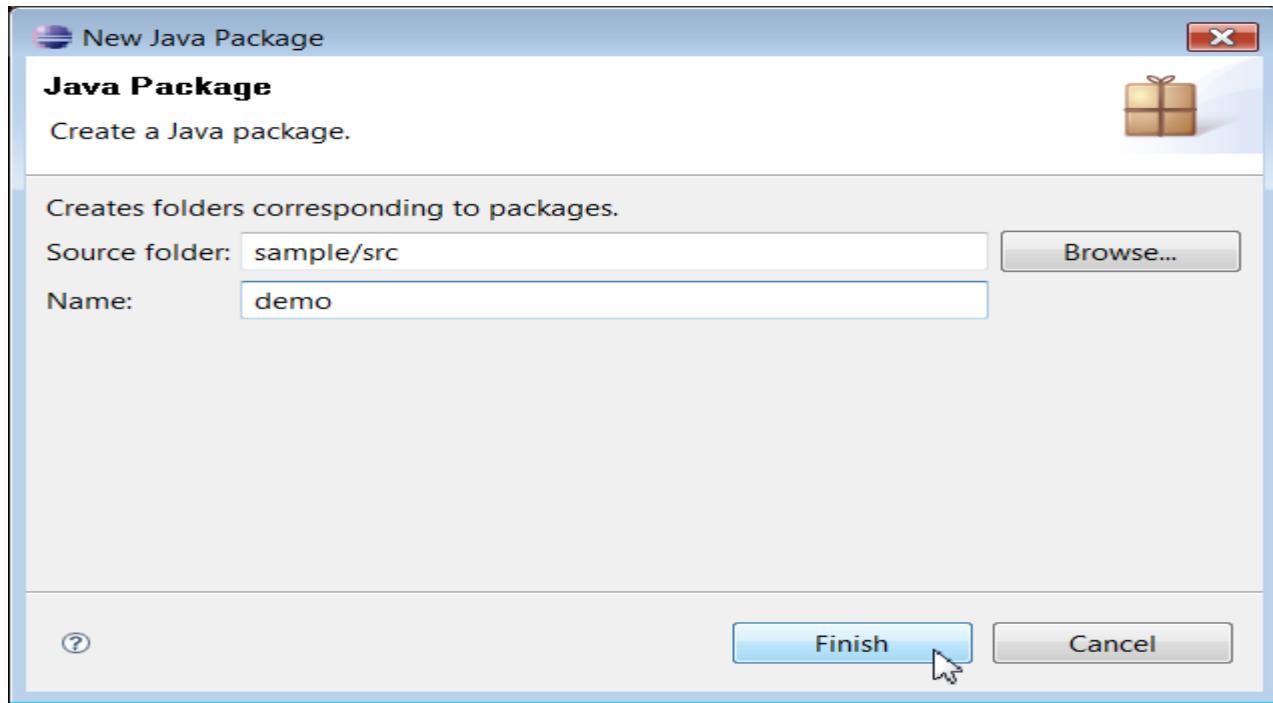


2. Create a new Package

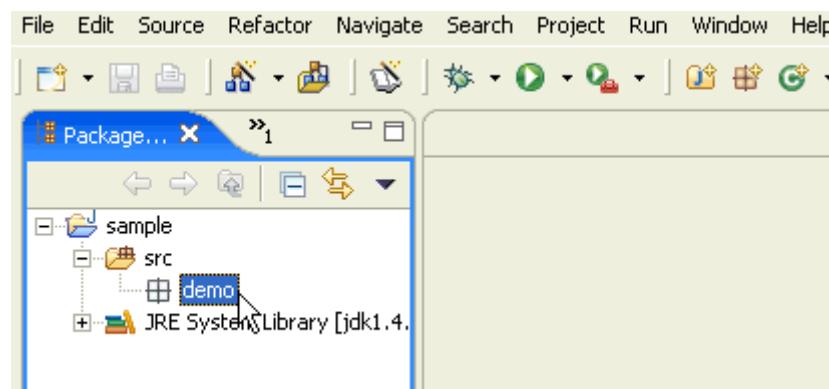
Select src in the Package Explorer and open the folder pop up menu **New > Package**



Enter the name of your package in the Name field and click on the Finish button.



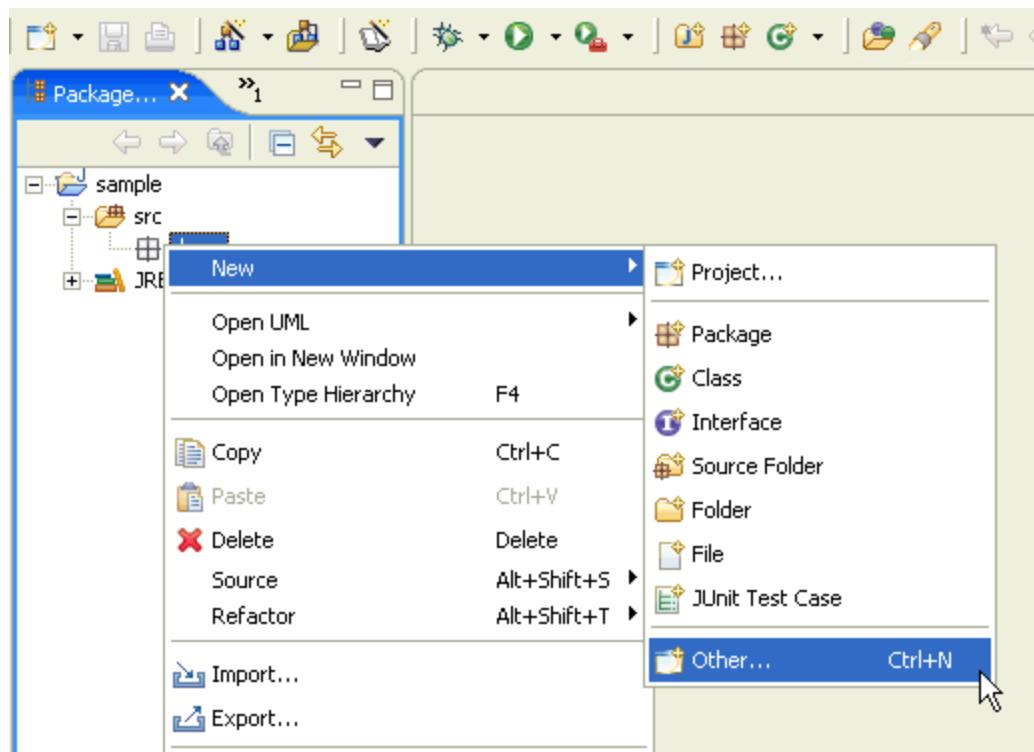
We have successfully created a project, a src folder and a package.



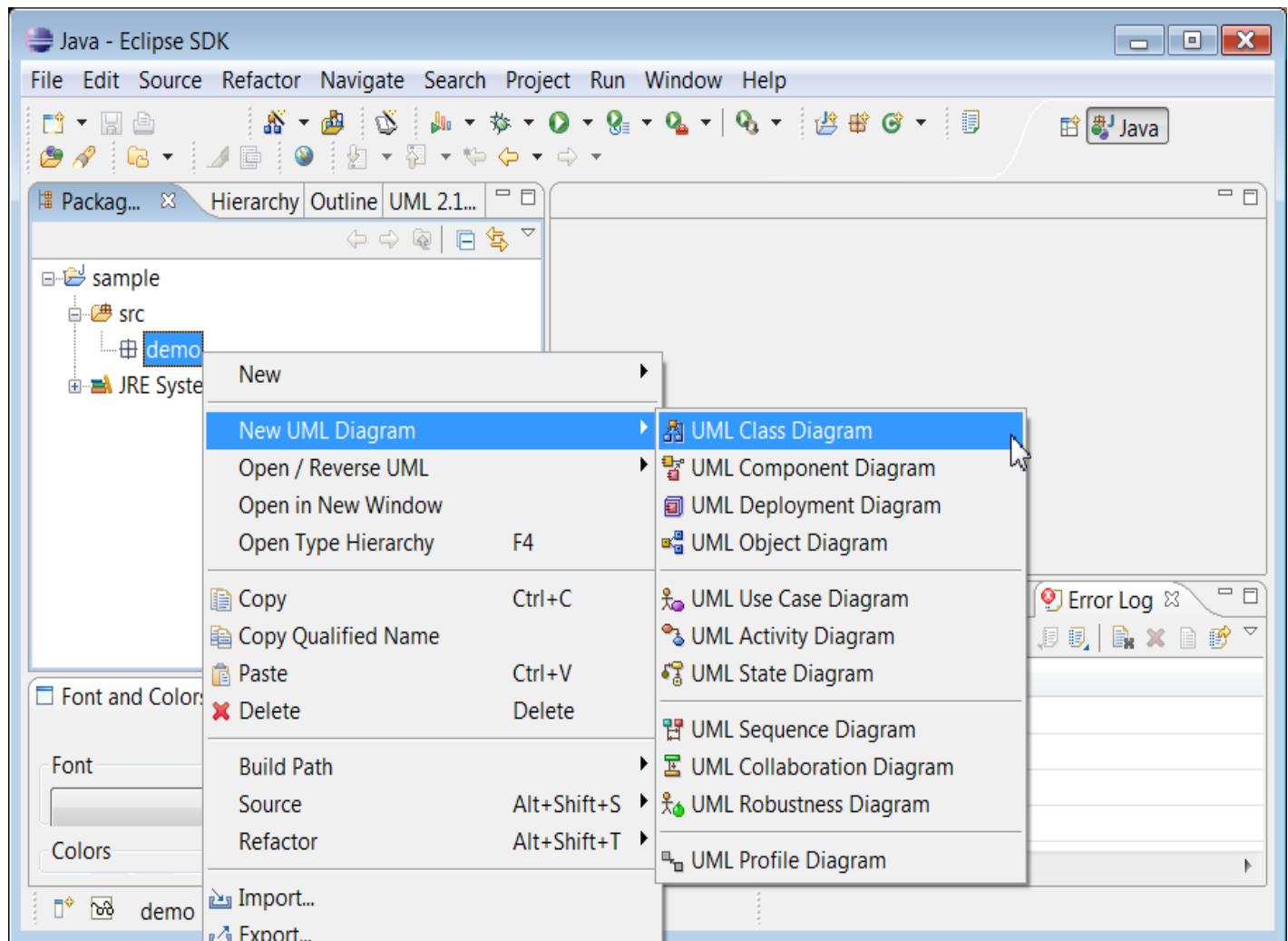
3. Create a Class Diagram

To create a new Class Diagram, select a Package in the Package Explorer, then open the Package pop up menu:

Select either **New > Others**.

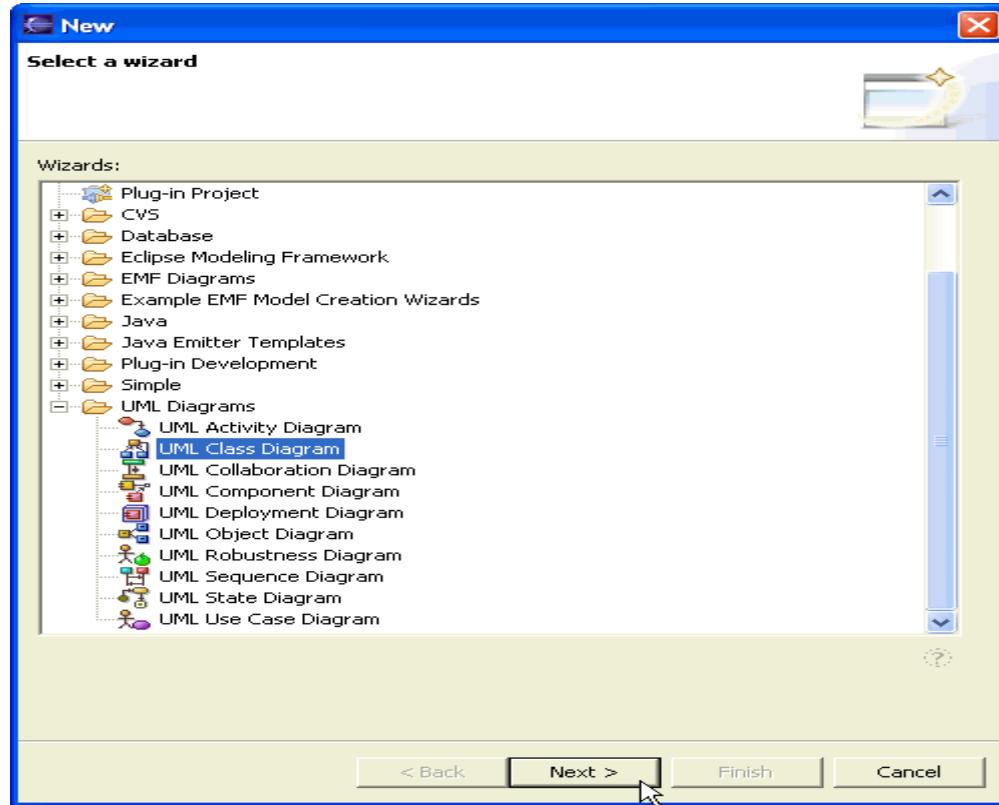


Or select **New UML Diagram > UML Class Diagram**.

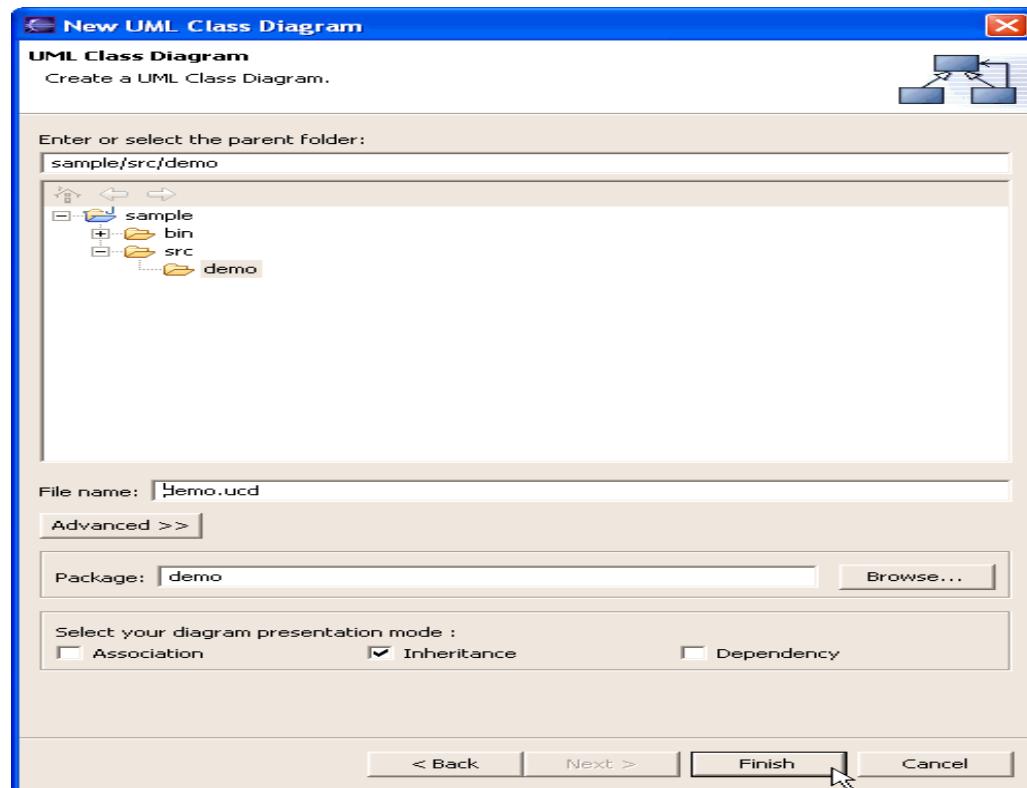


Select **Uml Diagrams > UML Class Diagram**.

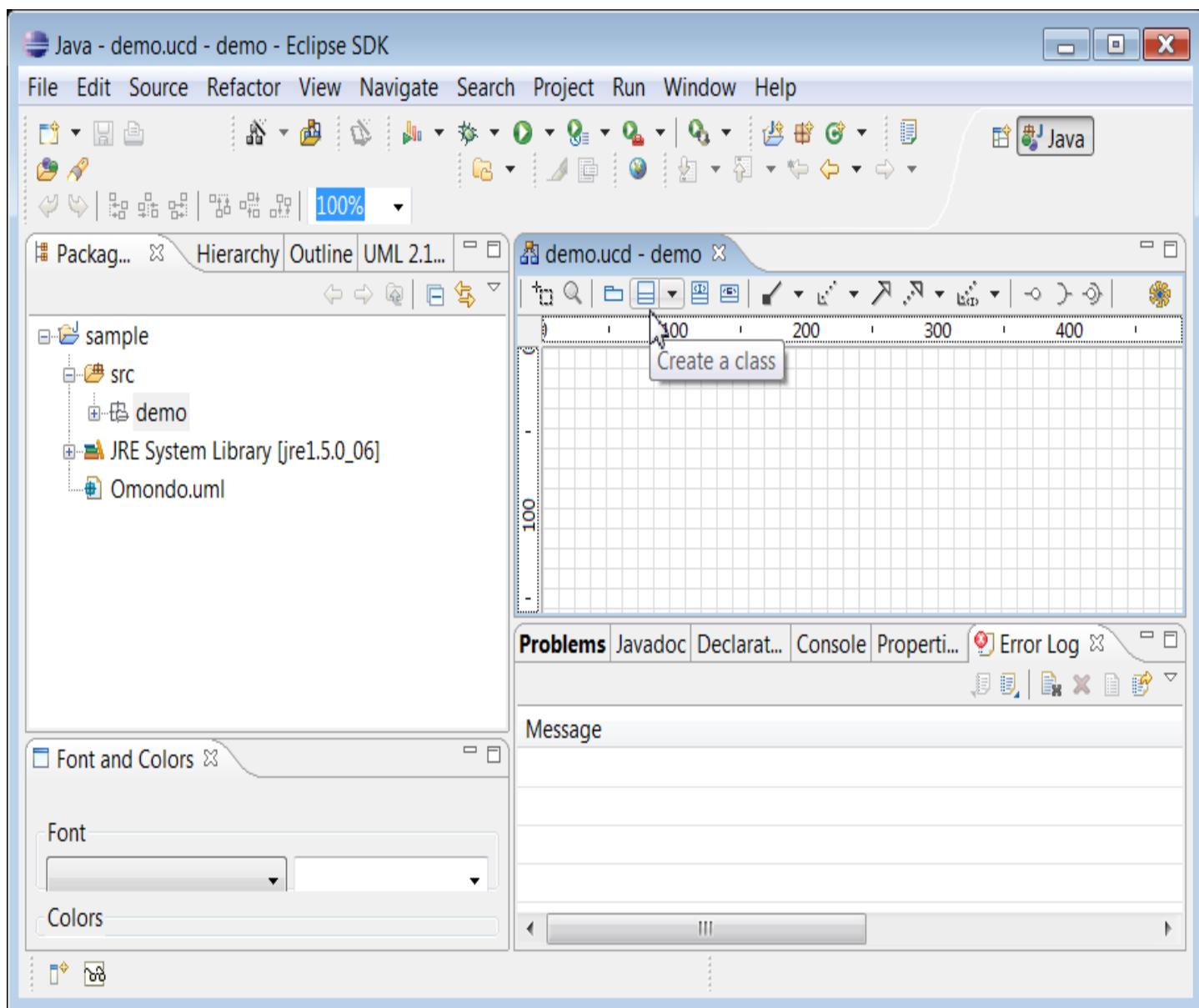
Click on the Next button.



Select the package you want to work with and enter the name of the Class Diagram in the File name field.



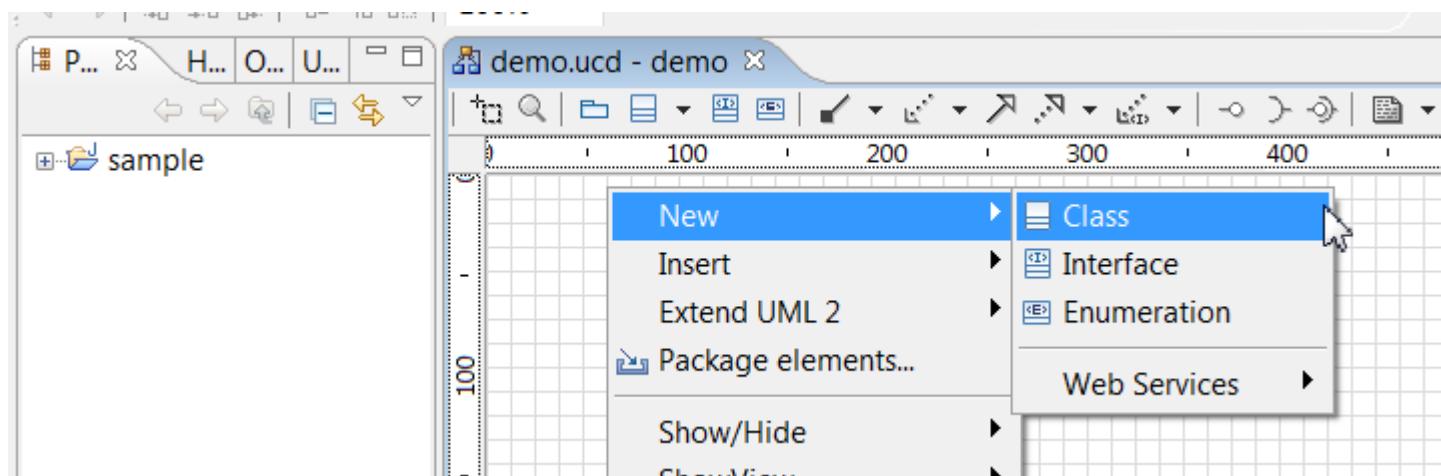
Your new class diagram editor has been created.



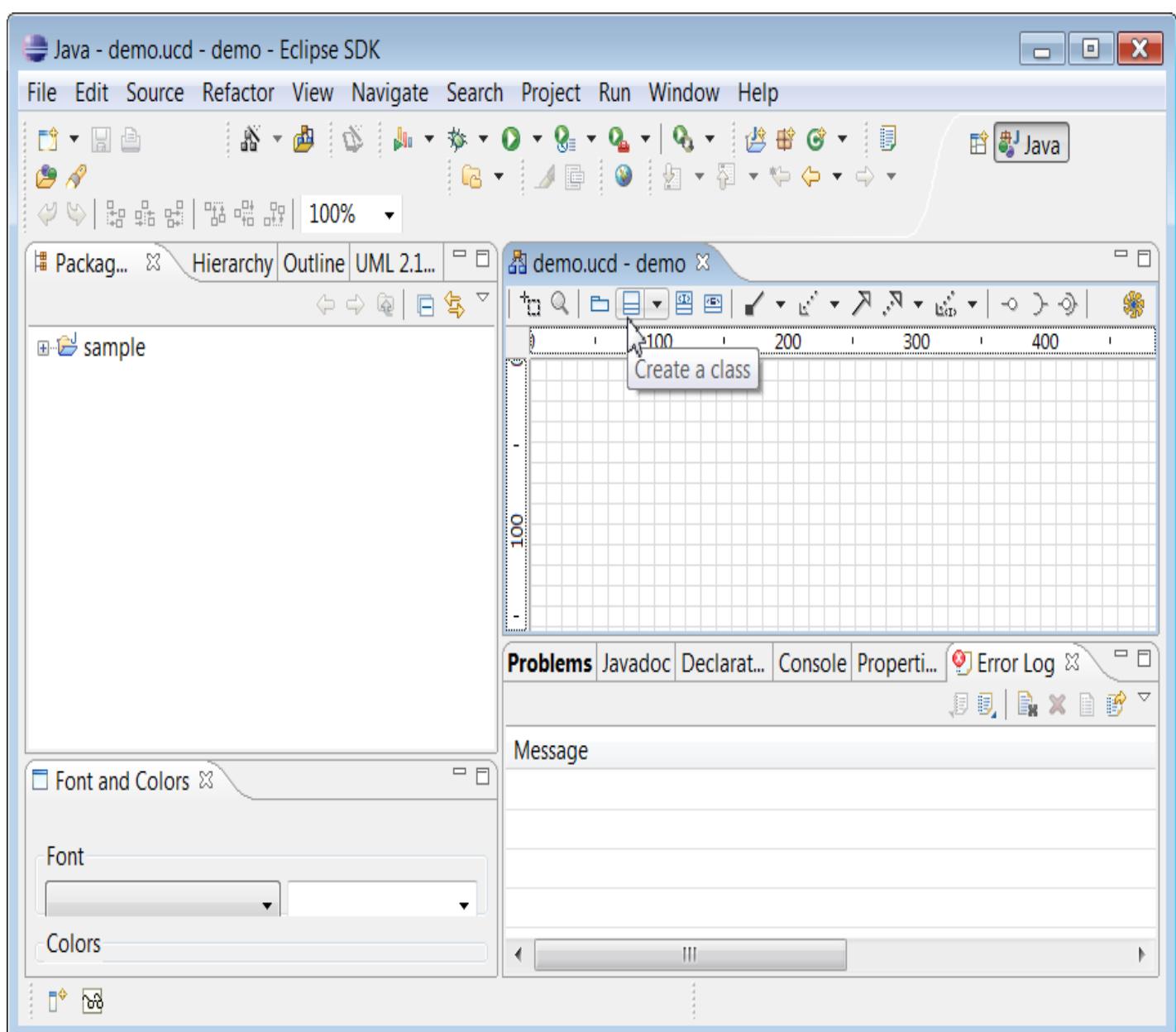
Classes and Interfaces

In this section, you will see how to insert new classes or interfaces from the class diagram graphical interface.

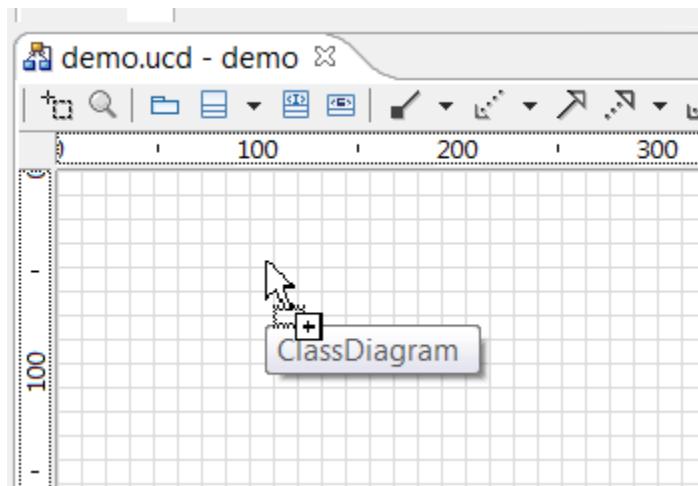
To create a new class (or interface) directly in your class diagram, select *New > Class* (or *New > Interface*) in the class diagram contextual menu.



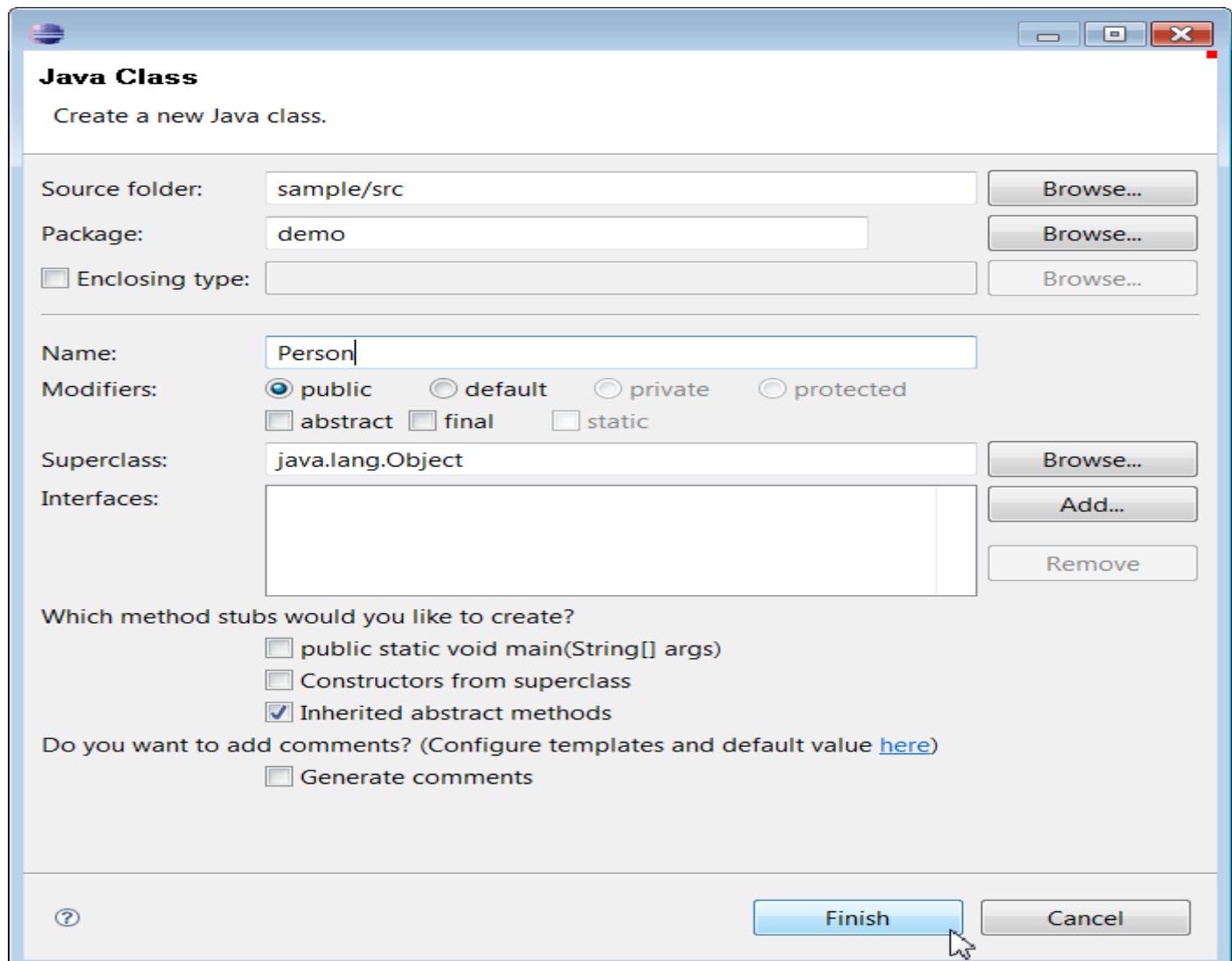
To create a new class is also possible by selecting the green icon in the tool bar (create a class diagram).



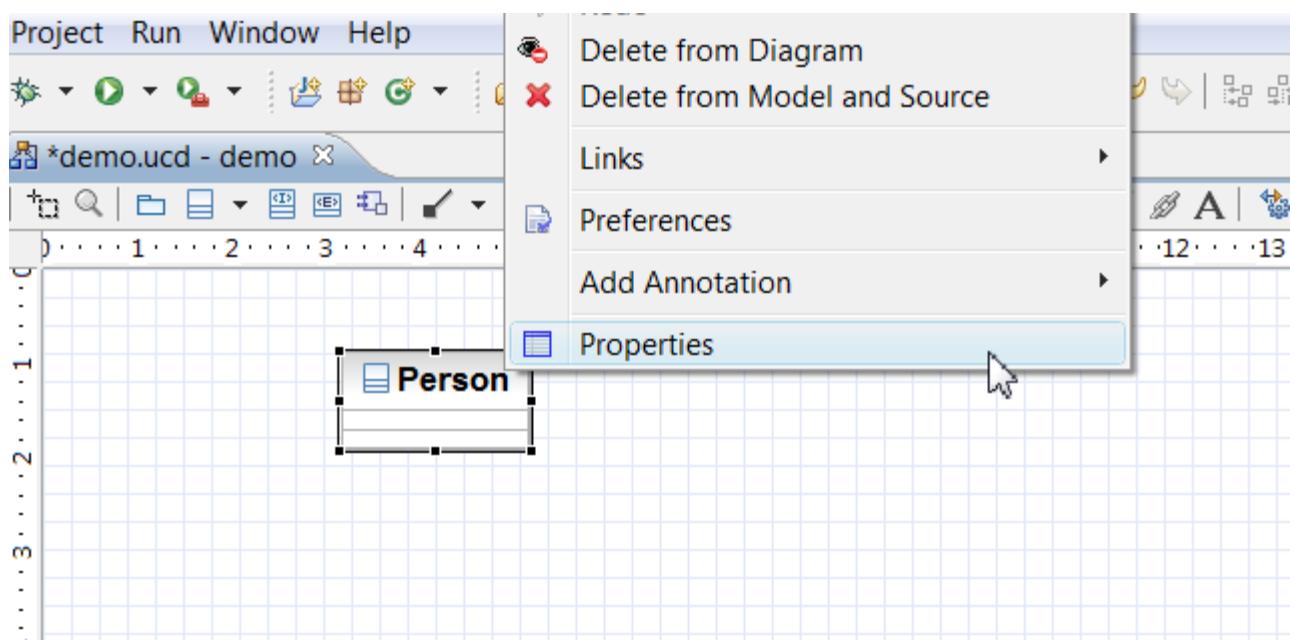
Drag and drop the icon inside your class diagram editor to create a new class.



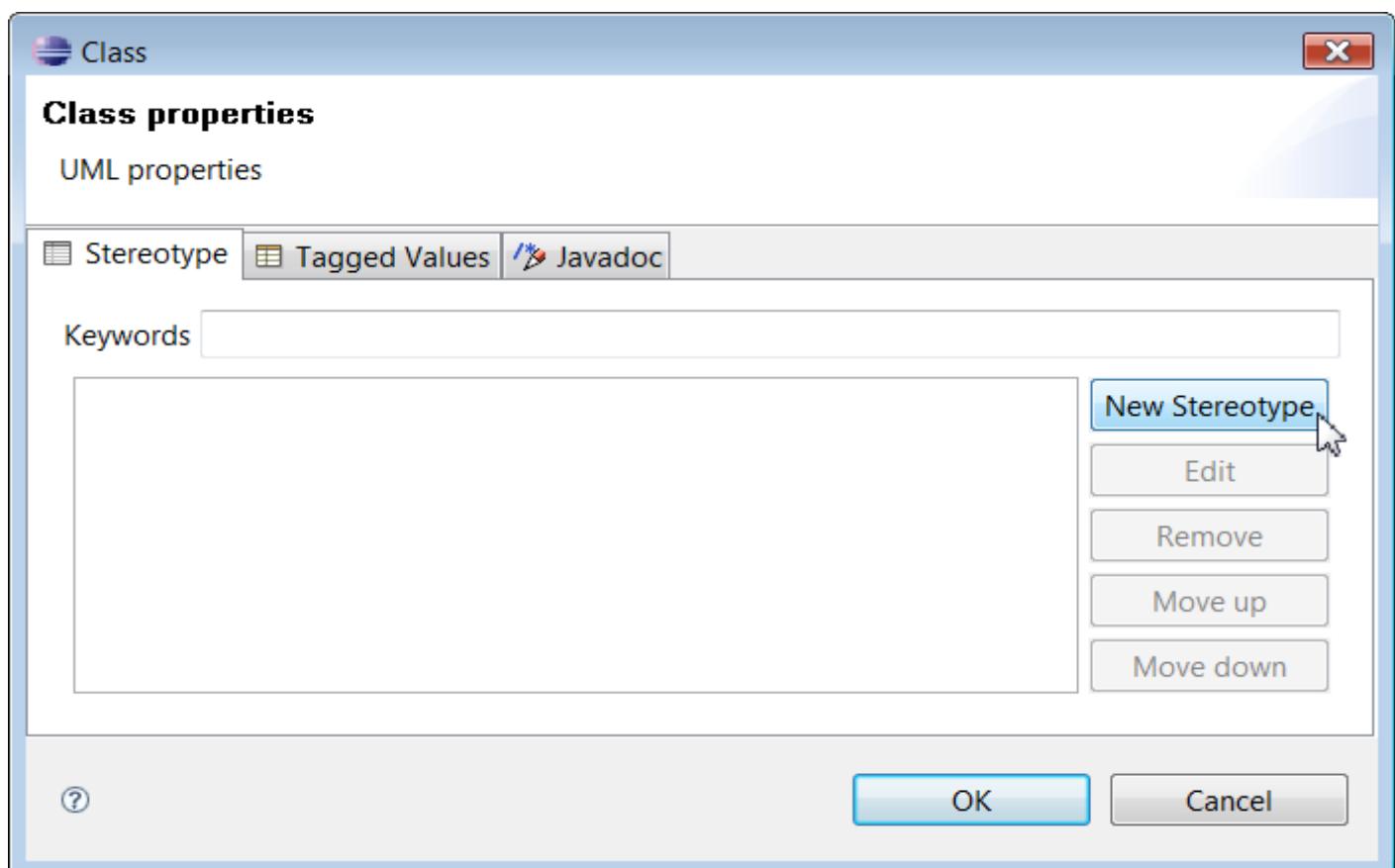
The eclipse class creation dialog (or interface creation dialog) appears.
Enter the name of the class in the Name field and click on the Finish button.



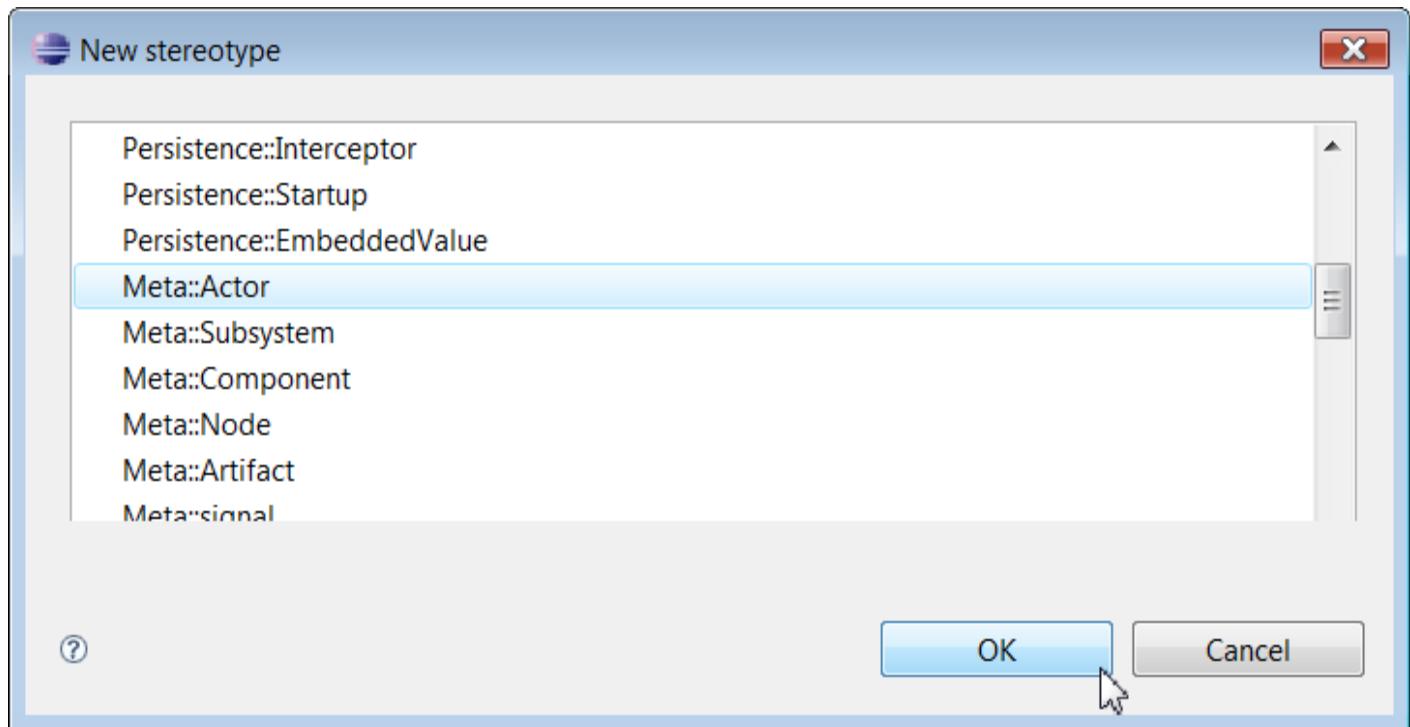
We are going now to add a stereotype on this class.
Click on the **Class inside the UML editor > Properties**.



At this stage you can add a keyword or a stereotype.
To add a keyword you need to type the keyword in the Keywords field.
To add a stereotype **click in the New Stereotype** menu.



Select Actor in the list and click on the Ok button.



After validating the dialog box, the class appears immediately in the class diagram. If you click on the class, its code appears in the java editor area and you can see the UML model which is synchronized with the UML Editor.

The screenshot displays a UML editor interface with several windows:

- *demo.ucd - demo**: Shows a class diagram with a single class named "Person". A tooltip for this class states: "Class Person (actor) Javadoc: ===== Press F7 for focus".
- sample.uml**: Shows the package structure: <Model> sample > Package demo > demo.ucd > <<actor>> <Class> Person
- Properties**: A table showing properties and their values for the Person class:

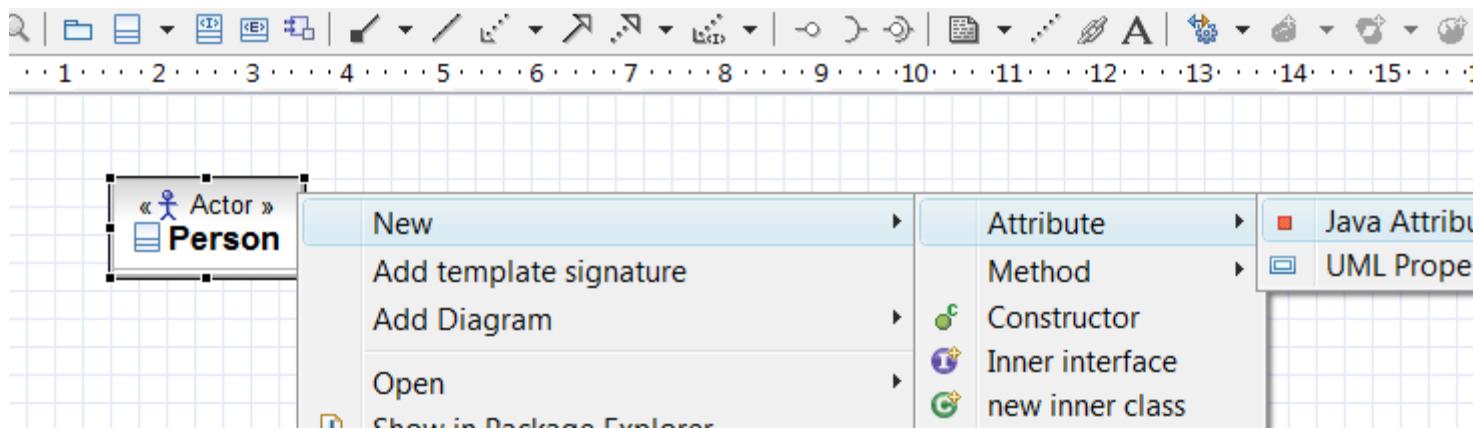
Property	Value
UML	
Classifier Behavior	
Client Dependency	
Is Abstract	false
Is Active	false
Is Leaf	false
Name	Person
Owned Port	
Powertype Extent	
Redefined Classifier	
Representation	
Template Parameter	
Use Case	
Visibility	Public
- Person.java**: Shows the generated Java code:

```
package demo;

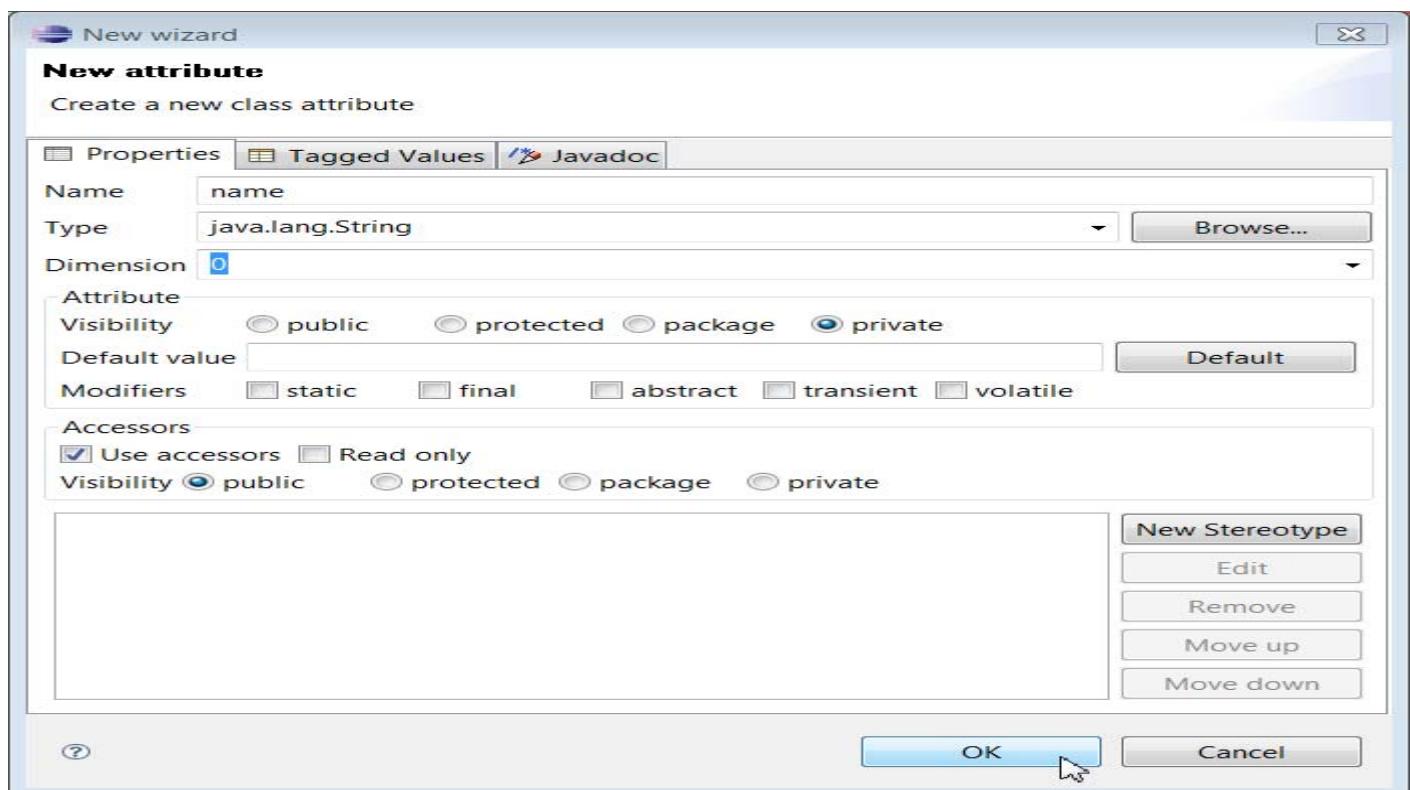
public class Person {
```

Attributes

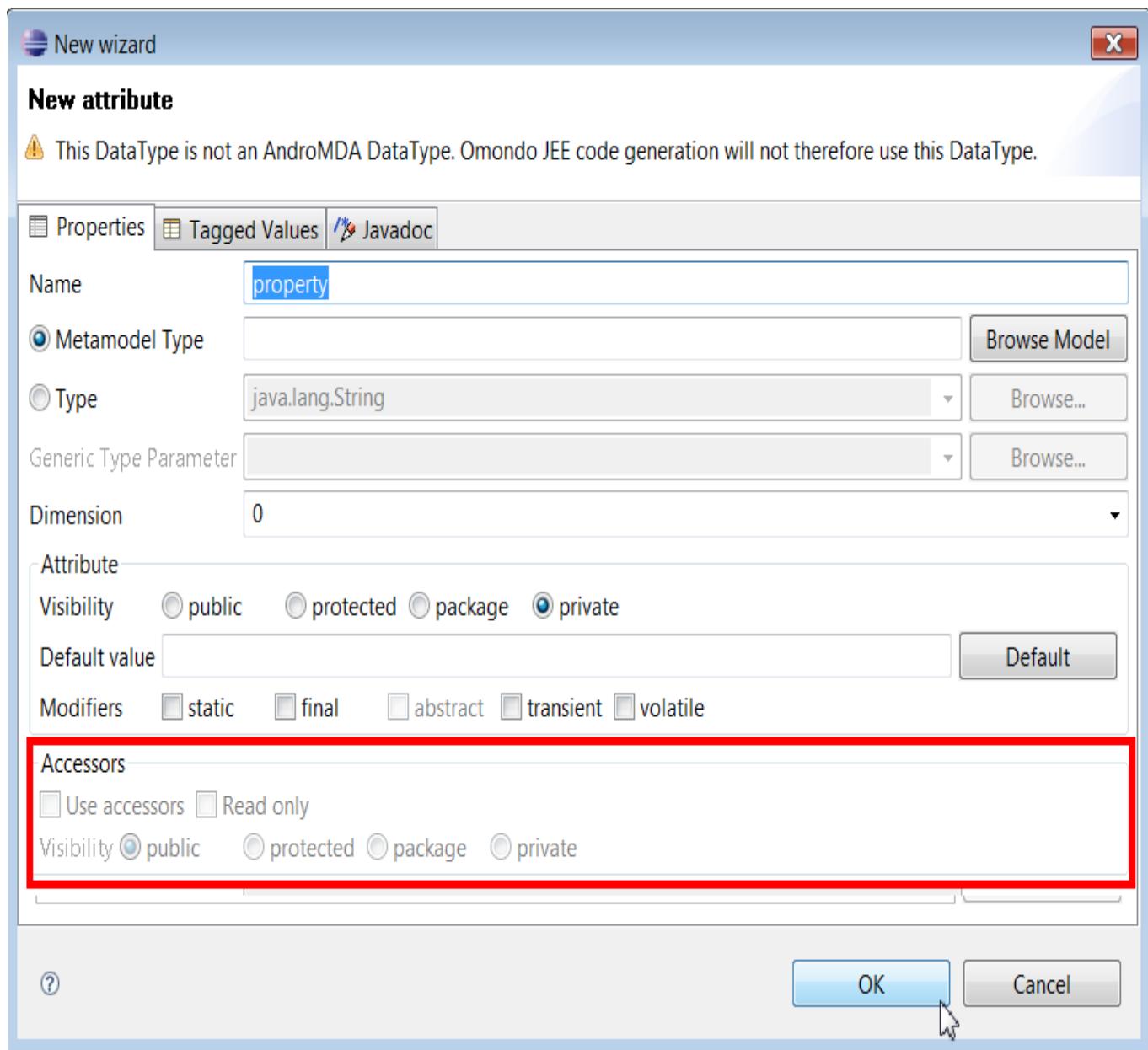
In this section, you will see how to insert new attributes in your classes from the class diagram graphical interface. You can decide to create two kinds of attribute which are either Java or just model. If you select java then the Java code will be generated immediately at the attribute creation in your diagram. If you select Property then only the UML property (e.g. attribute) will be created in your model but not in your code. To create a new attribute, select **New > Attribute > Java Attribute** in the class (or interface) contextual menu.



This launches an attribute creation dialog box. You have to set the attribute name, type, multiplicity, default value, visibility and modifiers. For an interface, the attribute is always **static final**. *Please note that it is not anymore possible to immediately add new stereotype at the attribute creation because of runtime model problem. You now need to create an attribute and then once it is created in your UML Editor to click on the attribute > property in order to add a stereotype.*



Please note that if you select **New > Attribute > UML Property** then the accessor's option will not be selected because only related to java and not having real need in modeling.



After validating the dialog box, the attribute appears immediately in the class. If you click on the attribute, its code appears in the java editor area and in the UML model.

By default, the getter and setter are displayed. In this example, the attribute is private and is related to its public accessors in the java code.

The screenshot shows the UML2 Java Integration interface. On the left, the 'sample.uml' model browser displays a package named 'demo' containing a class 'Person'. The 'Person' class has an attribute 'name' and operations 'setName()' and 'getName()'. On the right, the 'Person.java' code editor shows the generated Java code for the 'Person' class. The code includes private attribute 'name', getter 'getName()', and setter 'setName()' with Javadoc comments.

```

public class Person {

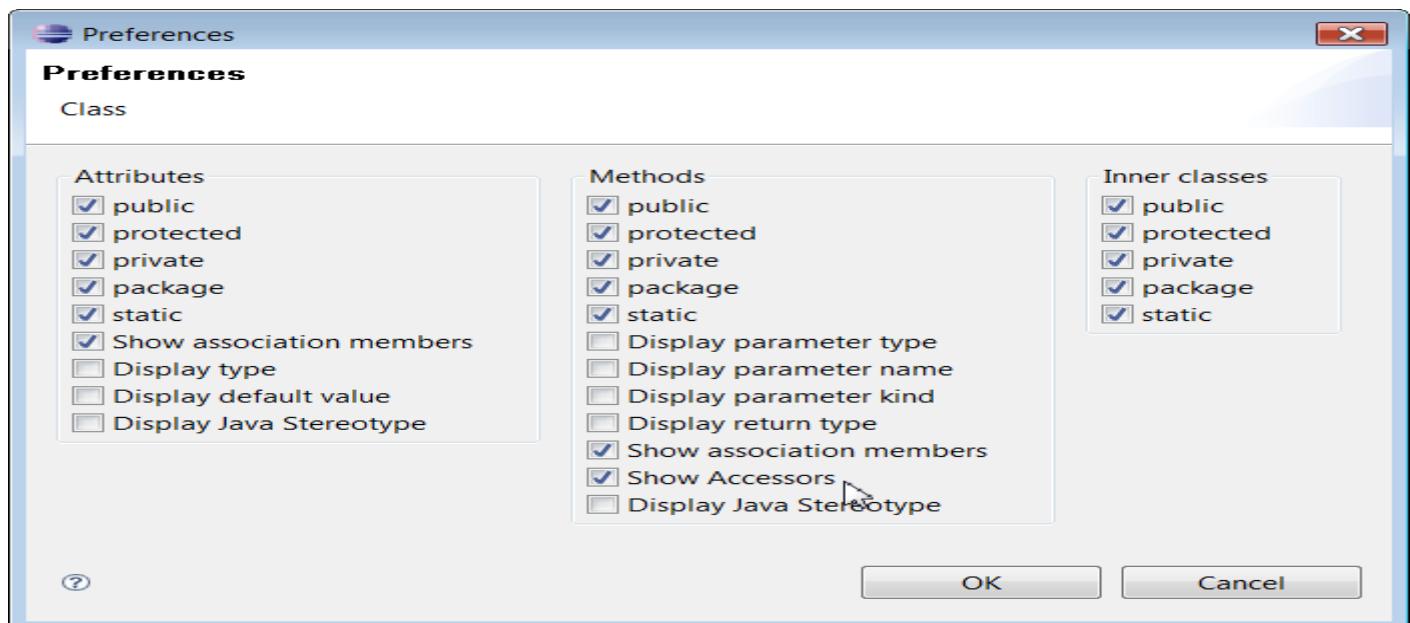
    /**
     * (non-javadoc)
     */
    private String name;

    /**
     * Getter of the property <tt>name</tt>
     *
     * @return Returns the name.
     */
    public String getName() {
        return name;
    }

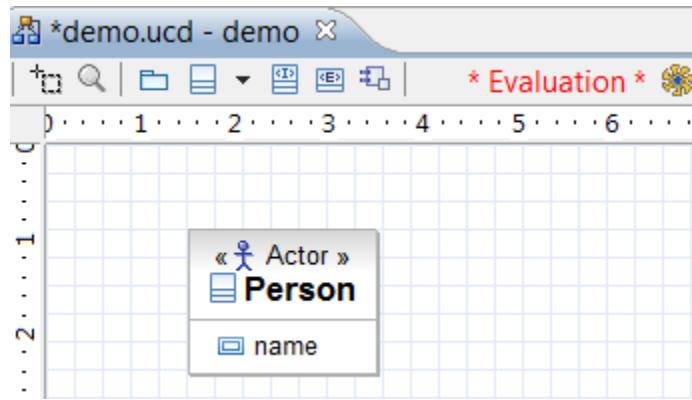
    /**
     * Setter of the property <tt>name</tt>
     *
     * @param name
     *          The name to set.
     */
    public void setName(String name) {
        this.name = name;
    }
}

```

If we want to use the [Property concept](#), then select a class in the class diagram editor.
Open the pop up menu >Preferences - Unselect the Show Accessors check box.



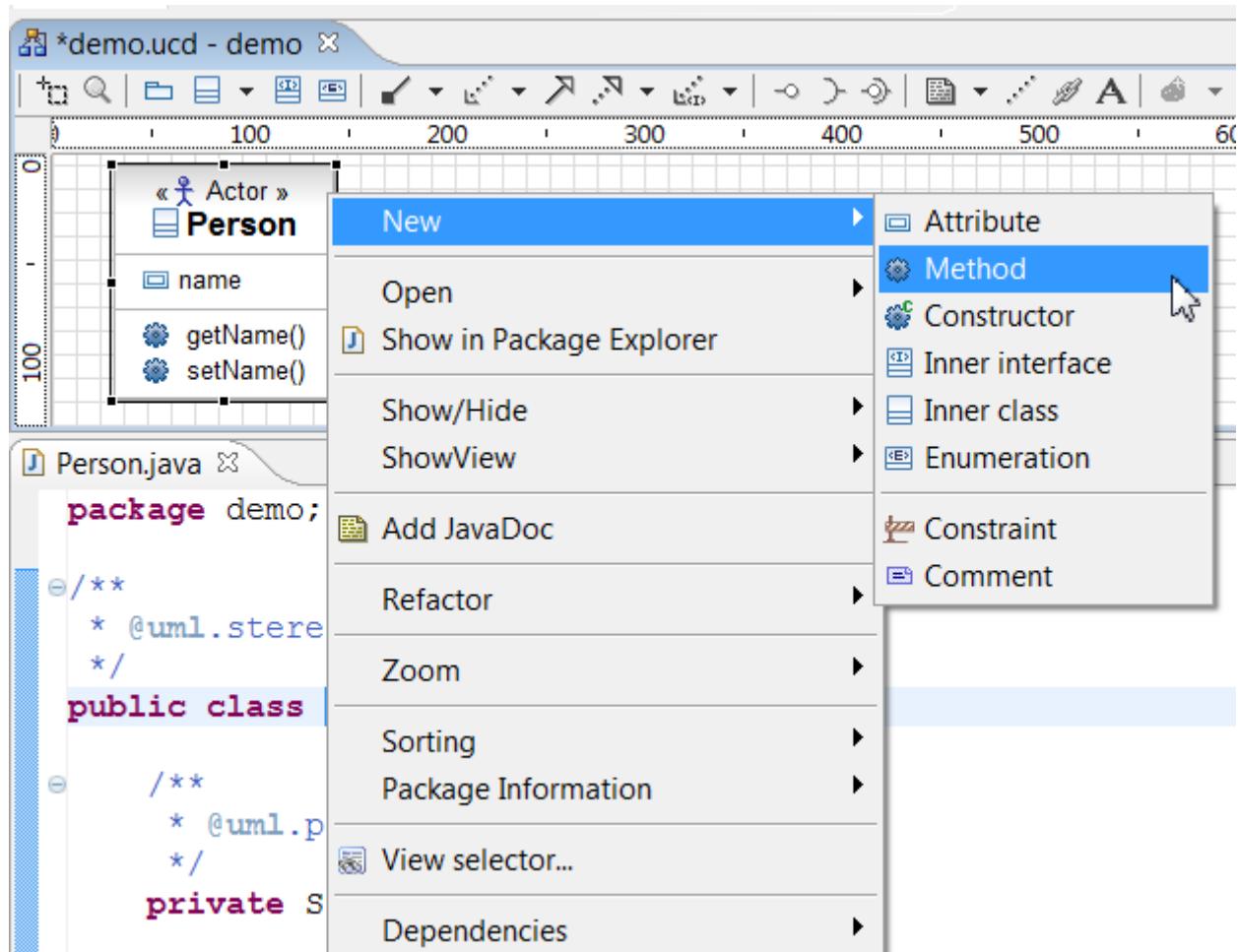
The class diagram editor now hides public methods.



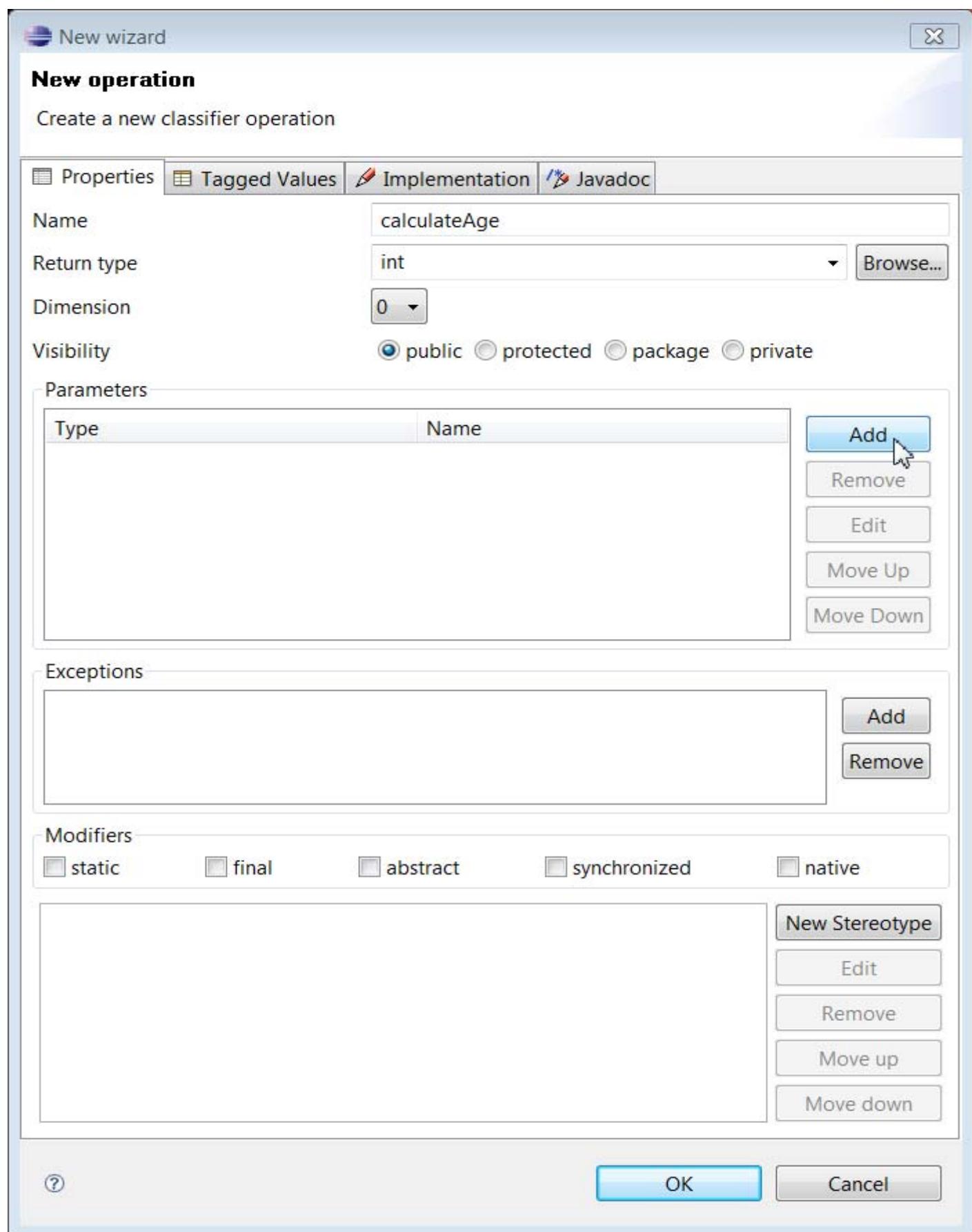
Methods

In this section, you will see how to insert new methods in your classes from the class diagram graphical interface.

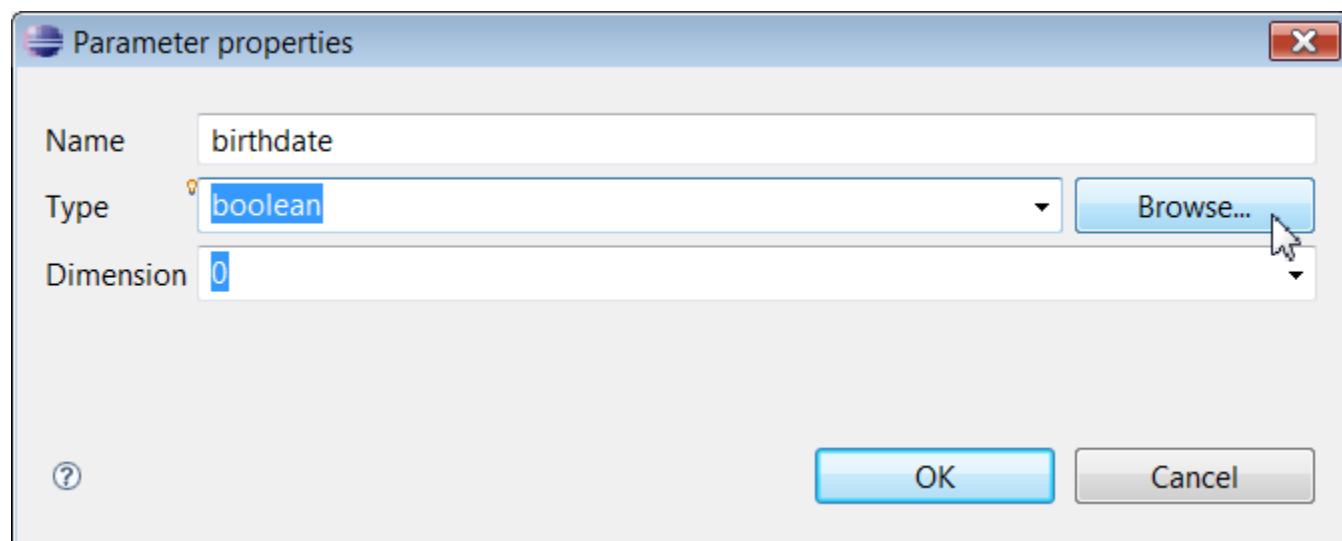
To create a new method, select **New > Method** in the class (or interface) contextual menu.



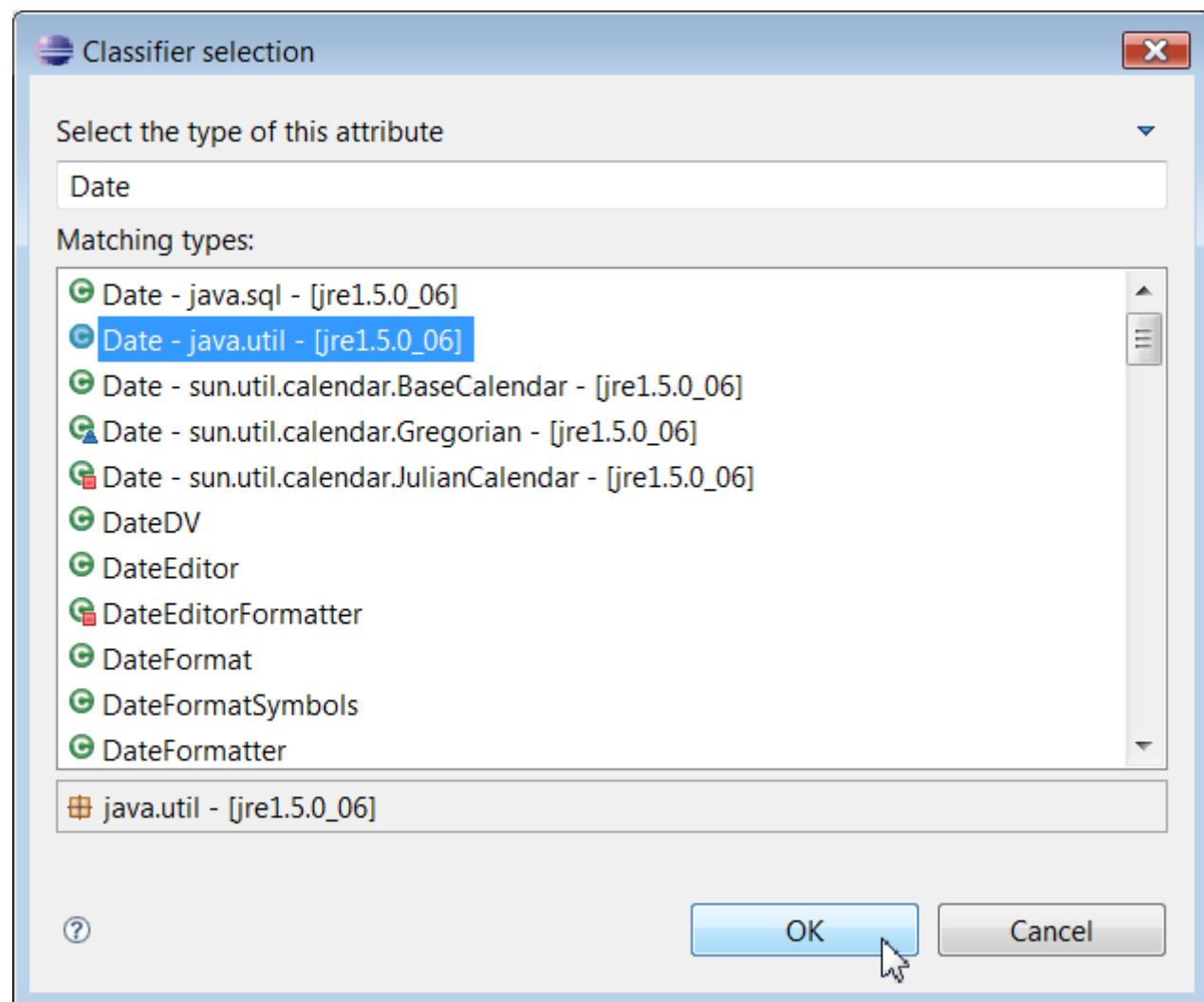
This launches a method creation dialog box. You have to set the method name, return type (with its multiplicity), visibility, parameters, exceptions and modifiers.



For each parameter, you must indicate its name and type.



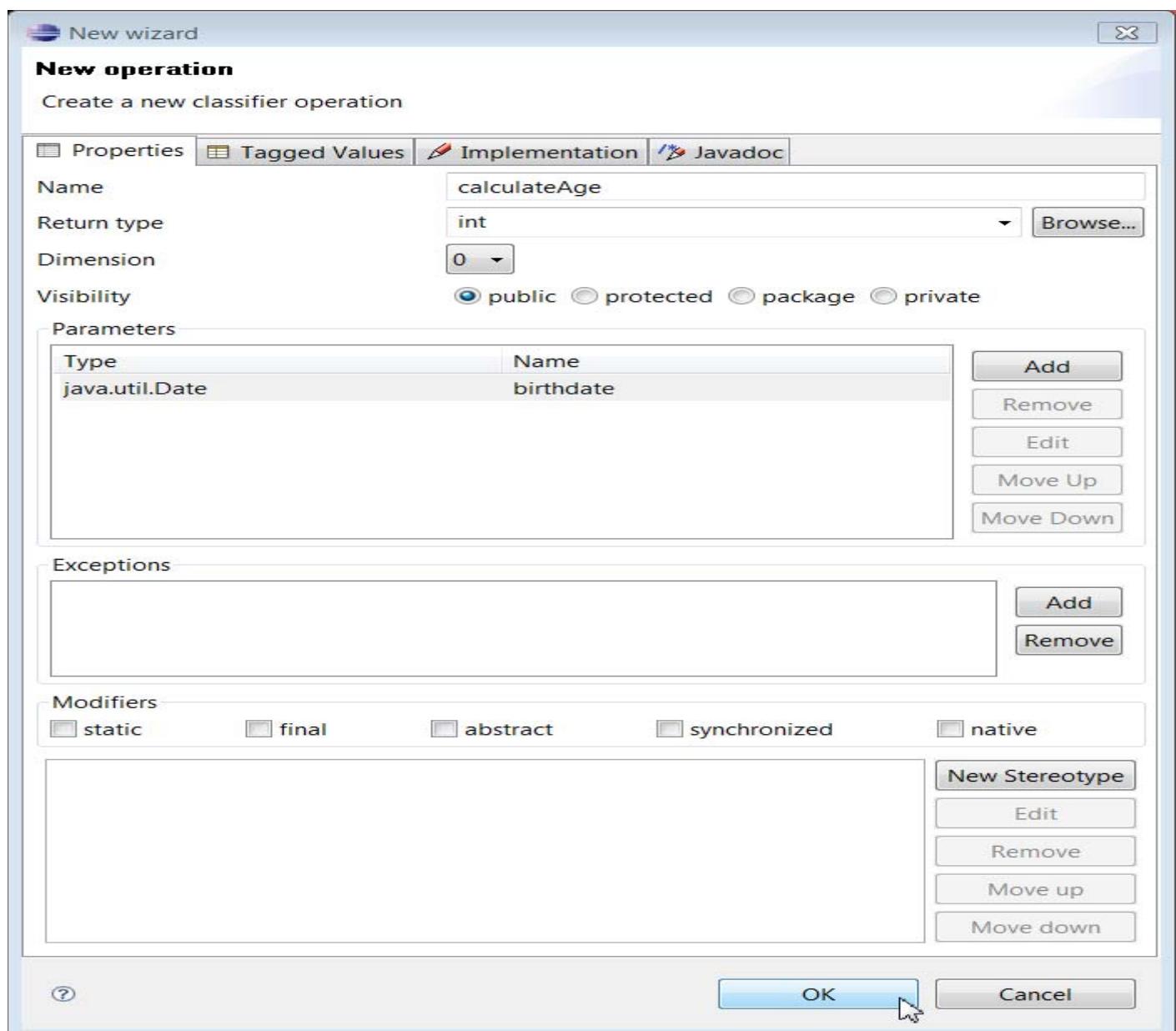
The parameter type is selected through a class finder.



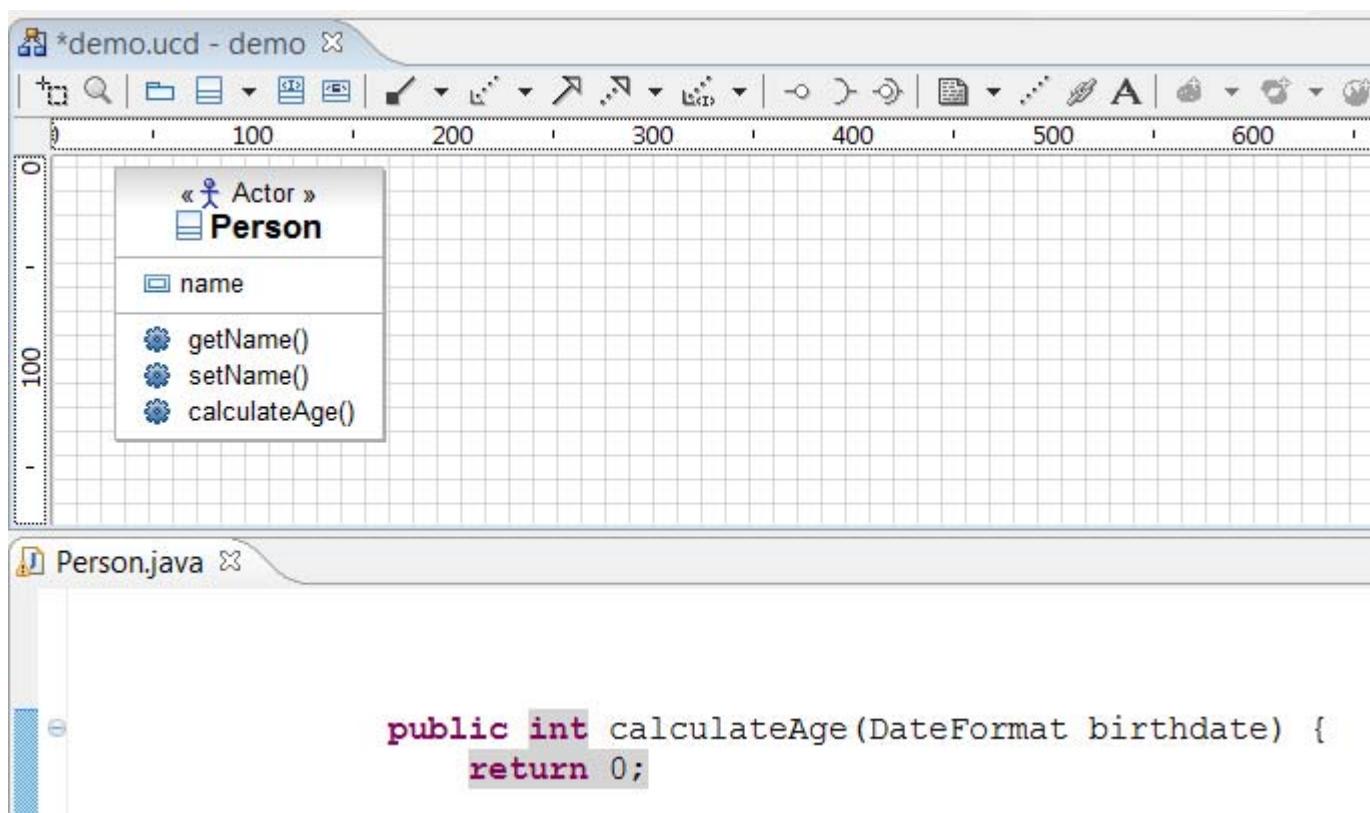
Then you can validate the parameter...



...and the complete method definition.

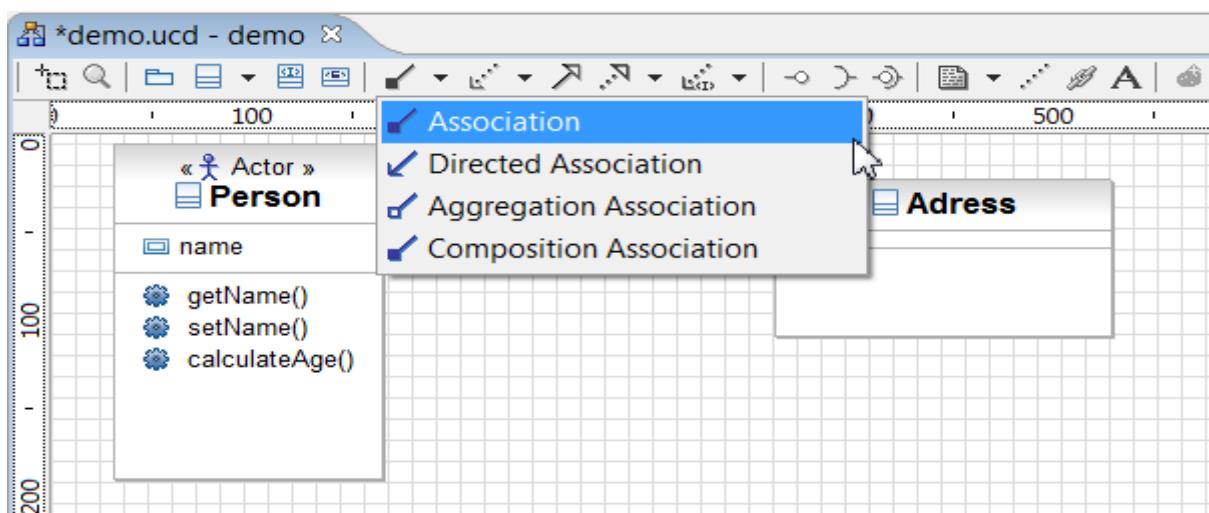


The method appears immediately in the class. If you click on the method, its code appears in the java editor area.

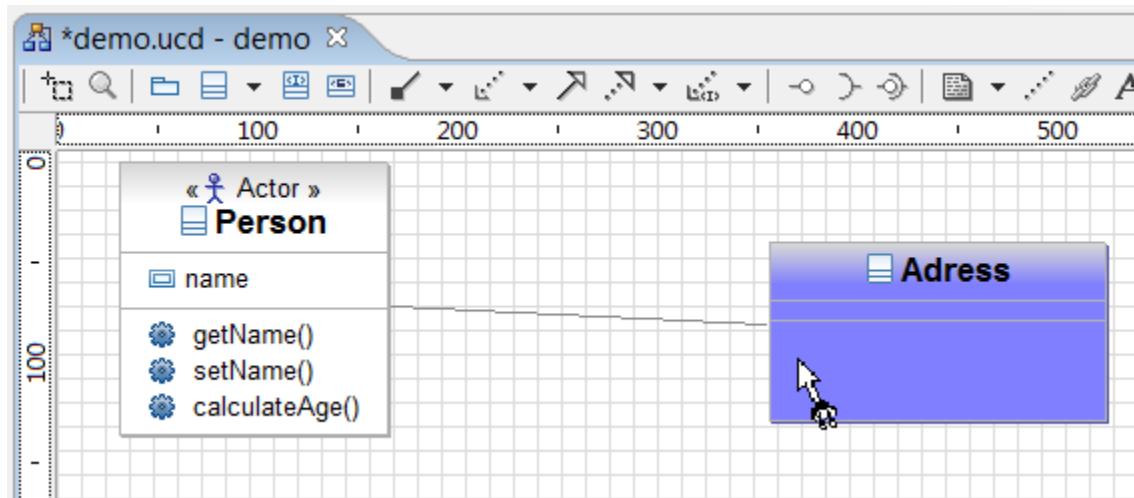


Associations

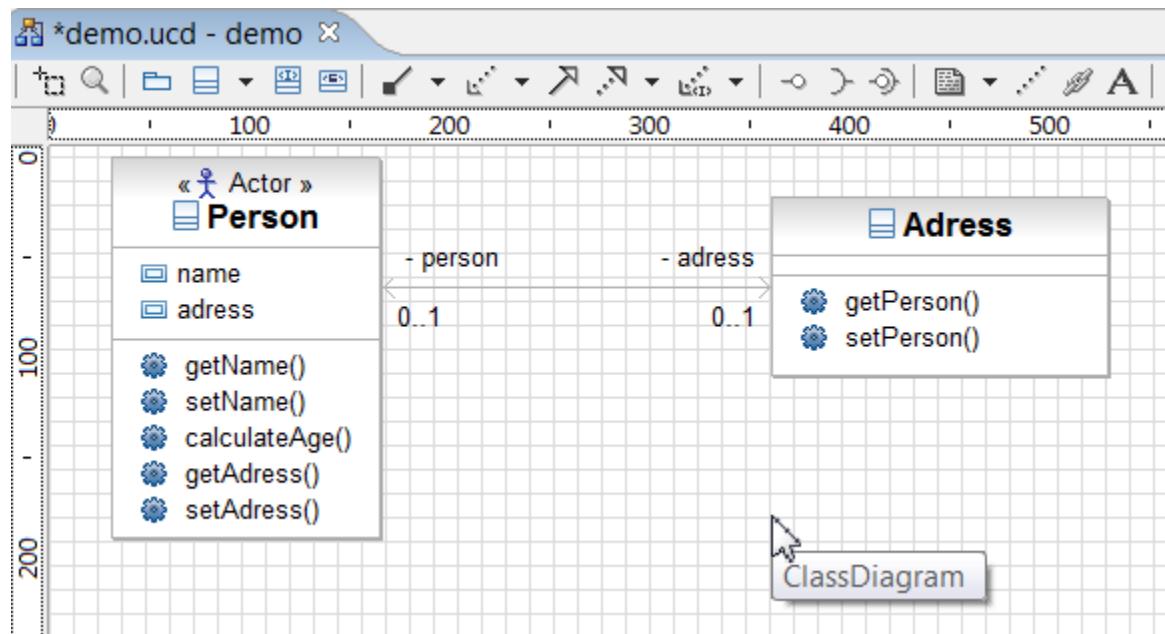
In this section, you will see how to insert new associations in your classes from the class diagram graphical interface. To create an association, select the *Association* toolbar button then click in the original class. We suppose that you have previously created another class named "Address". Select **Association** in the top down menu.



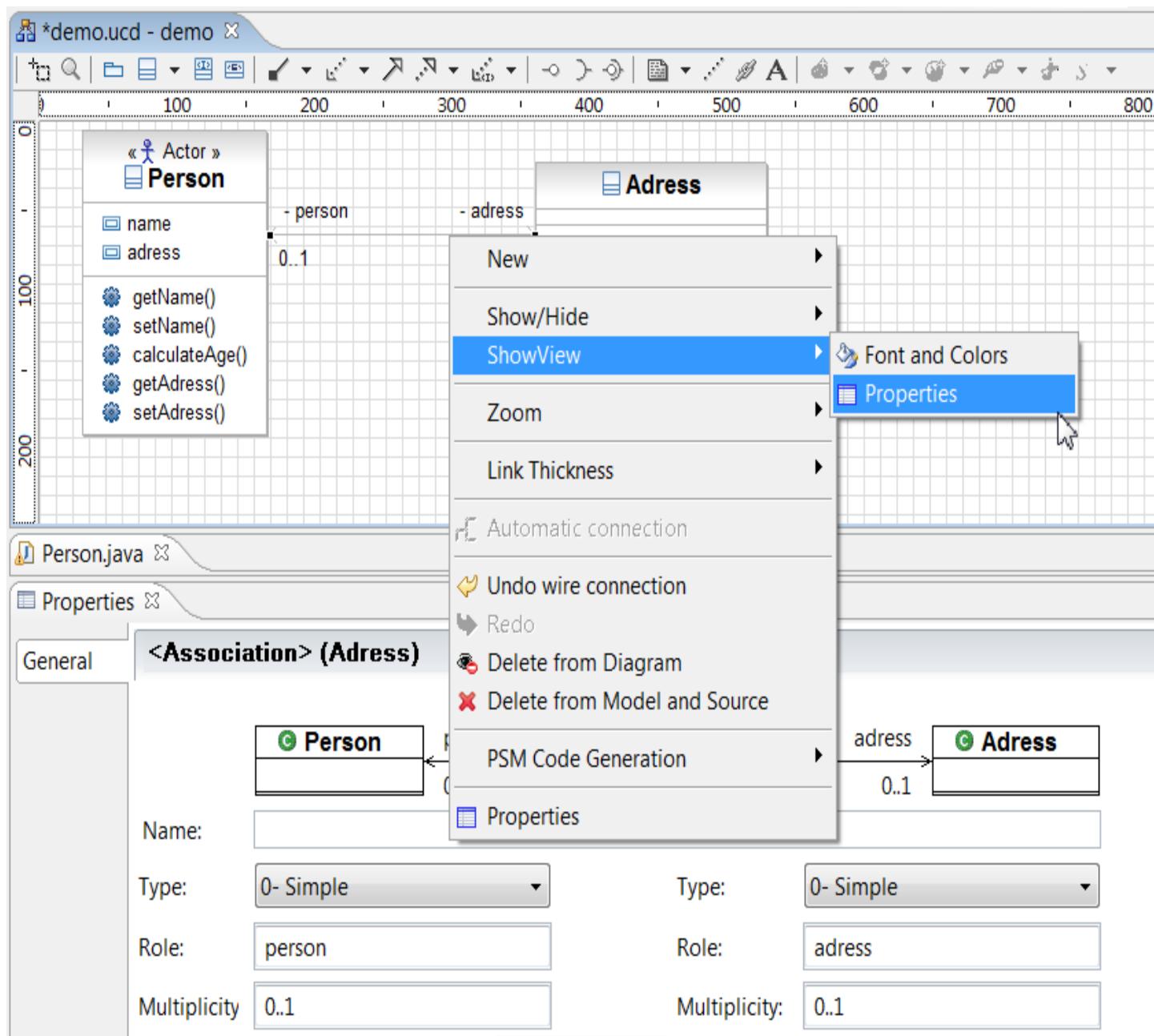
Drag and drop the association to the class named Person", then click on the class, and move the other end of the association to the class named Address.



Click on the class named Address and release the mouse button.



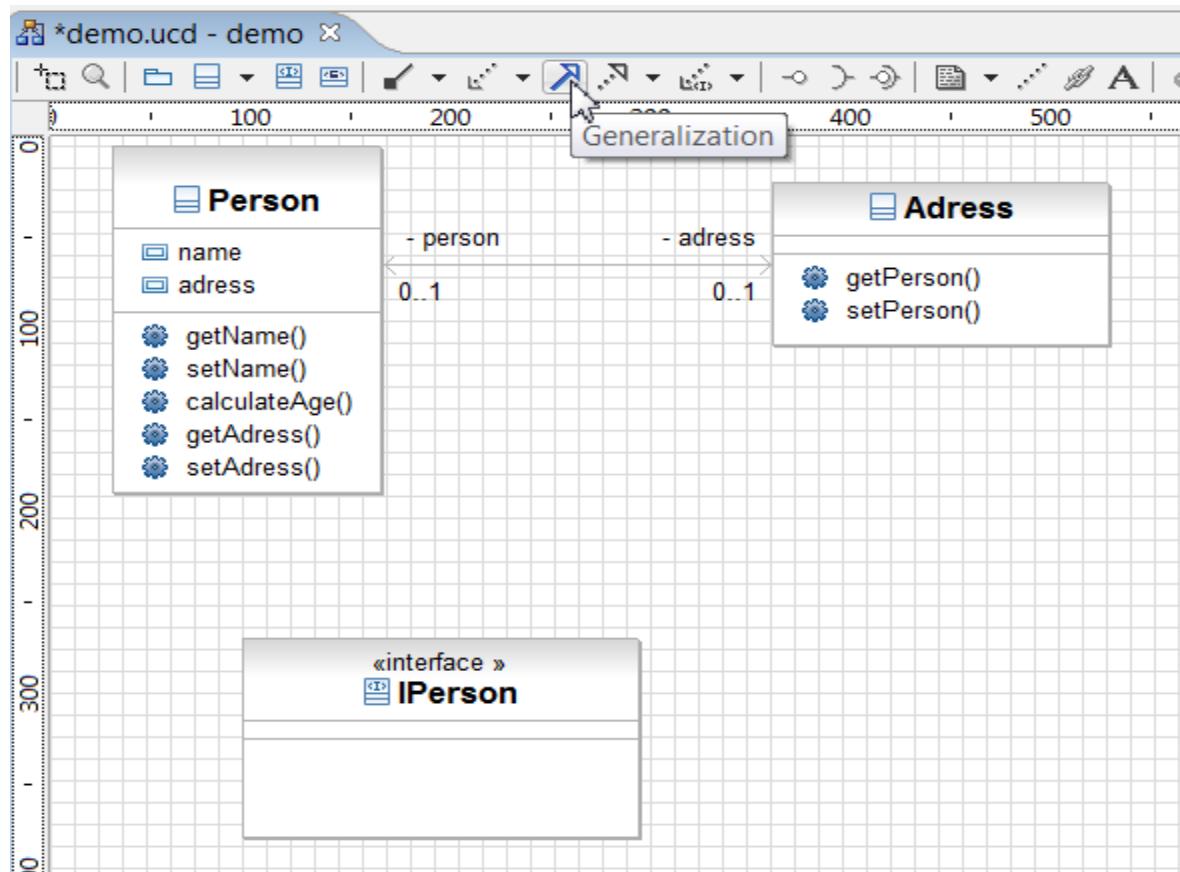
You can change properties either by selecting the association link and select **ShowView > Properties**, or by a double click on the association.



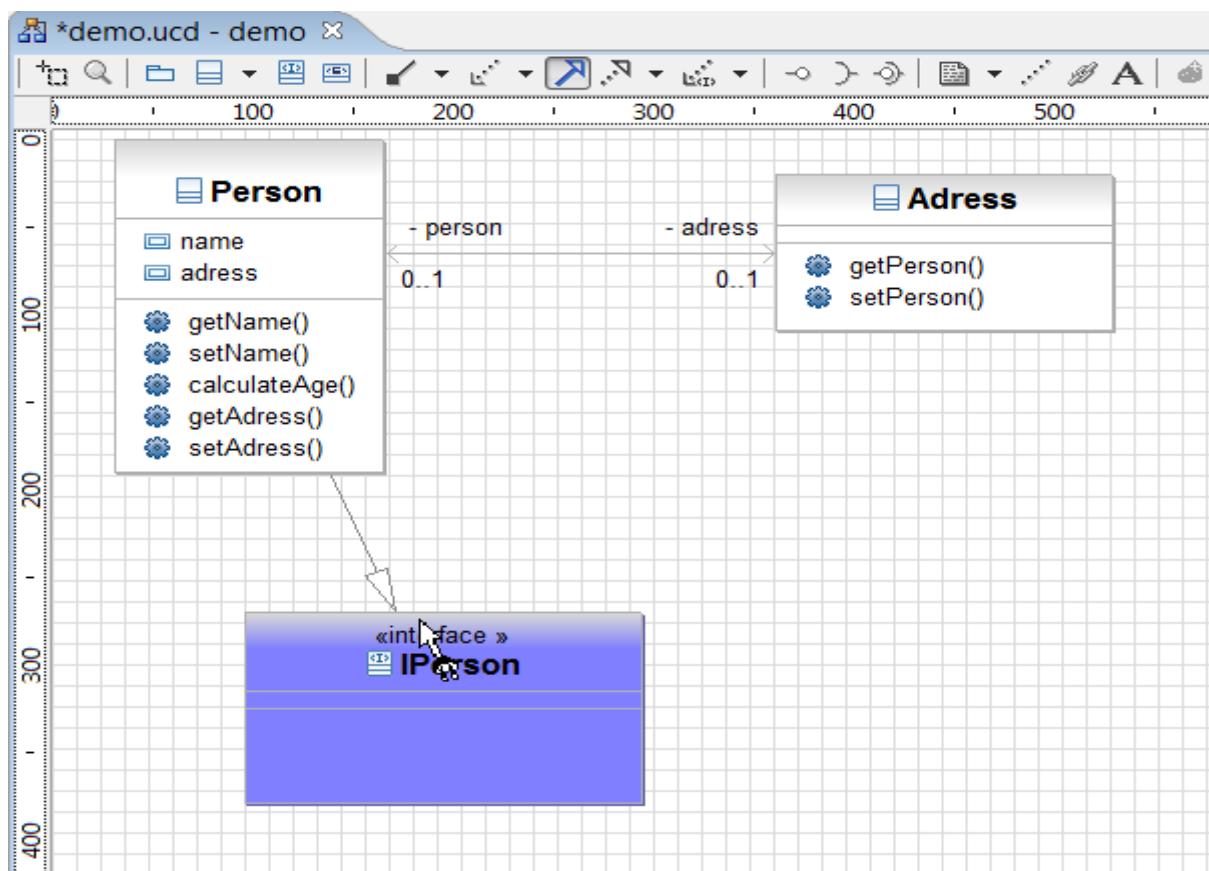
Implementation and Inheritance

In this section, you will see how to manage implementation and inheritance in your classes from the class diagram graphical interface.

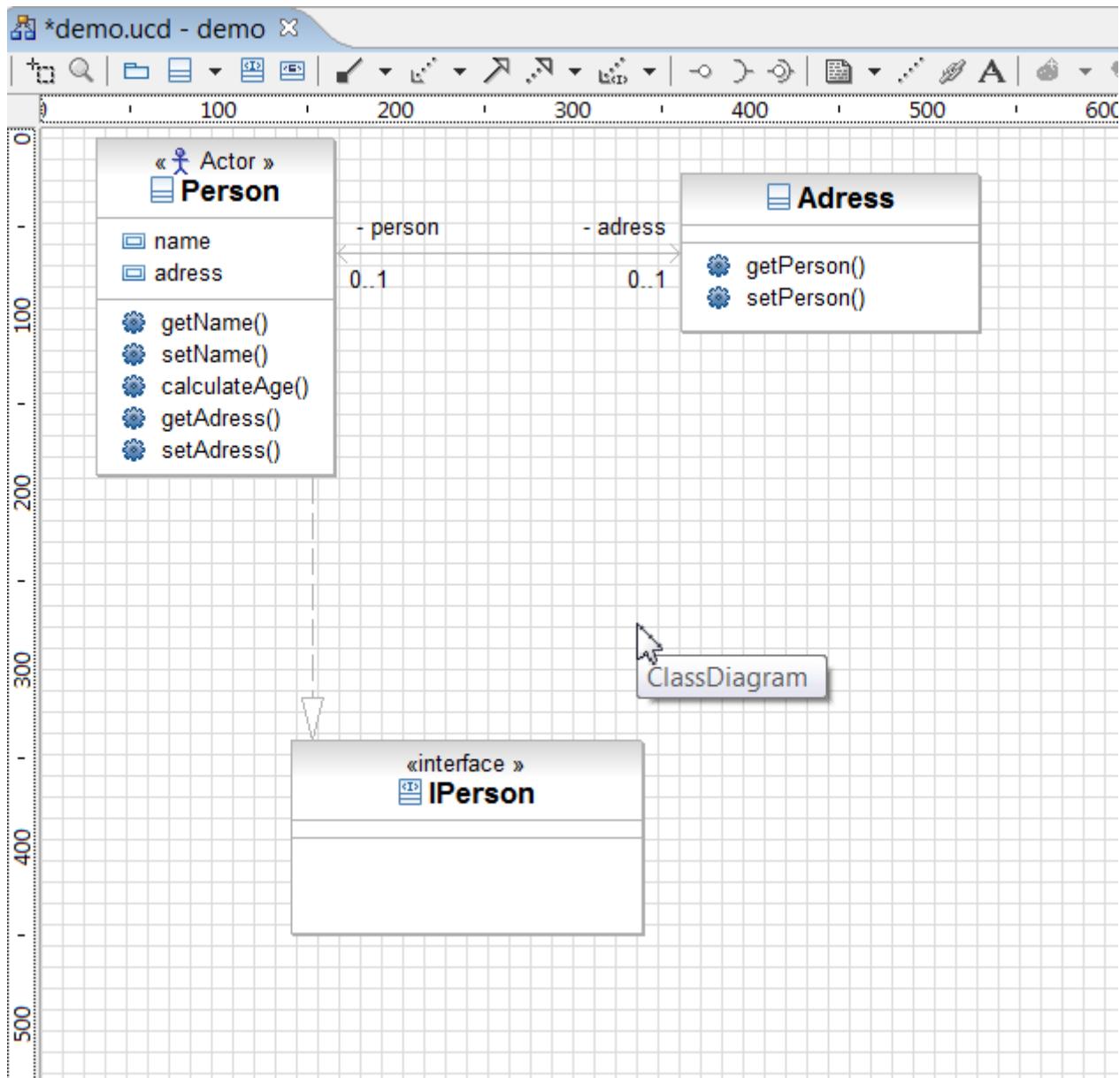
To declare an implementation or inheritance link, select the generalization toolbar button.



Drag and drop the generalization link from the class named person to the Interface named IPerson.



Then release the mouse in the target class or interface. The system will automatically recognize if the link must be an implementation or an inheritance.



Diagrams

In this section you will learn how to use the different EclipseUML diagrams.
The following areas are covered:

[Class Diagram](#)

- [Concept](#)
- [Drag'n'Drop](#)
- [New](#)
- [Insert](#)
- [Extend UML 2](#)
- [Package Elements](#)
- [Show Hide](#)
- [Show View](#)
- [Properties View](#)
- [JavaDoc](#)
- [Refresh XMI 2.1 Editor](#)

- [Sorting](#)
- [Package Information](#)
- [Show Hide Links](#)
- [Resize All Shapes](#)
- [Arrange Diagram](#)
- [PSM Model Driven Development Activation](#)
- [Class Diagram View](#)

- [Navigation](#)
- [Real-Time Synchronization](#)
- [Refactoring](#)
- [Working At Package Level](#)
- [View Selector](#)
- [Property Concept with detailed show hide Accessors option](#)

- [Association Class](#)
- [Format](#)
- [Toolbar](#)
- [Class Diagram Example](#)
- How to use the Class diagram contextual menus (4min): [Flash demo](#) / [.exe file](#)
- Association Class use (3min): [Flash demo](#) / [.exe file](#)
- Enumeration modeling features (2min): [Flash demo](#) / [.exe file](#)
- EclipseUML navigation (3min): [Flash demo](#) / [.exe file](#)
- Attribute is not display in my diagram - Why ? (3min) [Flash demo](#) / [.exe file](#)
- How to use Manhattan and manual links property (2min) [Flash demo](#) / [.exe file](#)
- How to use automatic anchor property (1min) [Flash demo](#) / [.exe file](#)

Concept

Class Diagrams are static and show an overview of a system. A system is composed of classes and the relationships among them.

The Class Diagram model describes the static structure of the symbols in the system. This model allows you to graphically represent symbol diagrams containing classes. Classes are arranged in hierarchies sharing common structure and behavior and are associated with other classes.

UML class diagrams show the classes of the system, their inter-relationships, and the operations and attributes of the classes.

Class diagrams are used to:

- analyze requirements in the form of a conceptual/analysis model
- explore domain concept in the form of a domain model
- depict the detailed design

Omondo is now introducing a new Agile Modeling concept using both Java having live UML code synchronization and metamodel elements.

To see how to mix both Java and UML elements see topic: [Drag and drop](#)

In a class diagram different elements are available:

1. **New Packages** can be created in the Class Diagram Editor :

- Using the class diagram toolbar.
- Using the class diagram editor's popup menu (Right click inside the class diagram editor >insert>package).
- Using drag and drop from the Package Explorer to the class diagram editor.

The [look and feel](#) of the package elements can be changed depending on the selected diagram presentation style. Three different diagrams presentation styles can be applied to the Class Diagram Editor :

- Omondo presentation



- UML Presentation



- Eclipse Presentation



2. **New Classes** can be created in the Class Diagram Editor :

- Using the class diagram toolbar.
- Using the class diagram editor's popup menu (Right click inside the class diagram editor >new>class).

 Class Using the class diagram editor's popup menu (Right click inside the class diagram editor >insert>class).

 Using drag and drop from the Package Explorer to the class diagram editor.

The look and feel of the class elements can be changed depending on the selected diagram presentation style. Three different diagrams presentation styles can be applied to the Class Diagram Editor :

- Omondo presentation



- UML Presentation



- Eclipse Presentation



3. **New Interfaces** can be created in the Class Diagram Editor :

 Using the class diagram toolbar.

 Interface Using the class diagram editor's popup menu (Right click inside the class diagram editor >new>Interface).

 Interface Using the class diagram editor's popup menu (Right click inside the class diagram editor >insert>class).

 Using drag and drop from the Package Explorer to the class diagram editor.

The look and feel of the interface elements can be changed depending on the selected diagram presentation style. Three different diagrams presentation styles can be applied to the Class Diagram Editor :

- Omondo presentation



- UML Presentation



- Eclipse Presentation

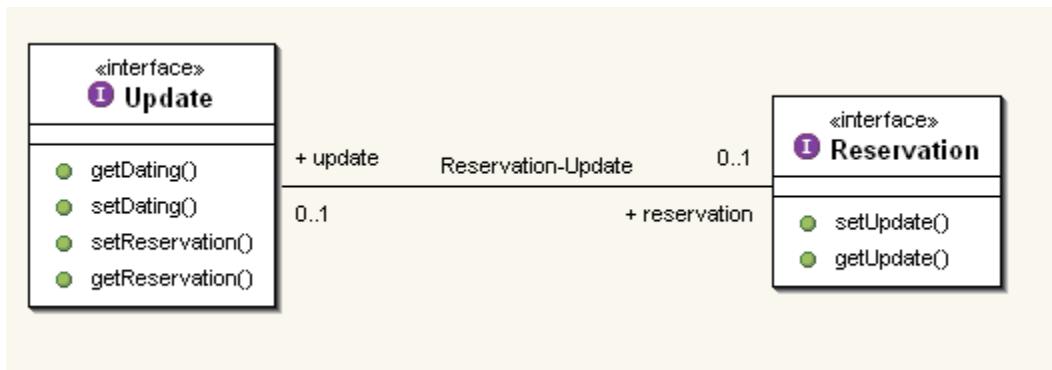


4. **New Associations** between classes and interfaces can be created in the Class Diagram Editor :
- Using the class diagram toolbar.

The creation of a association between classes and interfaces has four steps using the mouse:

1. Select the association icon in the toolbar.
2. Select a class or an interface (the color of the element becomes blue that means it is selected), this is the first association end.
3. Hold down the mouse button and drag the link from the first element to the second, this is the second association end.
4. Release the mouse button.

The properties of the new association are defined through the [preferences](#).



5. **New Generalizations** between classes and interfaces can be created in the Class Diagram Editor :

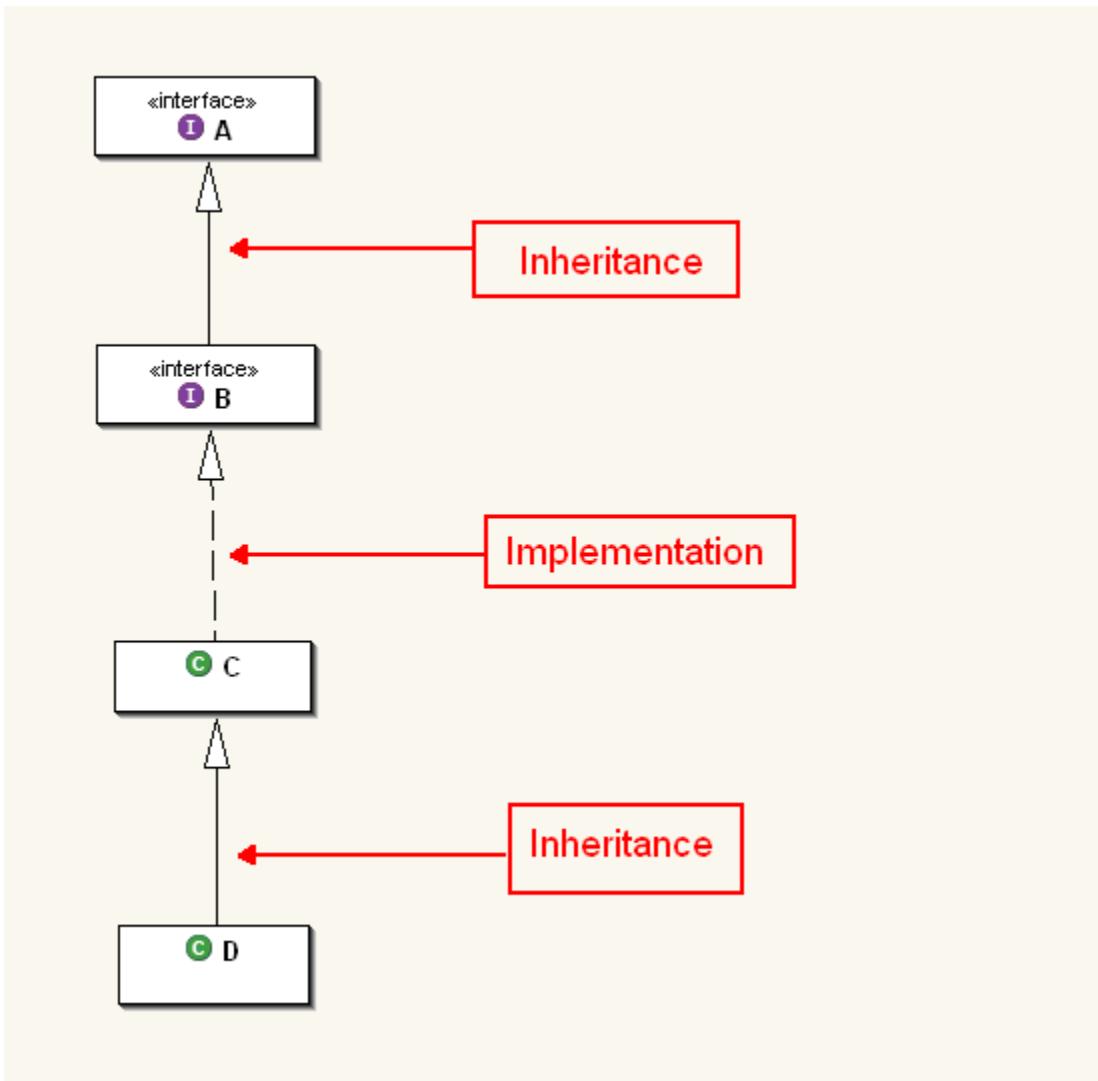
- Using the class diagram toolbar.

The creation of a generalization between classes and interfaces has four steps using the mouse:

1. Select the generalization icon in the toolbar.
2. Select a class or an interface (the color of the element becomes blue that means it is selected).
3. Hold down the mouse button and drag the link from the first element to the second.
4. Release the mouse button.

Depending on the nature of the selected elements, the created generalization is an inheritance or an implementation.

The properties of the new generalization are defined through the [preferences](#).



6. **New Dependencies** between any elements can be created in the Class Diagram Editor:
↗ Using the class diagram toolbar.

The creation of a dependency between two elements has four steps using the mouse:

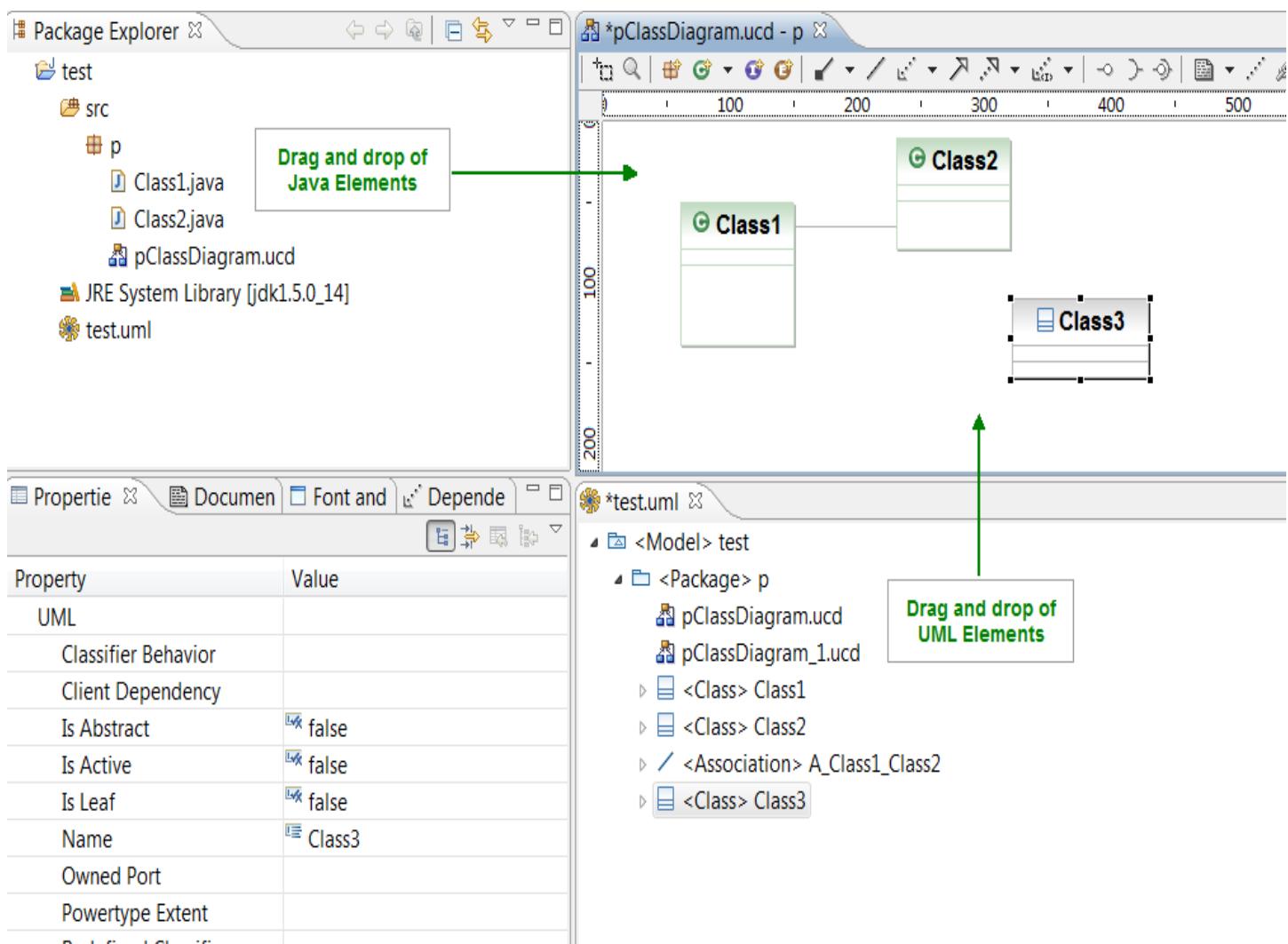
1. Select the dependency icon in the toolbar.
2. Select an element (the color of the element becomes blue, which means it is selected).
3. Hold down the mouse button and drag the link from the first element to the second.
4. Release the mouse button.

The properties of the new dependency are defined through the [preferences](#).

Drag and Drop

In a class diagram, you can drag and drop both Java and UML elements from:

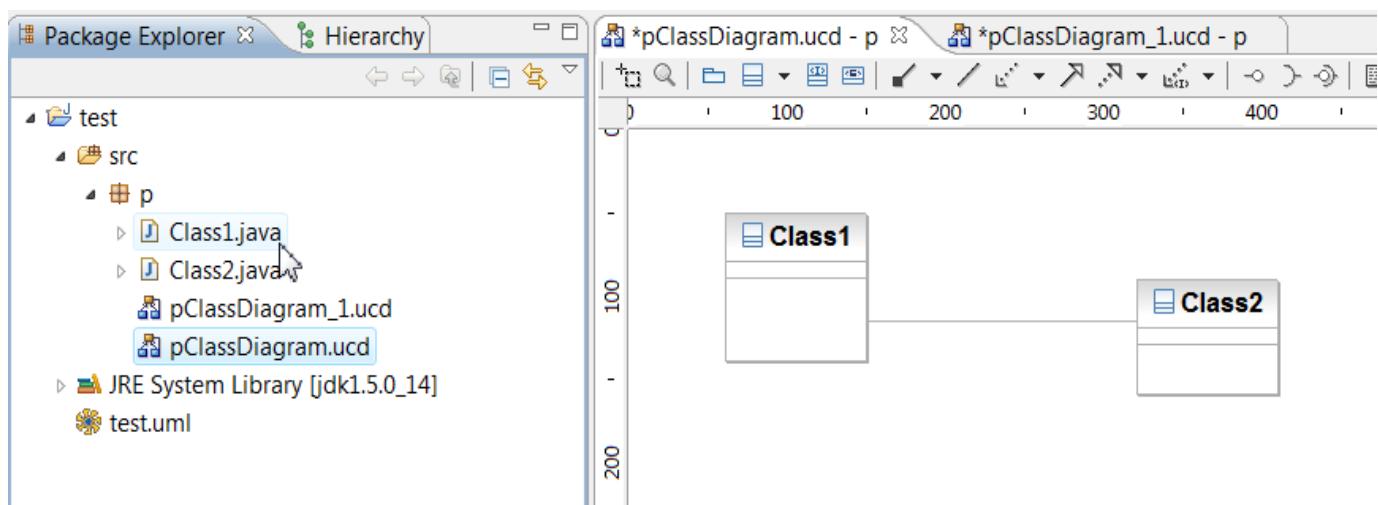
1. [Package Explorer](#)
2. [Model Editor](#)



1. From the Package Explorer you can drag and drop the following Java elements:

- Package
- Class
- Interface

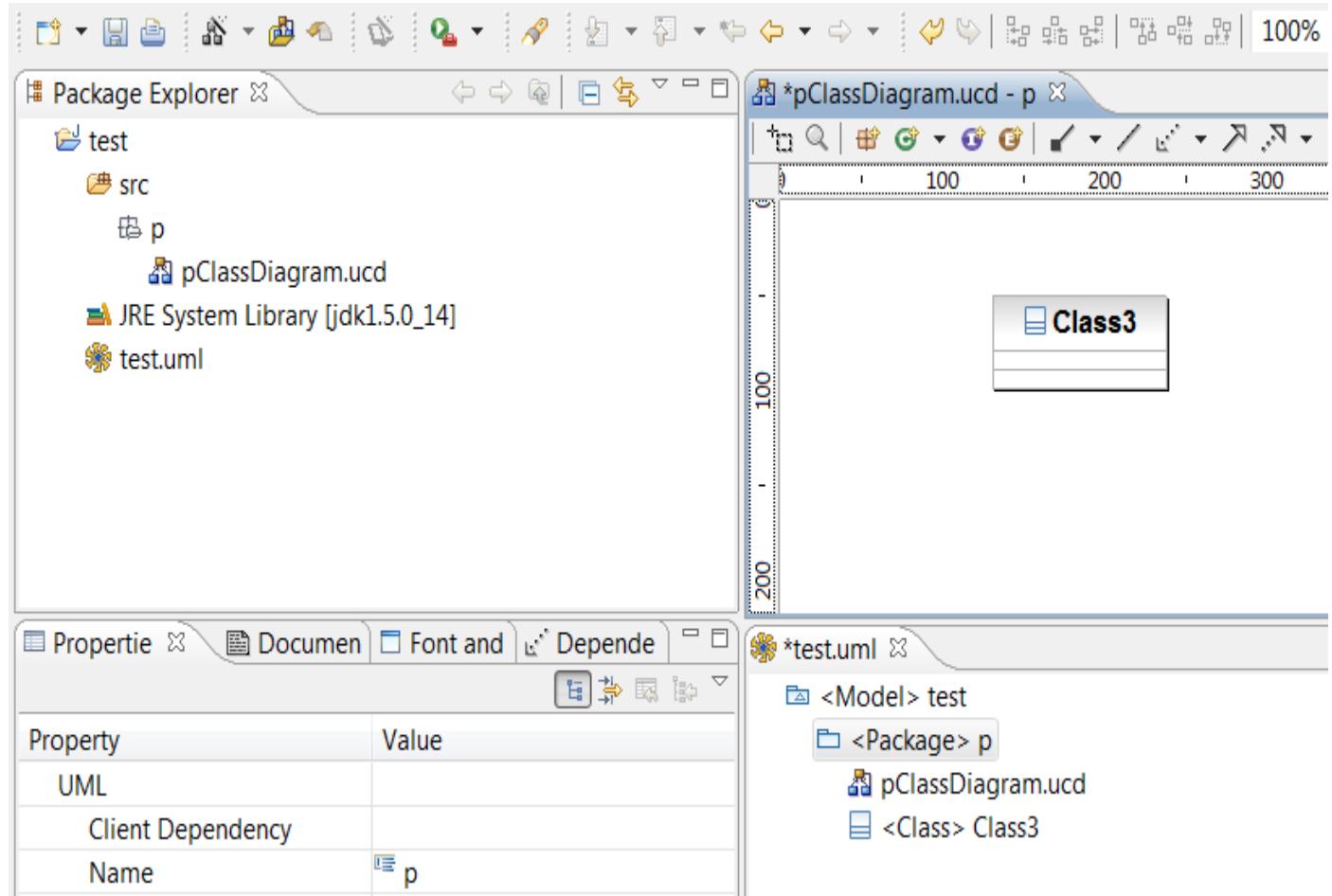
Select one element in your Package Explorer and drag it to the Class Diagram.
Release the mouse button to drop it into your diagram editor (e.g. *Class1* & *Class2*).



2. From the Model Editor you can drag and drop the following Java or UML (*just a metamodel element not related to Java*) elements:

- Package
- Class
- Interface

Select one element in your Model Editor and drag it to the Class Diagram.
Release the mouse button to drop it into your diagram editor (e.g. Class3)



New

This section describes how to add new Java or model elements to your Class Diagram.
It is possible to create Java or only UML Superstructure model elements (e.g. no Java code generation).

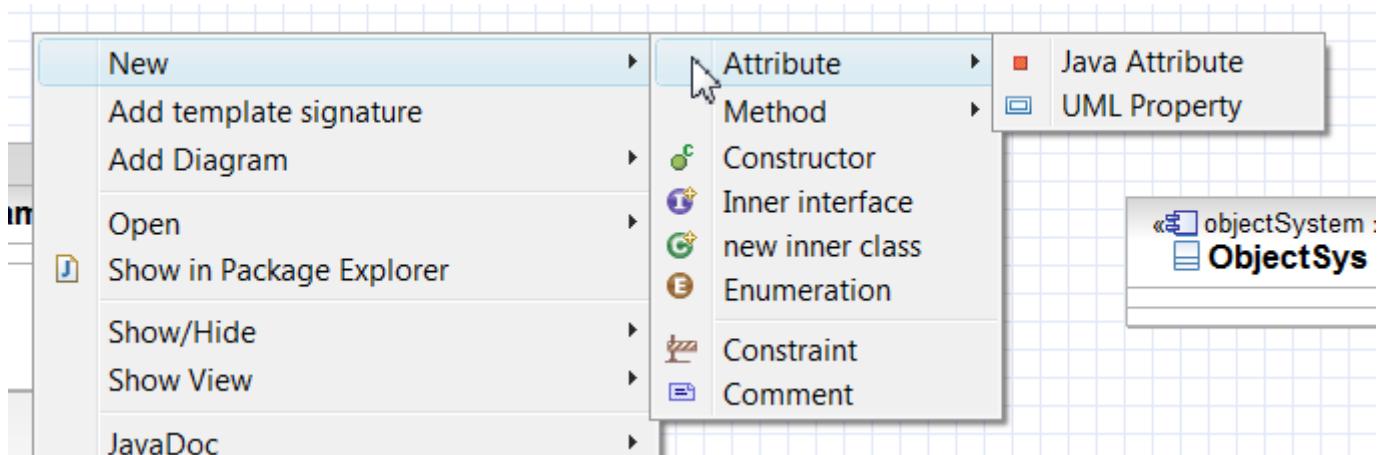
1. [New Java Elements](#)
2. [New UML Metamodel Elements](#)

The **New menu** can immediately for example create from the UML Editor new:

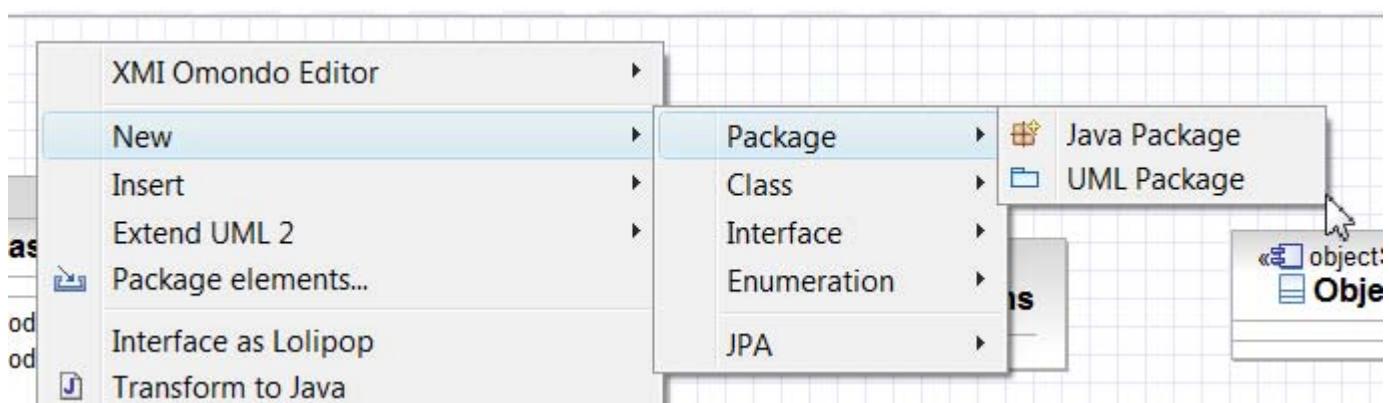
- Class, Interface, Enumeration or Packages in your Java Code.
- Class, Interface, Enumeration or Packages in your UML Superstructure Model.

The **New menu** is available at:

- **Classifier level** by selecting for example a **Class > New > ...**



- **Class Diagram level** by selecting the **diagram background > New > ...**



1. New Java Elements

1. [Create New Classifiers \(e.g. Class, Interface and Enum\)](#)
2. [Add a metamodel method \(no Java Code generation for the method\) to a Java Class](#)

1. 1 Create a new Java classifier

Java elements can be added to the Class Diagram using the [Package Explorer drag and drop](#) or using contextual menus.

You can add new classifiers using the class diagram contextual menu.

Click on the diagram background (be sure that the mouse is not selecting any other element) > **New ...**

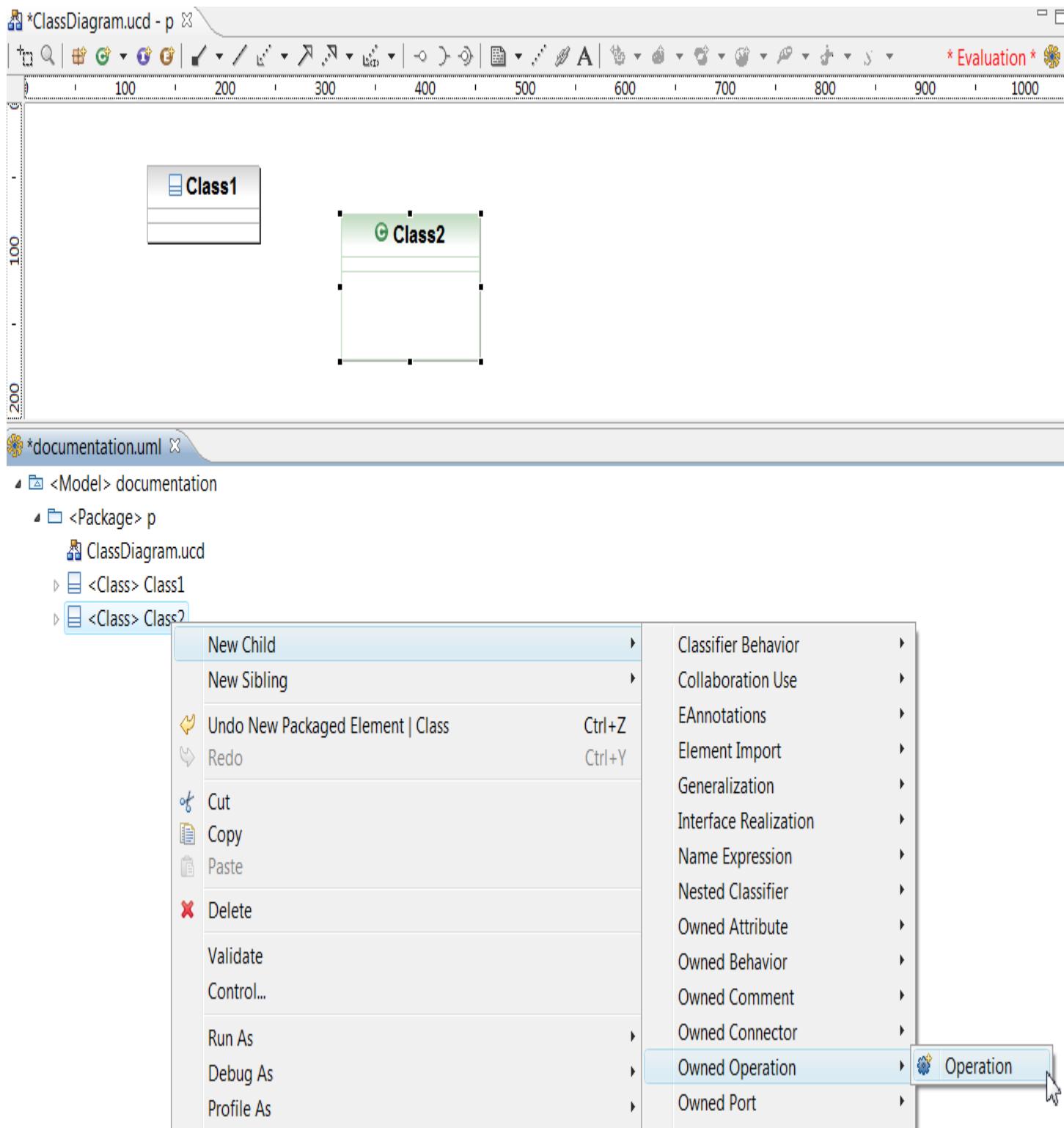
- Package
- Class
- Interfaces
- Enumeration

Please note that the Jee elements will automatically be shown in the class diagram New menu, after you have selected the [Class Diagram PSM /Model Driven Development menu](#).

1.2 Add a metamodel method (no Java Code generation for the method) to a Java Class

UML Property and Operation can be added in the metamodel and not created in the Java code using the drag and drop XMI Editor option.

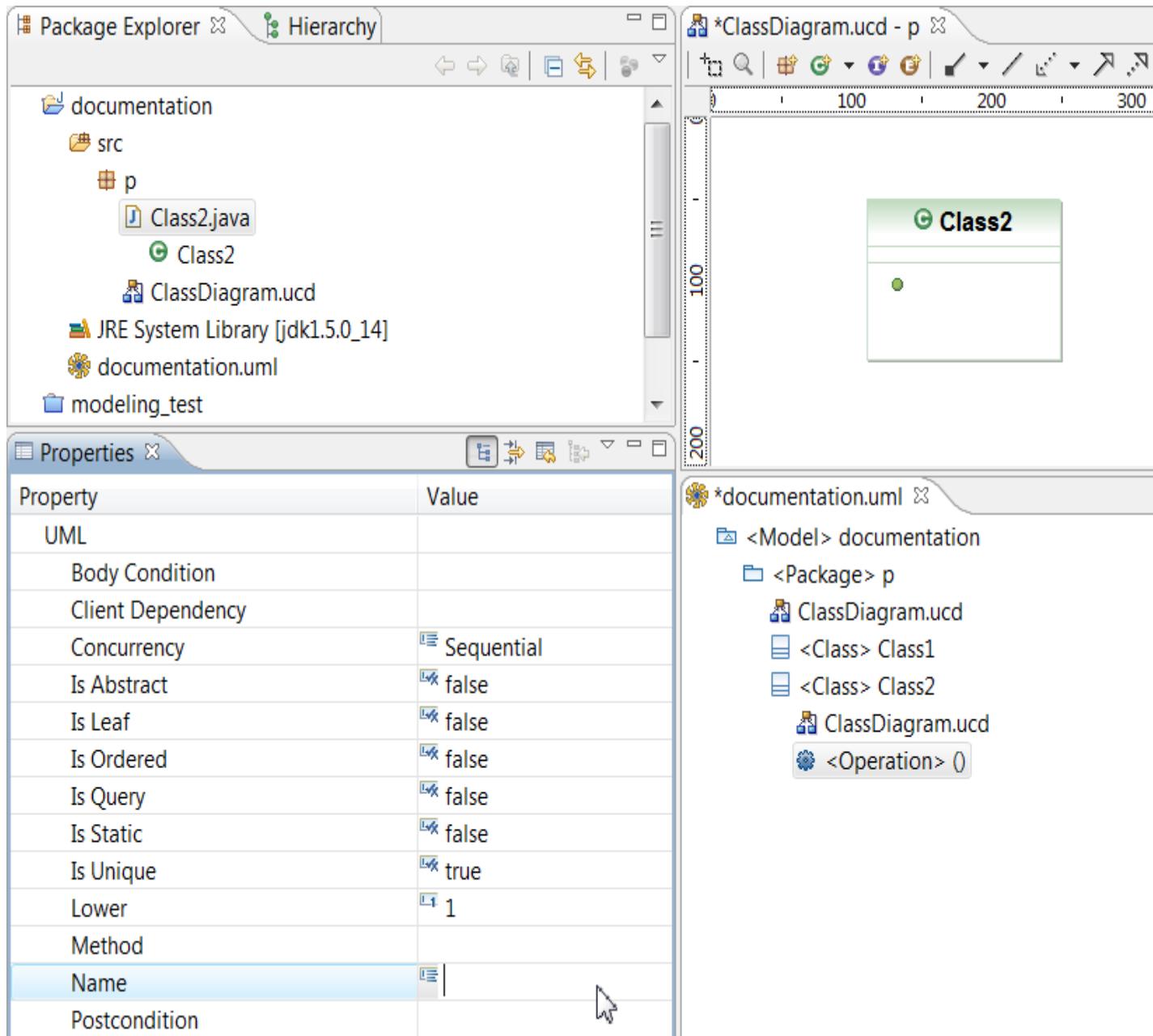
This is useful feature if you want to create the skeleton of your application and not the body. Select a Java Class in your XMI Editor > New Child > Owned Operation > Operation



A method has been created in your Class Diagram.

Note that this method has no name.

You can add a name using the Property view in order to keep it as a metamodel Operation, or click on the method in the class diagram and add a name using the Property menu in order to transform it to a Java Class.



2. New UML Metamodel Elements

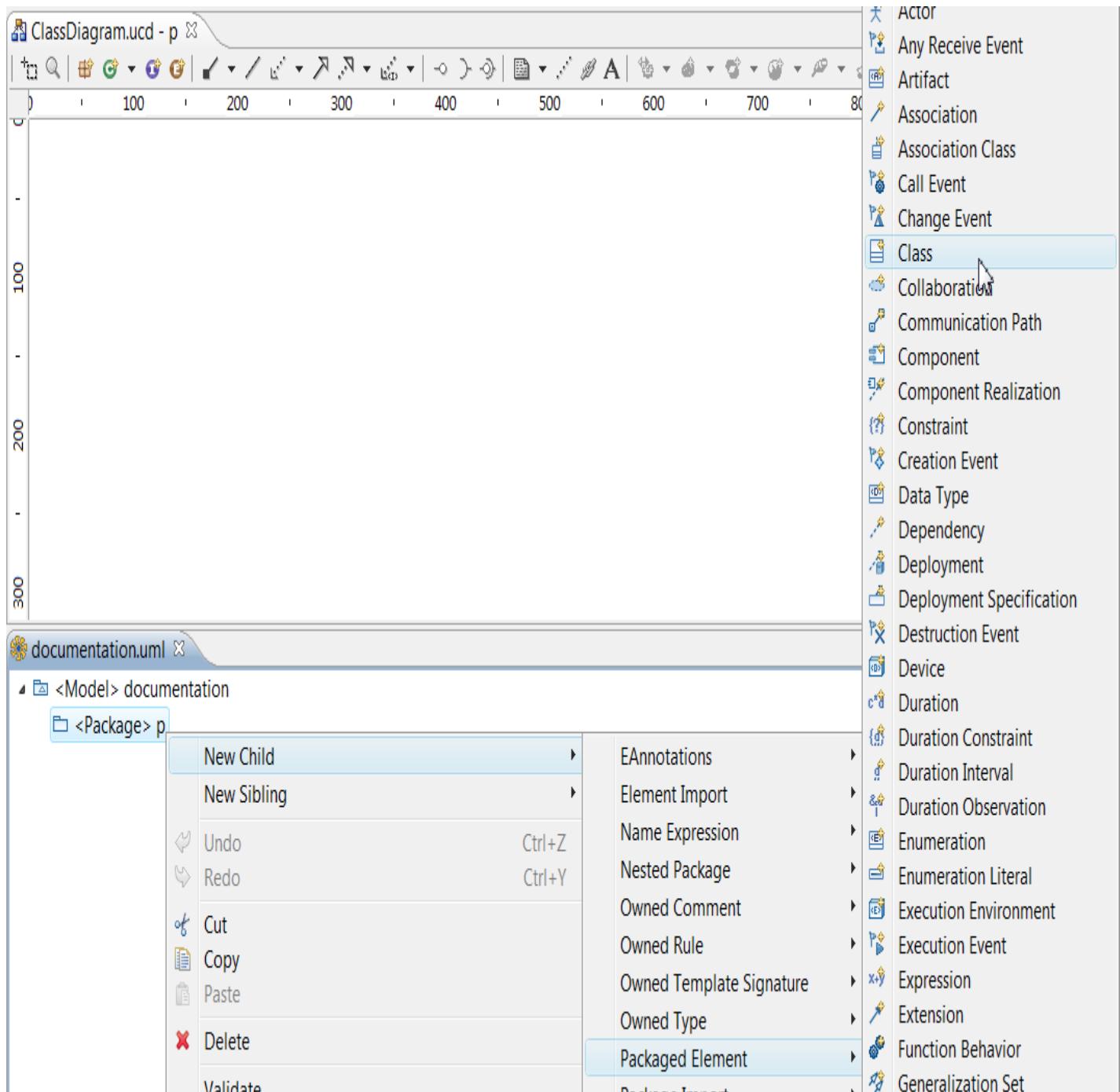
You can create metamodel elements which are not related to Java using the XMI 2.1 Editor drag and drop to the Class Diagram or the Class Diagram contextual menu.

From the Model Editor you can drag and drop the following UML (*just a metamodel element not related to Java*) elements:

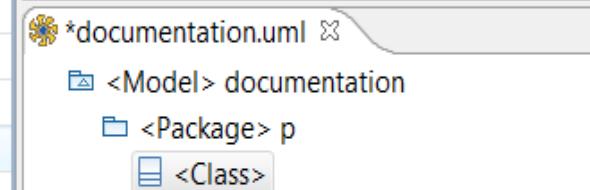
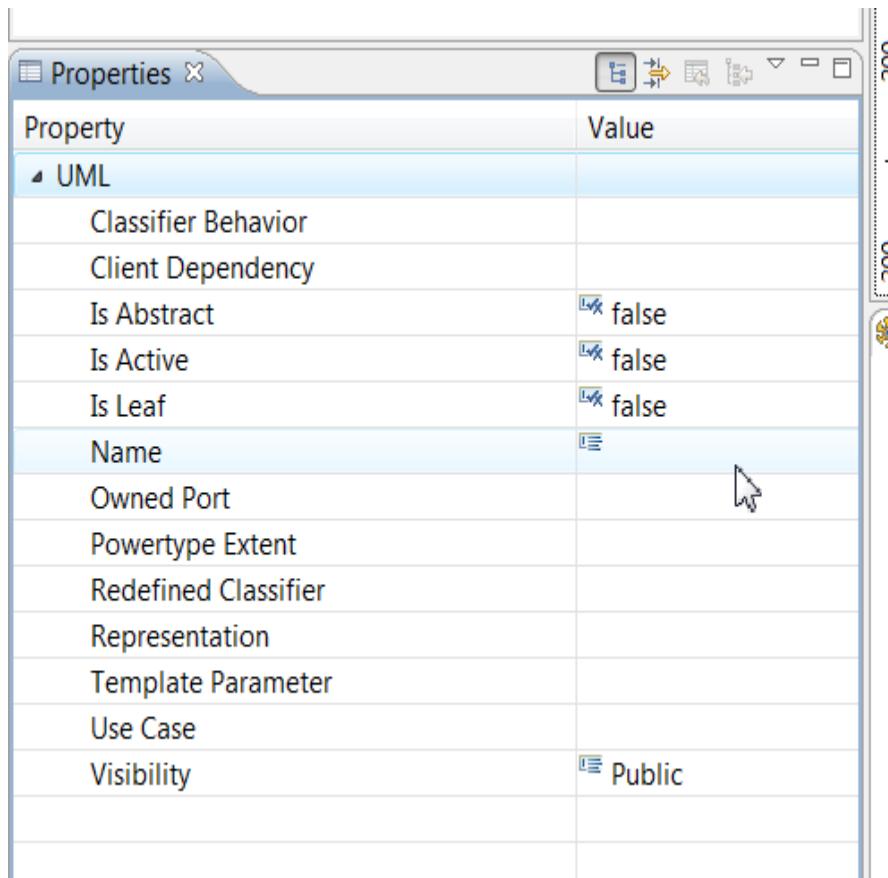
- Package
- Class
- Interface
- Enumeration

To create a new Metamodel Class in your XMI Editor you need to:

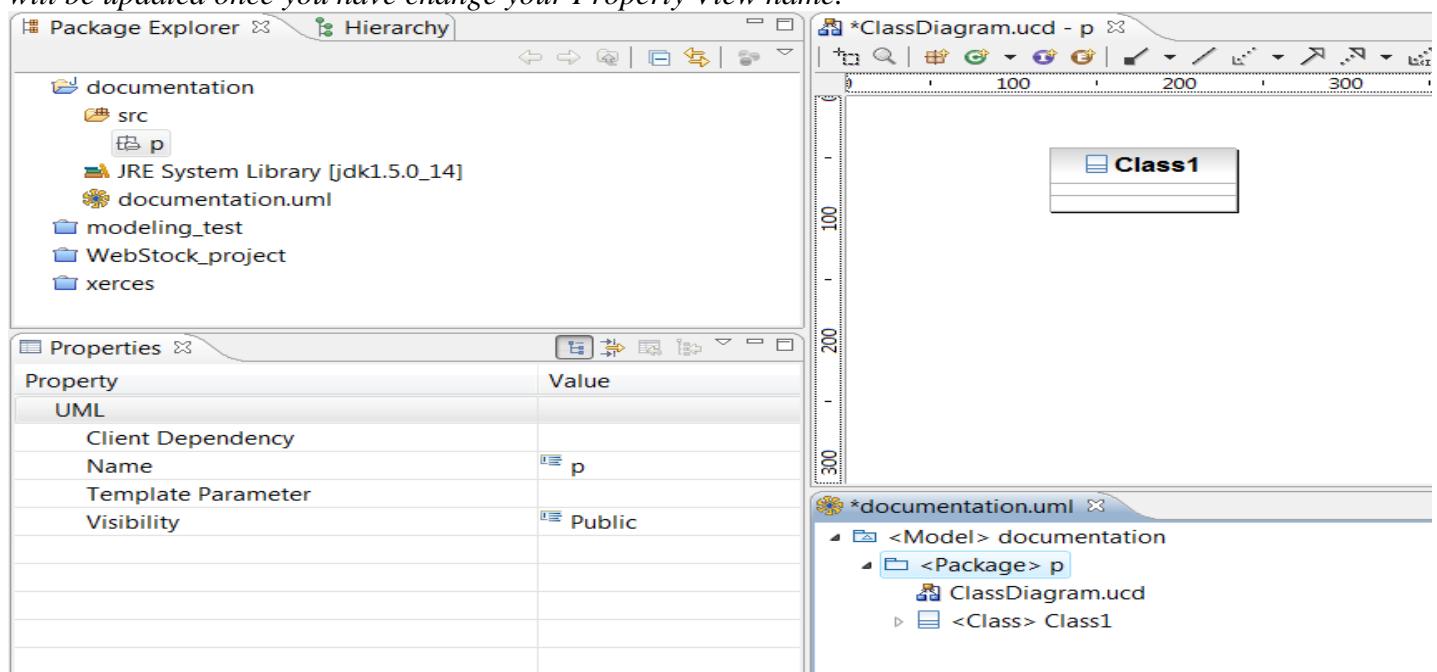
Click on the **package > New Child > Packaged Element > Class**



A none named class will be created in your XMI Editor representing your metamodel instance.



You need to drag and drop this Class to your UML Editor to create the metamodel class. No Java Code will be created. Note that the automatic Class diagram naming will add a modeling name to your class. When you have decided which name should be given, then you can change the name directly in the Property View. All previous diagrams including this renamed Class will be automatically renamed inside the full project. It means that if you have 10 diagrams having the same elements, all 10 diagrams will be updated once you have change your Property View name.



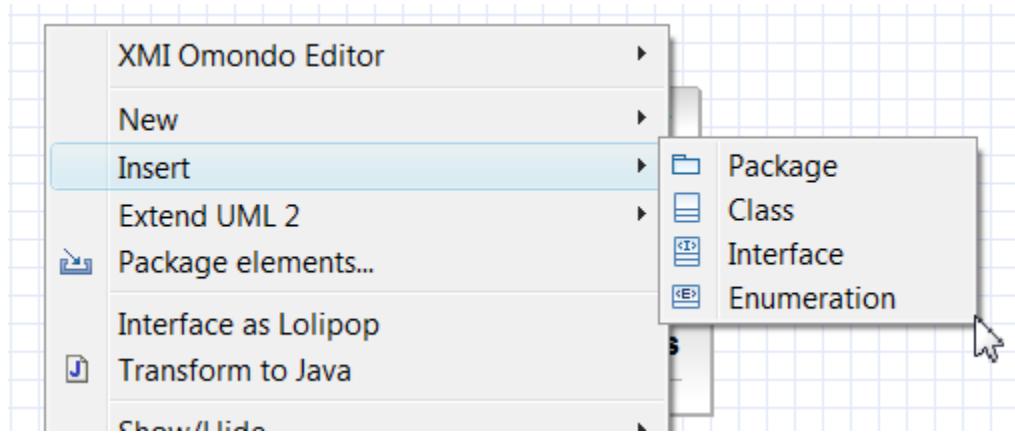
Insert

This section describes how to select existing Java elements which are in your project and add them to your diagram. *Please note that the drag and drop from the Package Explorer is doing the same job except that Insert can help you to select faster the needed classifier because of keyword completion.*

Click on the Class Diagram background to open the contextual menu > **Insert**

You can insert the following elements:

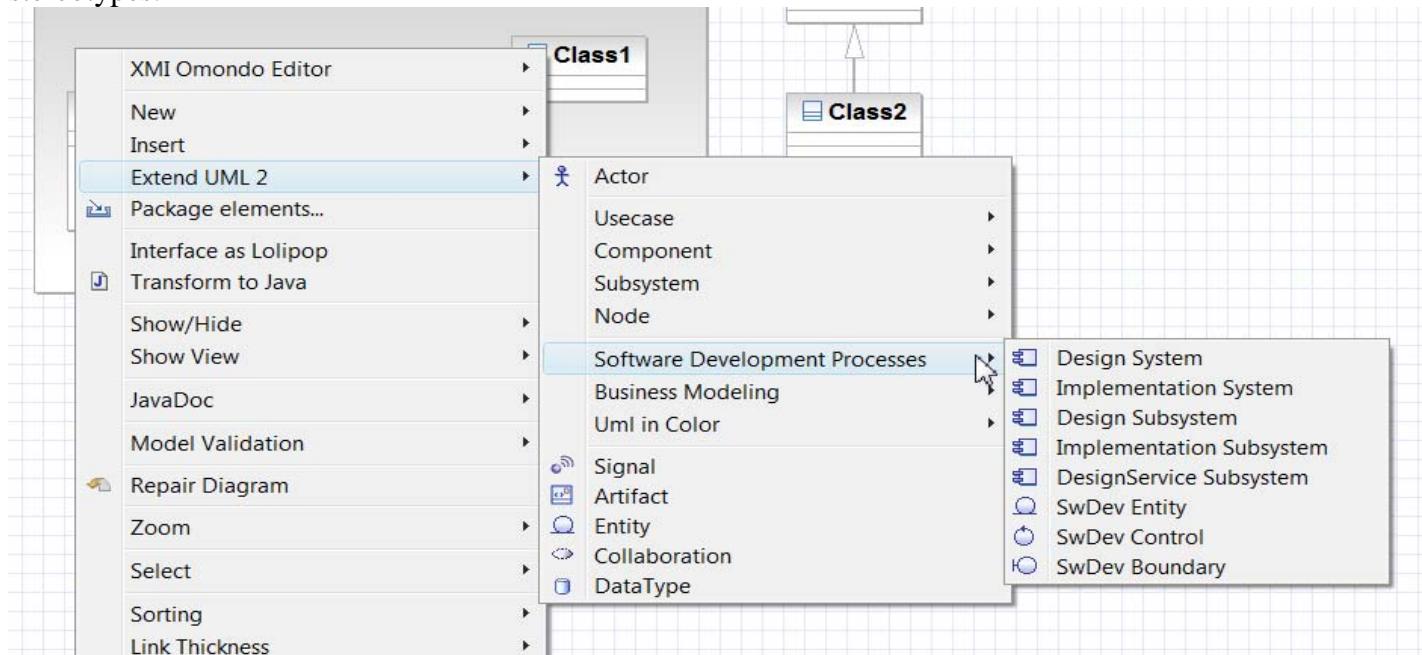
- Package
- Class
- Interface
- Enumeration



Extend UML 2

This section describes how to add new UML 2 elements.

This option allows adding new UML 2 elements inside your class diagram and use predefined business stereotypes.

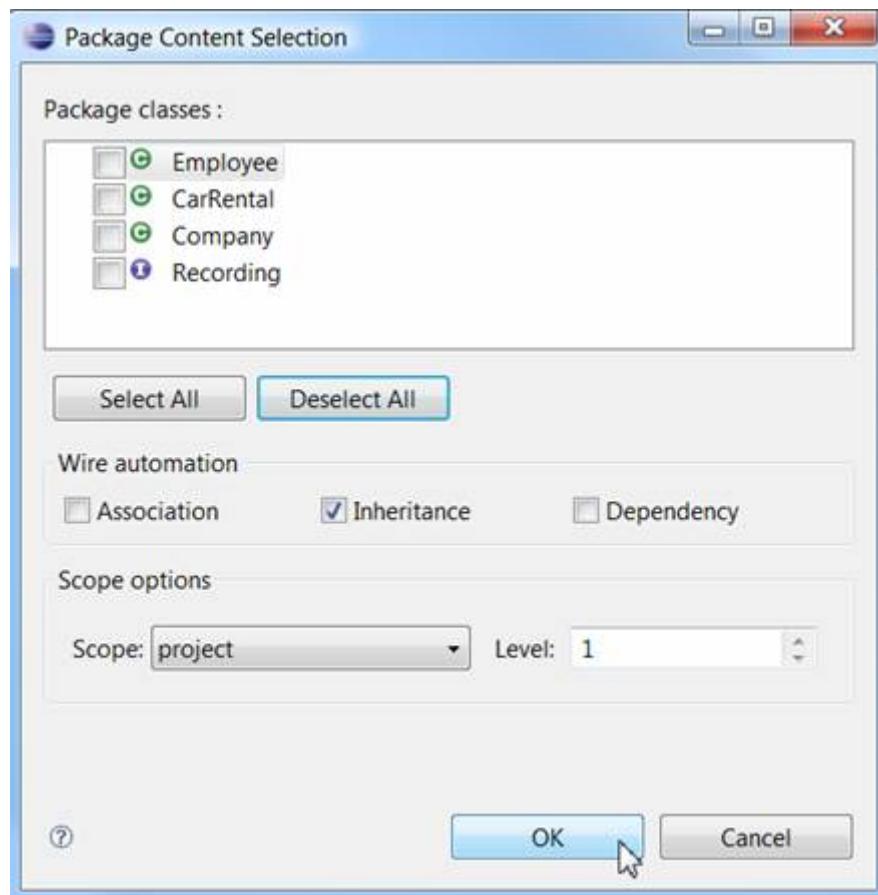


Package Elements

This section describes how to add existing project elements inside your class diagram. You can insert in your class diagram any classifiers of the package which is not currently displayed in the diagram.



Open the class diagram contextual menu > **Package elements**

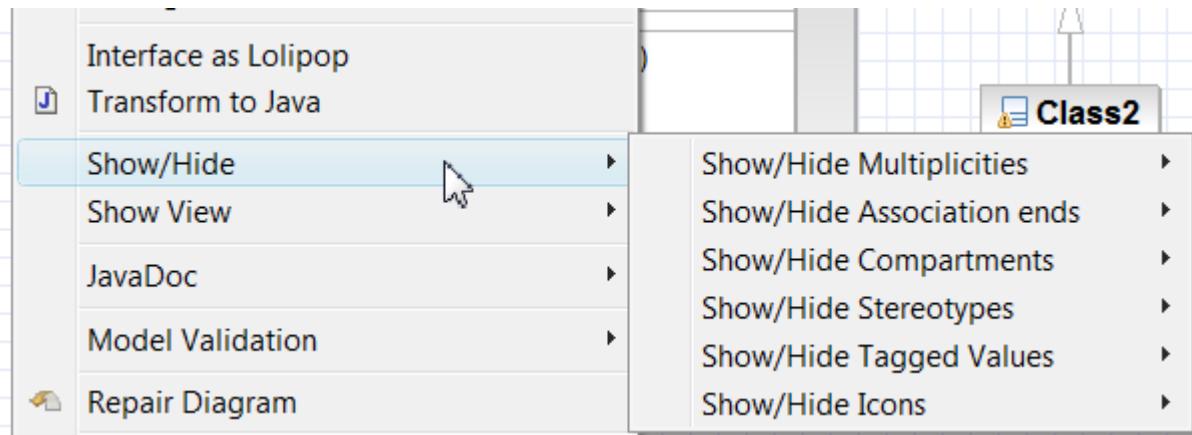


Show Hide

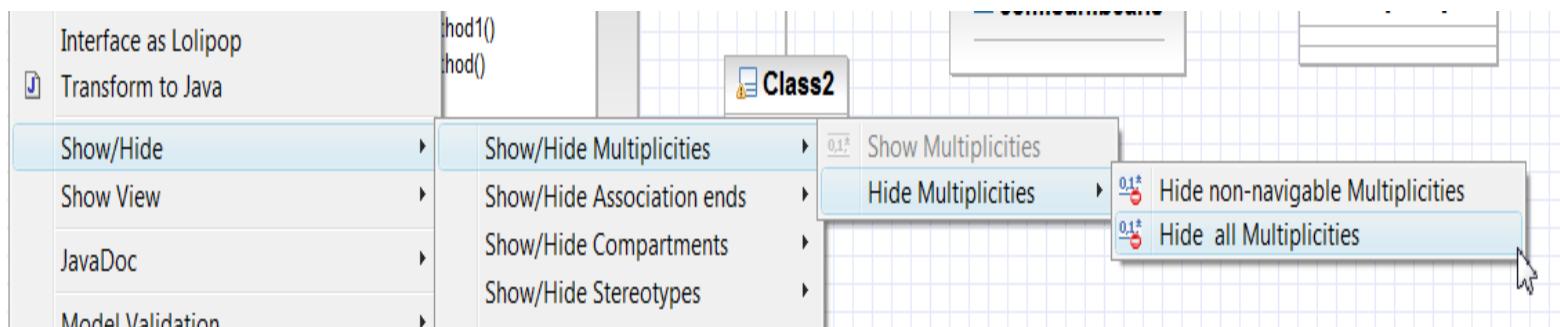
This section describes how to Show or to Hide:

- [Multiplicity](#)
- [Association ends](#)
- [Compartments](#)
- [Stereotypes](#)
- [Tagged Values](#)
- [Icons](#)

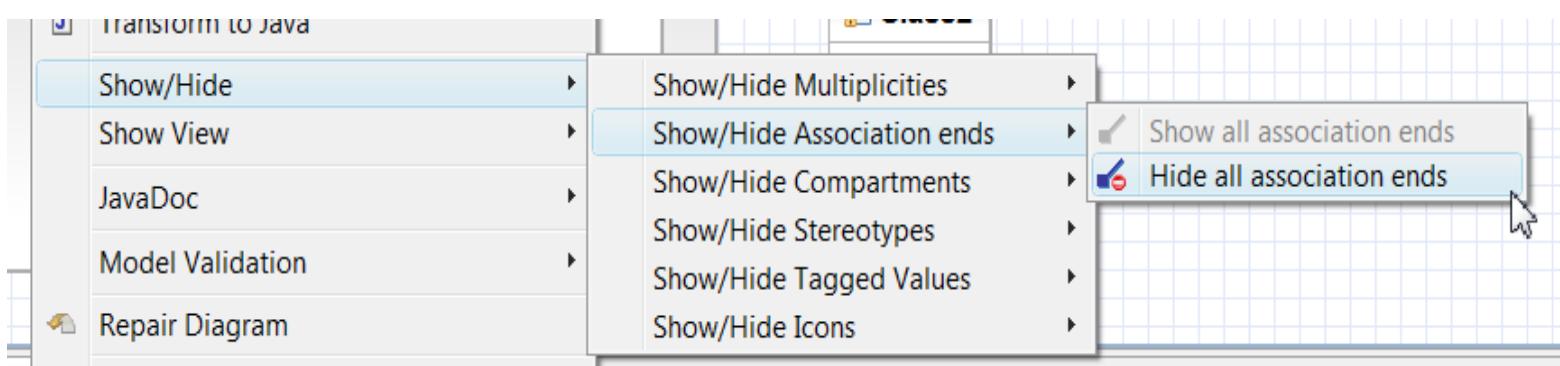
The Show Hide menu allows you to customize your additional UML elements inside your class diagram.



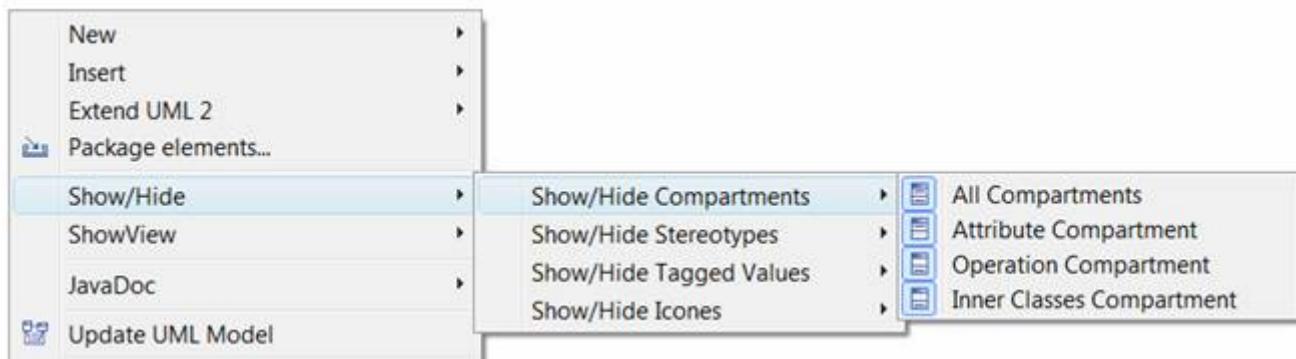
The Show/Hide Multiplicity allows you to show or hide association multiplicity on both or just one side.



The Show/Hide Association Ends allows you to show or hide association ends at diagram level.

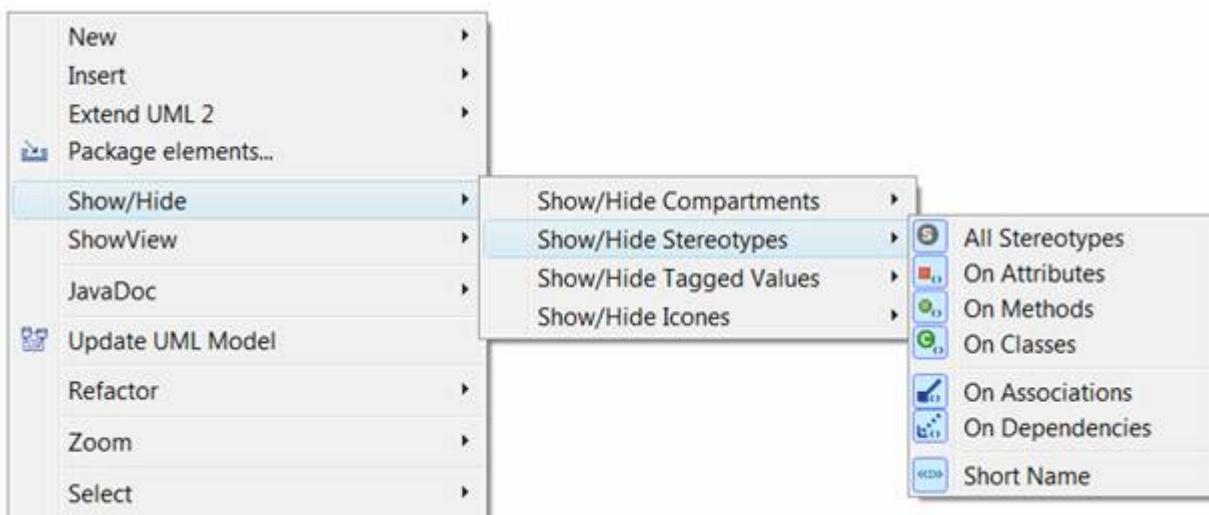


The Show/Hide Compartments allows you to show attributes, methods and inner elements compartments in the diagram.

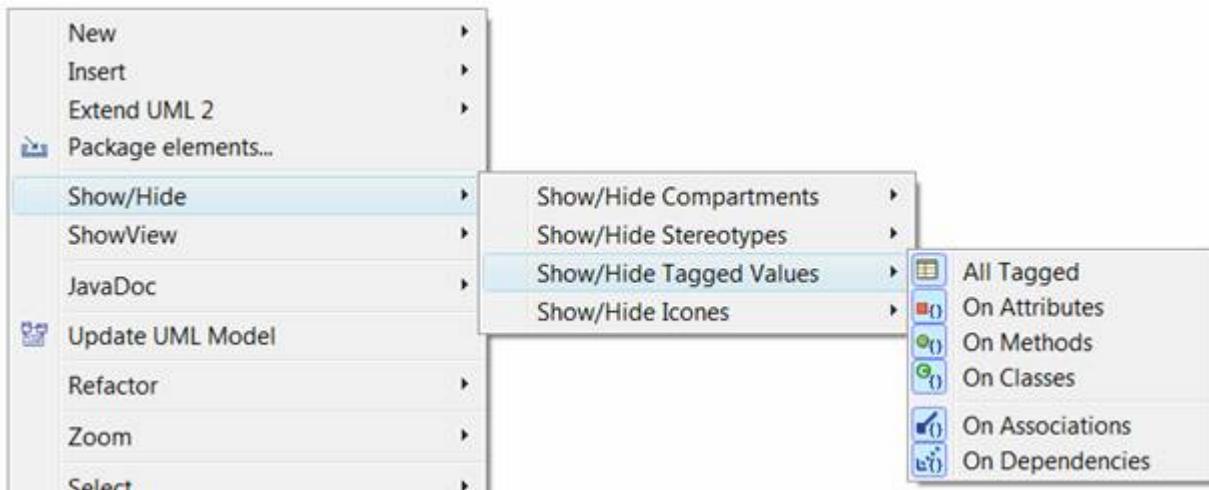


The Show/Hide Stereotype allows you to show or to hide Stereotypes on attributes, methods, classes, associations and dependencies.

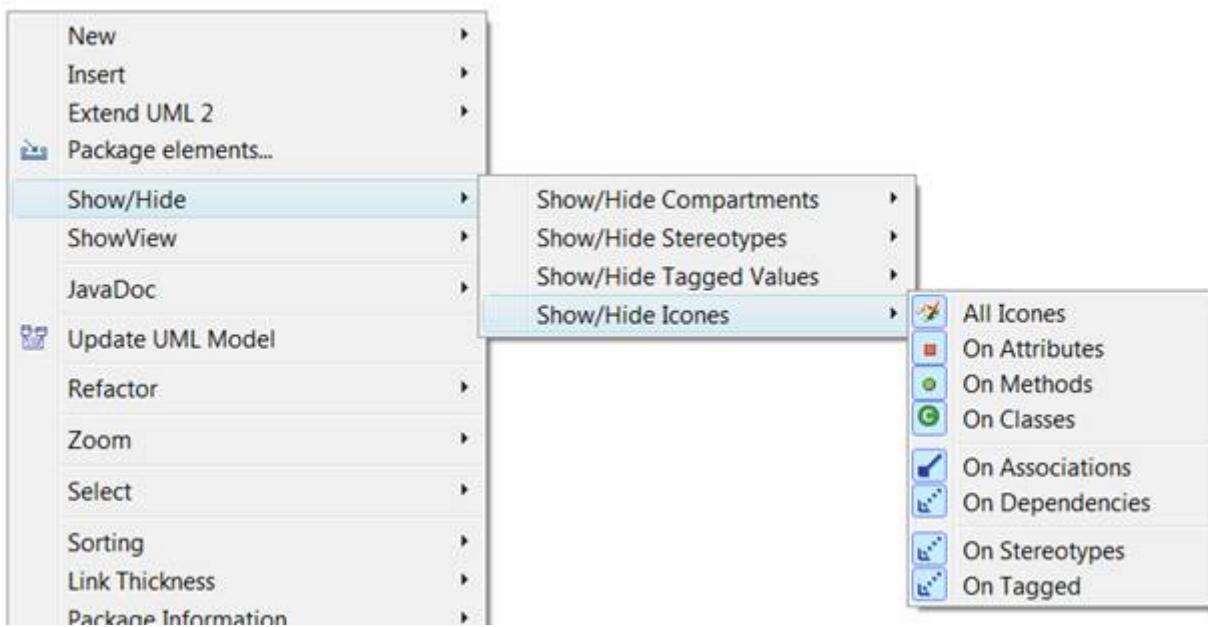
The Short Name option reduces the stereotype number of characters displayed on your diagram.



The Show/Hide Tagged Values allows you to show or to hide Tagged Values on attributes, methods, classes, associations and dependencies.



The **Show/Hide Icons** allows you to show or to hide Icons on attributes, methods, classes, associations dependencies, stereotypes and tagged values.



Show View

This section describes how to open additional modeling views directly from the Class diagram. Click on the Class diagram background to open the contextual menu **Show View >...**

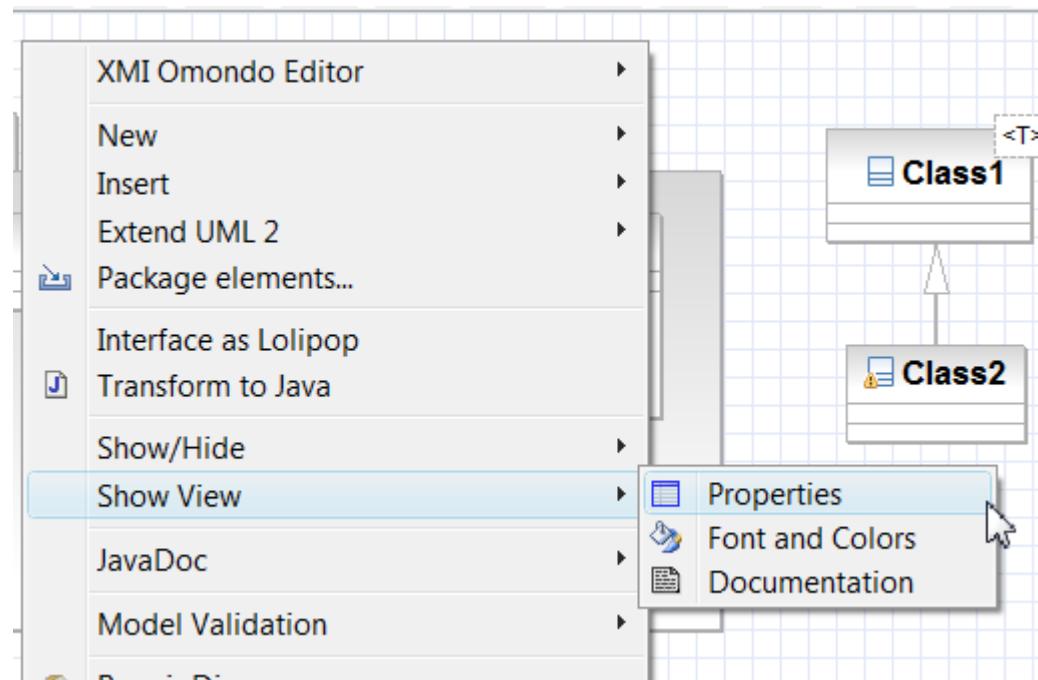
- **Properties** View is an Eclipse View synchronized with EclipseUML
- **Font and Colors** is an Omondo View
- **Documentation** is an Omondo View



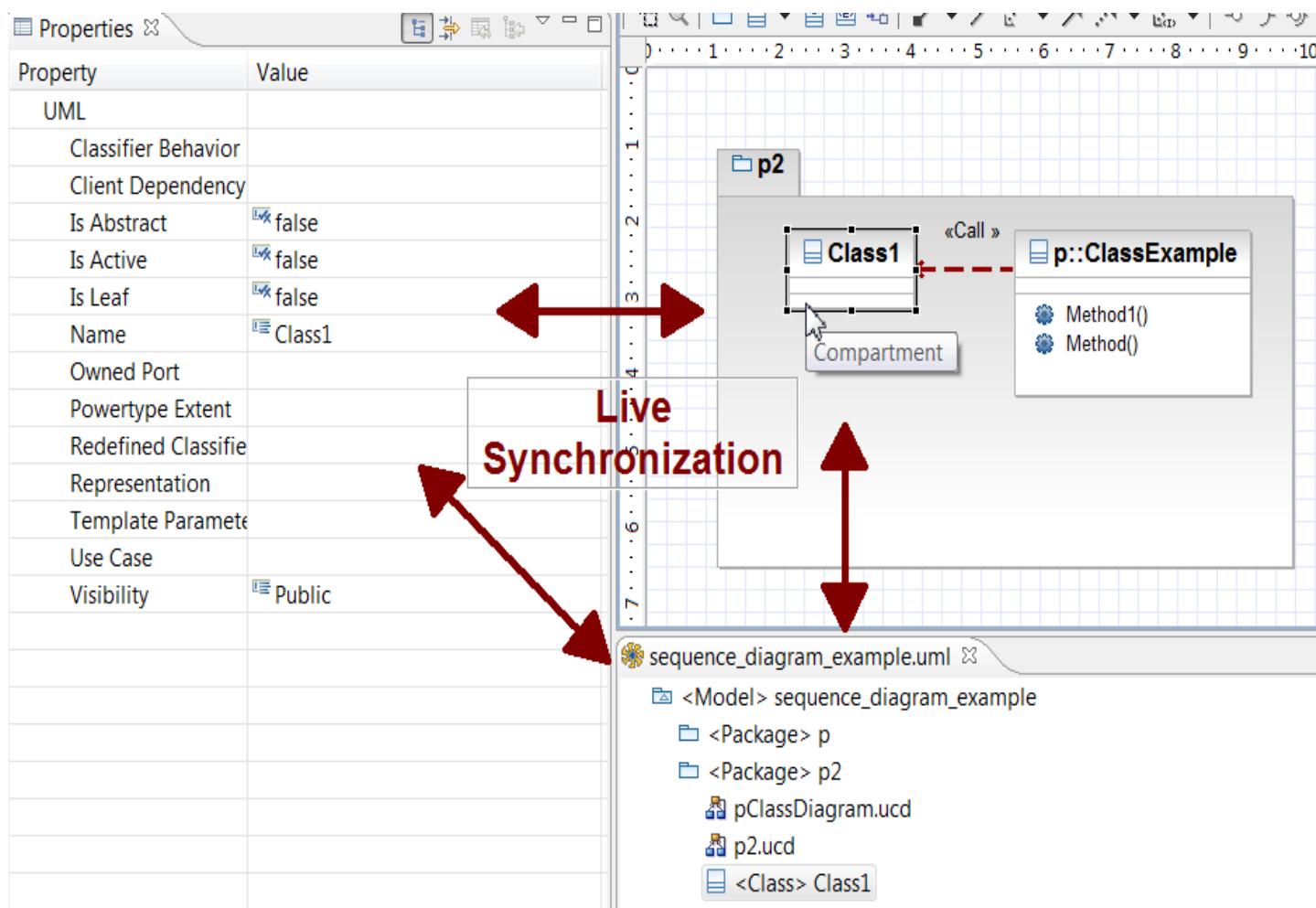
Properties View

The Properties View allows you to display UML Superstructure model property of each element. It is an Eclipse View which has been used by Omondo in the EclipseUML modeling Cycle.

To activate this view you can for example select the class diagram contextual menu > **Show View > Properties** or use Eclipse menus.



The Properties View is live synchronized with the UML Editor and the UML Superstructure model.



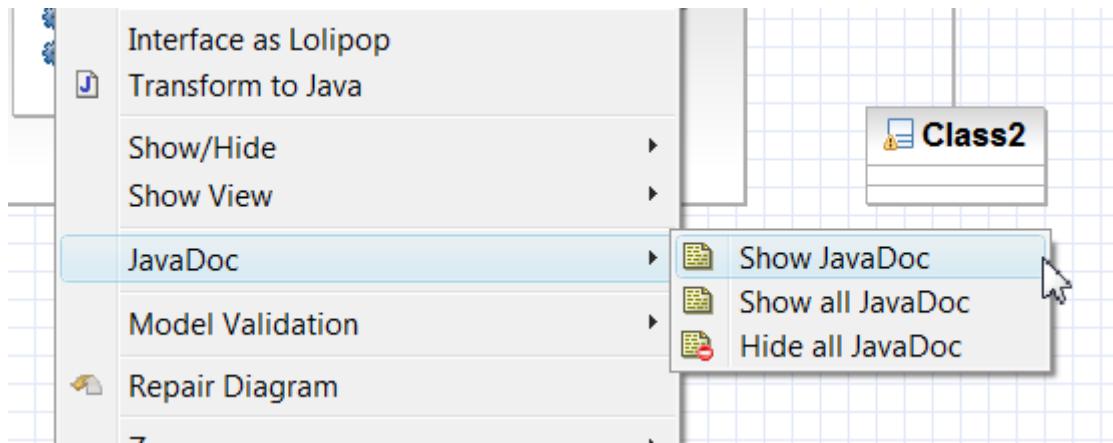
JavaDoc

This section describes how to add JavaDoc inside your class diagram. The JavaDoc will be displayed as a note and will only be a graphical information (e.g. no Javadoc is saved in the UML Superstructure model).

You can activate this option by selecting the class diagram contextual menu > **JavaDoc**

The following features are available:

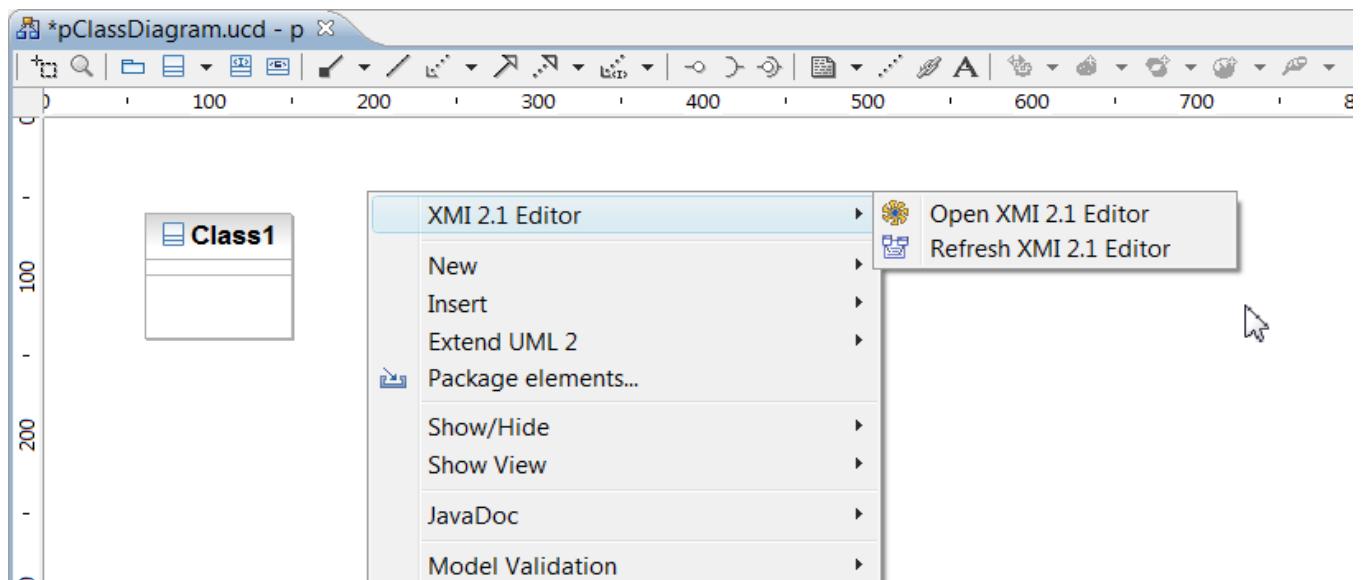
- **Show JavaDoc** will show the first two line of the JavaDoc
- **Show all JavaDoc** will show all the JavaDoc
- **Hide JavaDoc** will hide all Javadoc notes which are displayed in the diagram



Refresh XMI 2.1 Editor

This section describes how to refresh your XMI 2.1 Editor.

You can update your XMI 2.1 Editor at any time of your modelling process by opening the class diagram contextual menu **XMI 2.1 Editor** > **Refresh XMI 2.1 Editor**.

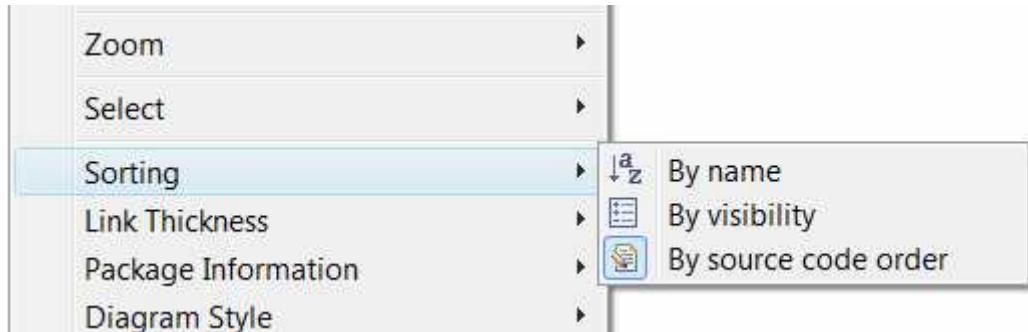


Sorting

This section describes how to organize the display order.

You can select the attributes and method display order by:

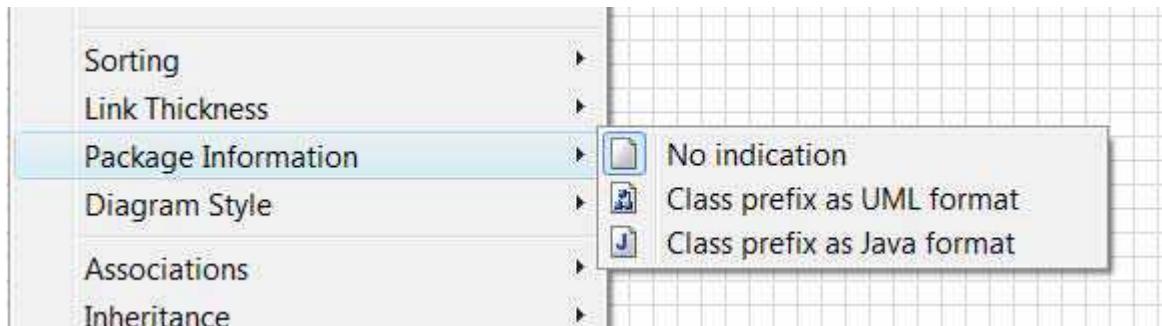
- Name
- Visibility
- Source code order



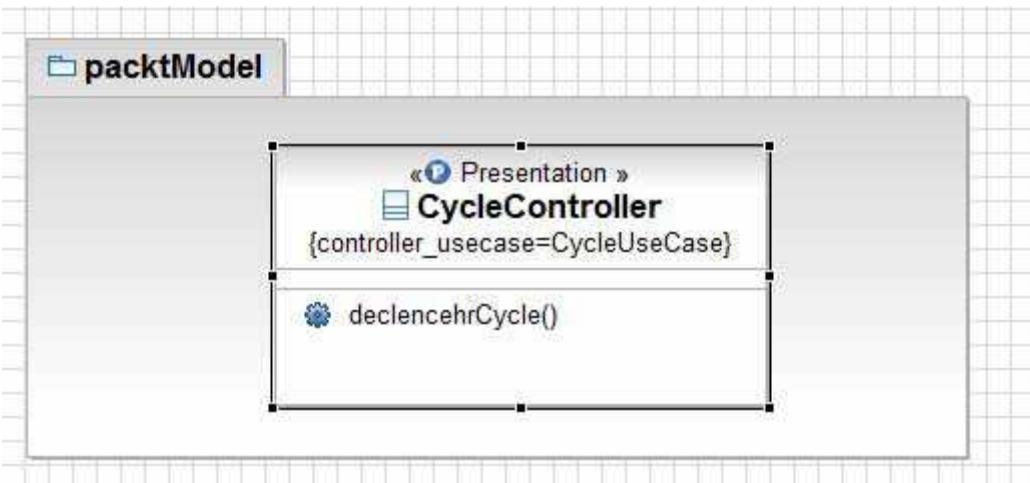
Package Information

This section describes how to display the package information on a class, interface or enumeration. You can select the kind of package information that you want to be displayed.

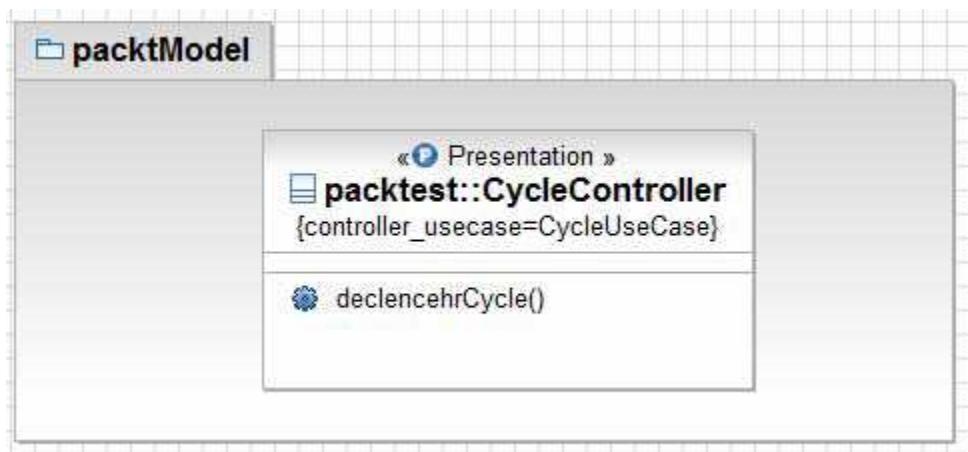
Select the Class Diagram contextual menu > **Package Information**



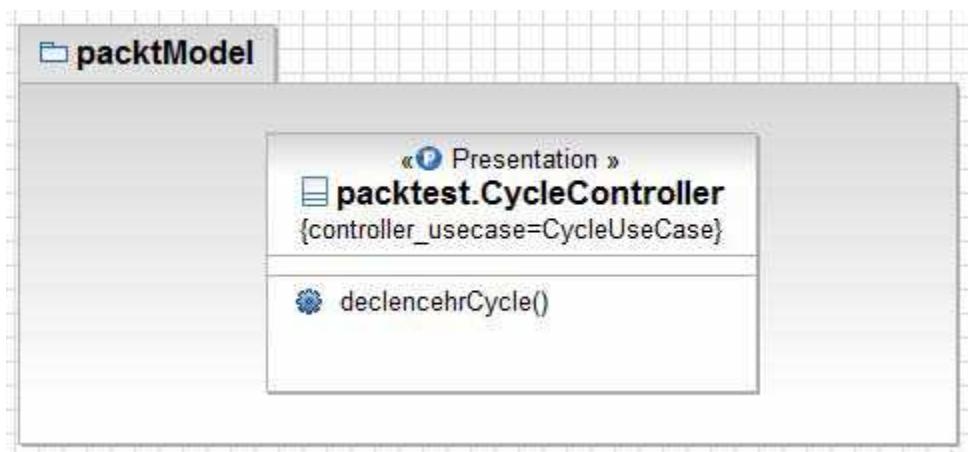
No indication:



UML format:



Java format:

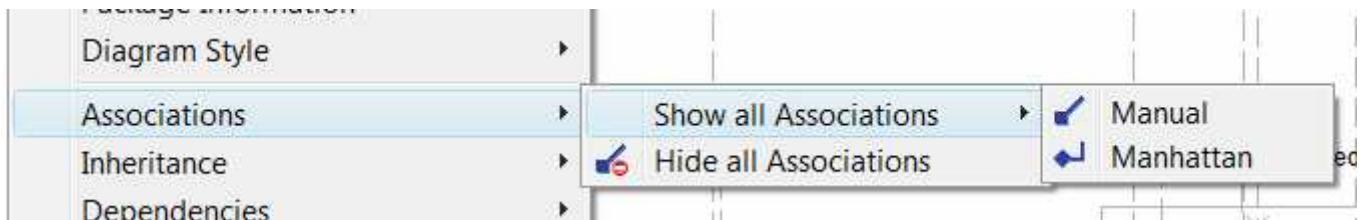


Show Hide Links

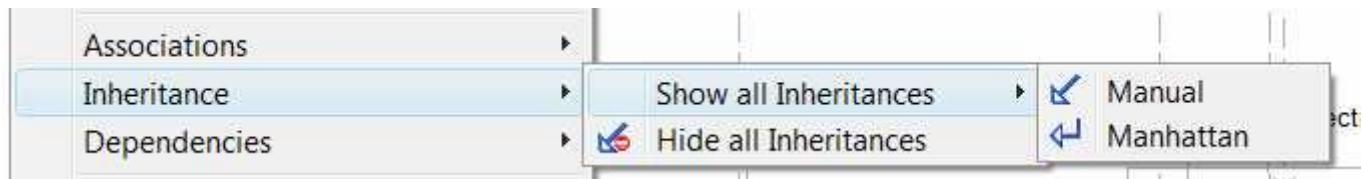
This section describes how to show hide links. This feature is related to the [dynamic navigation concept](#).

You can display different views of your diagram by selecting **the class diagram contextual menu > Associations, Inheritances or Dependencies**.

The class diagram Association contextual menu allows you to show or to hide association from your diagram.



The class diagram Inheritance contextual menu allows you to show or to hide Inheritance from your diagram.



The class diagram Dependencies contextual menu allows you to show or to hide Dependencies from your diagram.



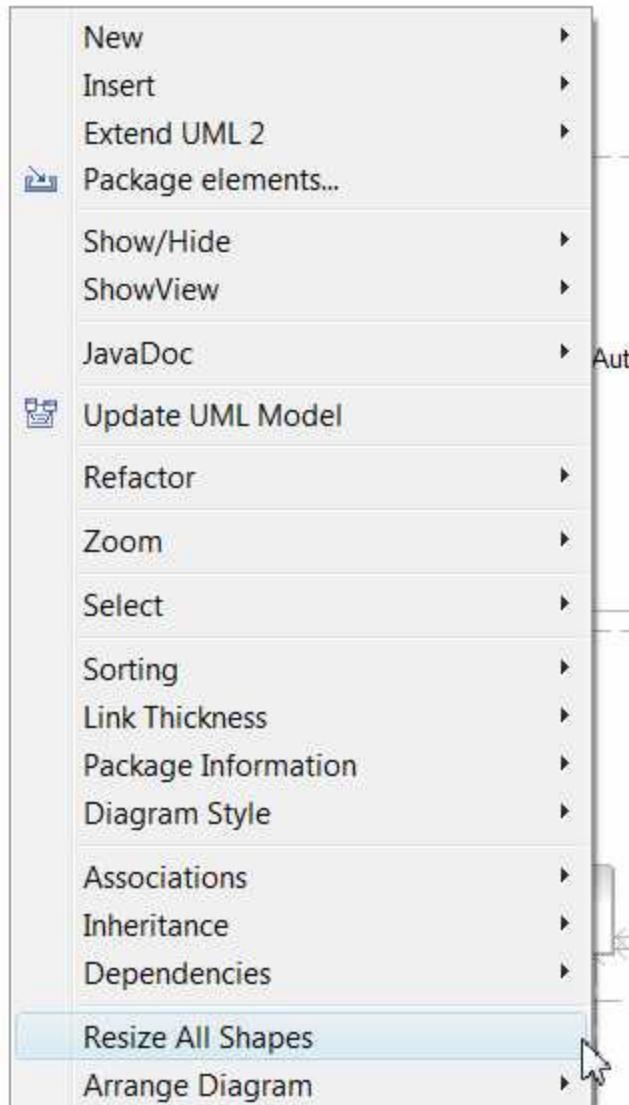
Resize All Shapes

This section describes how to resize all shapes.

Omundo will not launch any layout without your permission, because this could destroy current users UML positioning work.

We therefore prefer to let our user resize what they consider to be important, or to keep the same size for modeling reasons.

The Resize All Shapes menu minimizes the size of elements to the minimum needed size. This size to be displayed corresponds to characters width included inside the classifier name.



Arrange Diagram

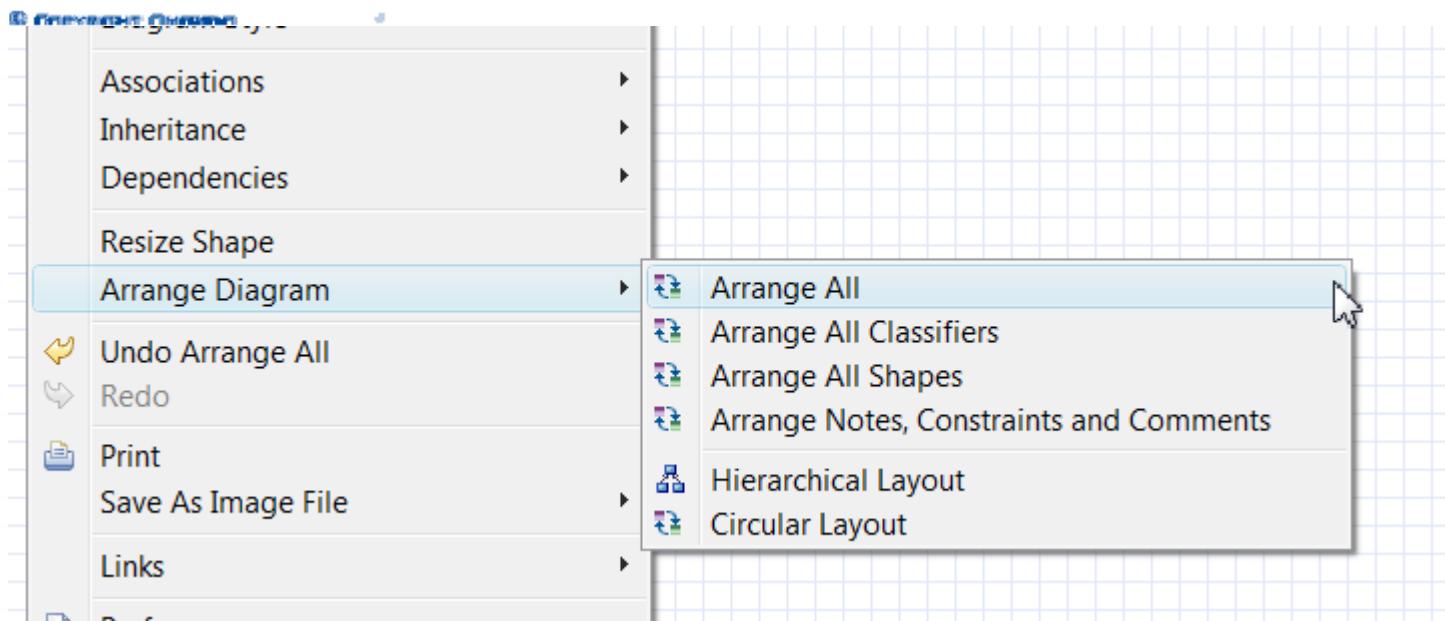
This section describes how to arrange all shapes.

The Arrange Diagram menu is available in the class diagram contextual menu.

You can display the class diagram elements in an arithmetic order.

The following options are possible:

- **Arrange All** will arrange all diagram elements
- **Arrange All Classifiers** will only arrange classifiers and their connectors
- **Arrange All Shapes** will arrange all shapes but will not arrange classifiers inside a package
- **Arrange Notes** will only arrange Notes, Constraints and Comments
- **Hierarchical Layout** will arrange all diagram elements in a hierarchical order
- **Circular Layout** will arrange all diagram elements in a circular order

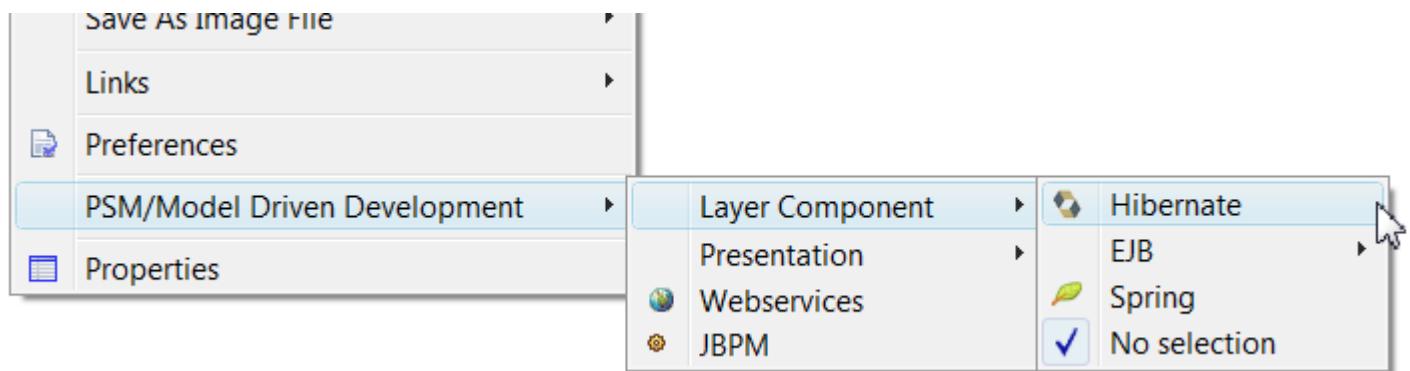


PSM - Model Driven Development activation

This section describes how to activate JEE elements inside your class diagram.

Please note that JPA is always activated because of specific other plugins need such as [Oracle Enterprise Pack for Eclipse 11g](#)

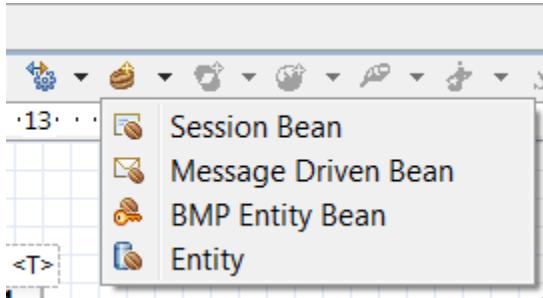
The Platform Specific Model Preferences menu is available in the class diagram contextual menu
PSM / Model Driven Development > ...



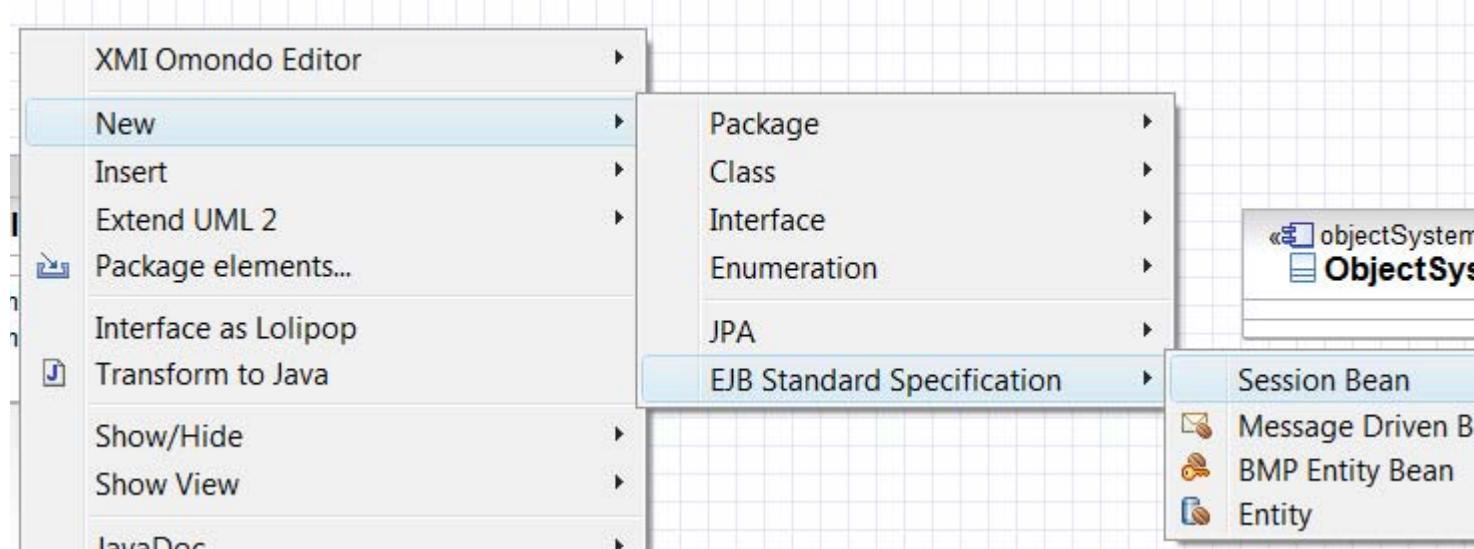
Selecting a layer Component allows activating PSM toolbar and contextual menus.

For example, if you select EJB Standard Specification, then to activate:

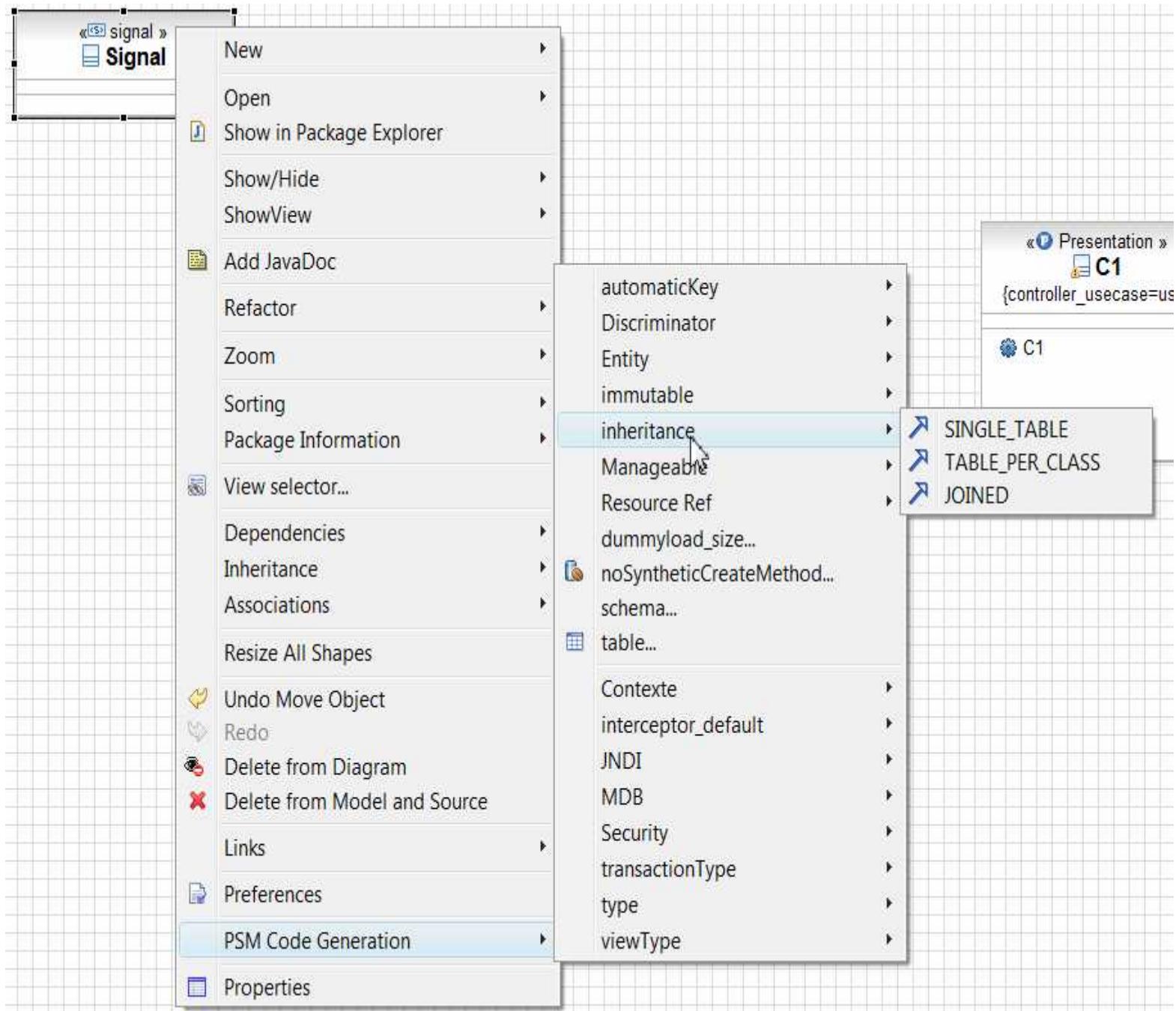
- The toolbar will be activated



- The class diagram contextual menu New > EJB Standard Specification is activated



- The class contextual menu



Presentation

When a UML Class Diagram is created, the diagram presentation mode can be selected. The diagram presentation mode allows you to focus on what it is important to show inside the new Class Diagram.

Three kinds of elements are provided :

1. Association
2. Inheritance
3. Dependency

We recommend selecting just one element at a time in order to create a better presentation. All elements could be selected at once if needed; this is mostly useful for a simple class diagram presentation.

Having too much information in the diagram is not recommended.

Navigation

Different navigations are possible using EclipseUML.

You can either decide to use Dynamic or Static navigation.

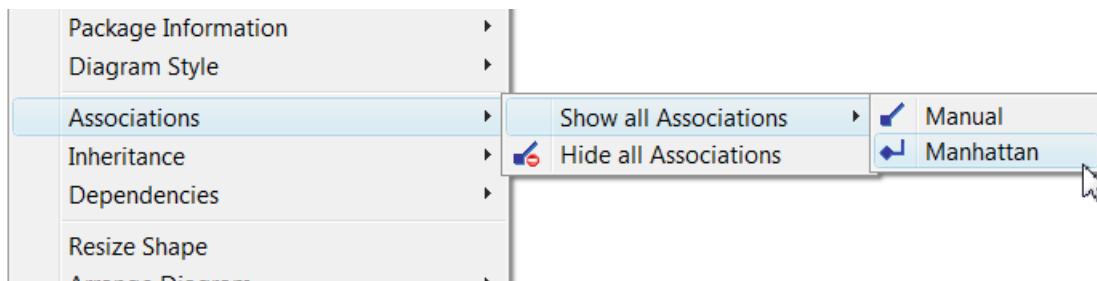
1. [Dynamique navigation](#) will use the same Class Diagram as a viewer of the UML Superstructure Model and will refresh the UML Editor after each model query.
2. [Static navigation](#) will create a new diagram for each view.

We recommend using both dynamic and static navigations inside the same project because each navigation concept is used at different modeling stages.

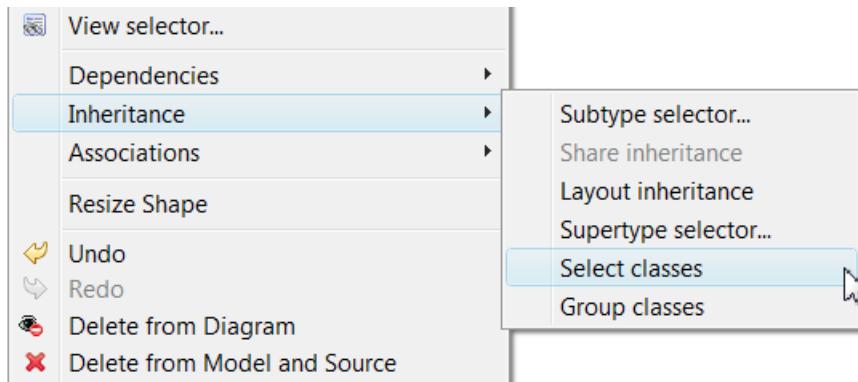
1. Dynamic Navigation

Dynamic navigation is available using:

- the **Class Diagram menu Association, Inheritance or Dependencies.**



- the **classifier contextual menu Dependencies, Inheritance and Associations.**



See a dynamic navigation example at: http://www.forum-omondo.com/documentation_eclipseuml_2008/eclipseuml2008_dynamic_navigation.html

2. Static Navigation

Static navigation is available using:

- [1. Navigation from a Class](#)
 - [1.1 To a Class-Centric Diagram](#)
 - [1.2 To a State Diagram](#)
 - [1.3 To a Sequence or Collaboration Diagram](#)
- [2. Navigation from a package](#)
 - [2.1 To a Package-Centric diagram](#)
 - [2.2 To a Class Diagram](#)

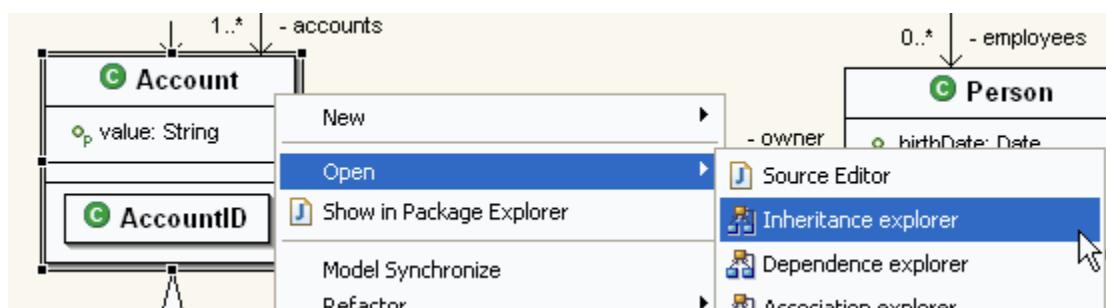
1. Navigation from a Class

1.1 To a Class-Centric Diagram

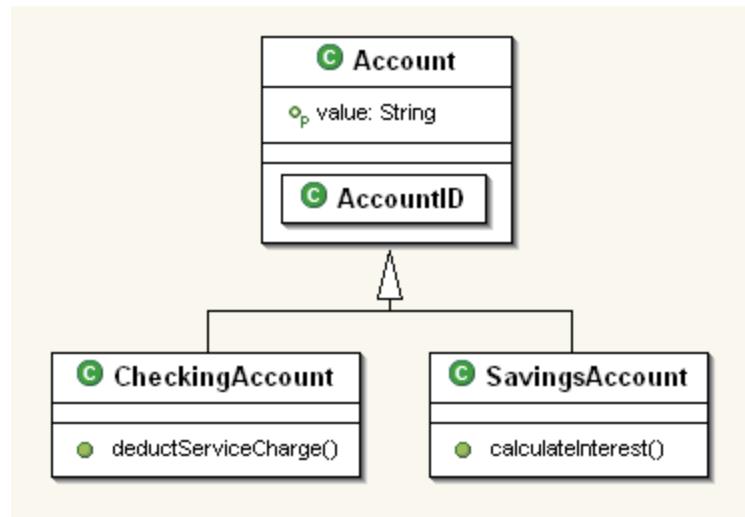
Navigation from a class to a class-centric diagram is available with 4 options:

- **Inheritance default explorer**

Select a class or an interface, open the popup menu and select **Open>Inheritance explorer**

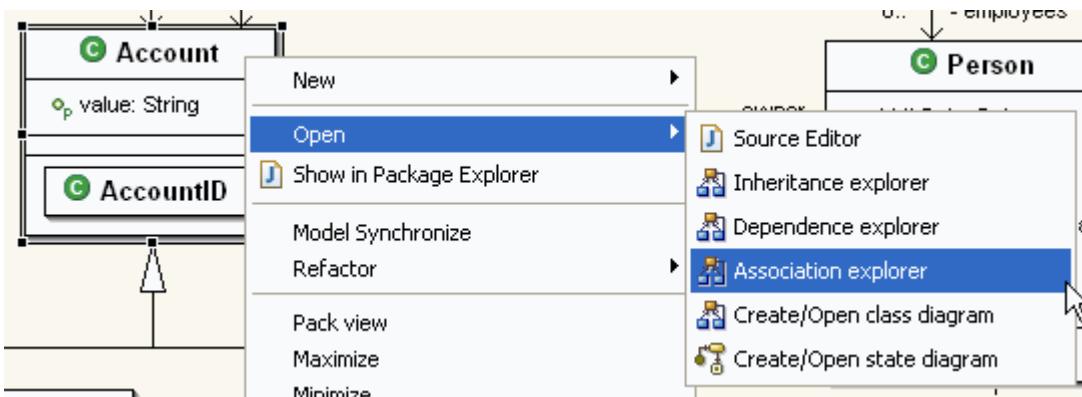


The default Inheritance explorer diagram display the immediate (level 1) inheritance environment of the selected class or interface.

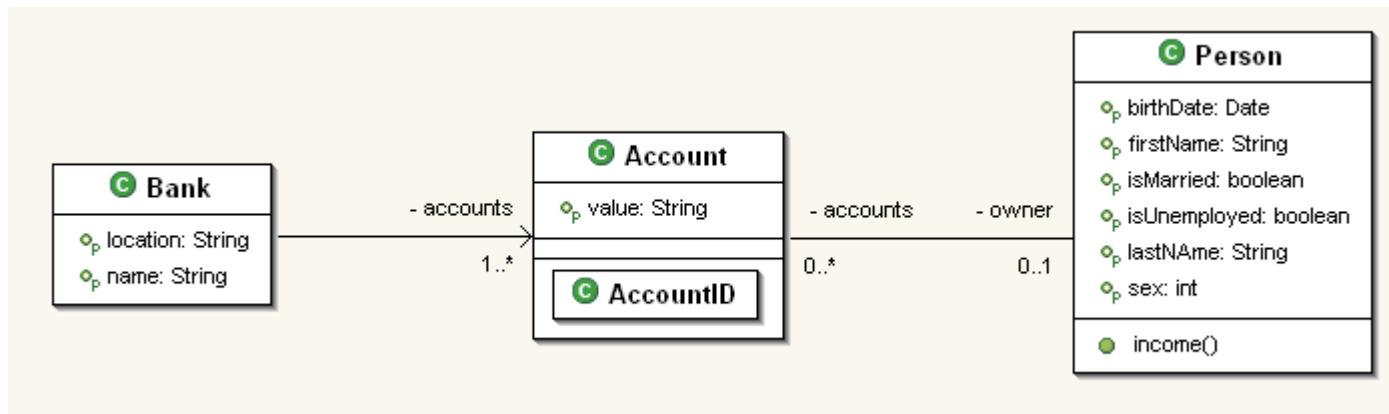


- **Association default explorer**

Select a class or an interface, open the popup menu and select **Open>Association explorer**

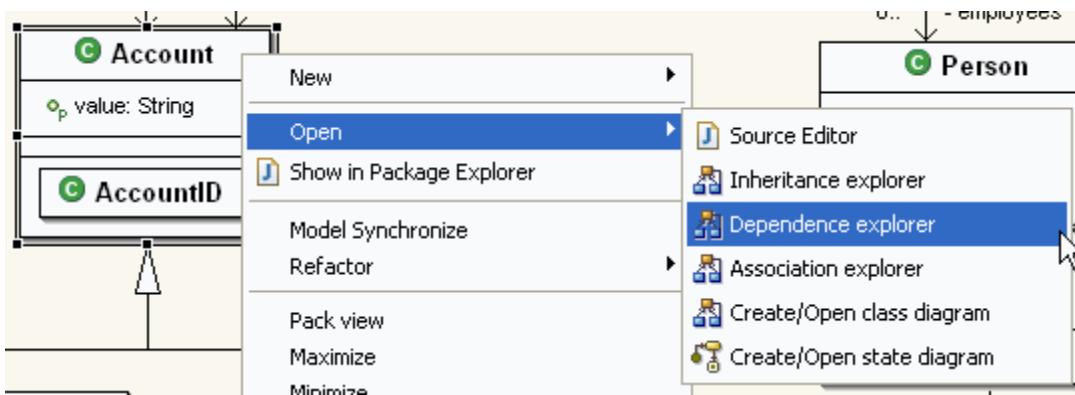


The default Association explorer diagram display the immediate (level 1) association's environment of the selected class or interface.

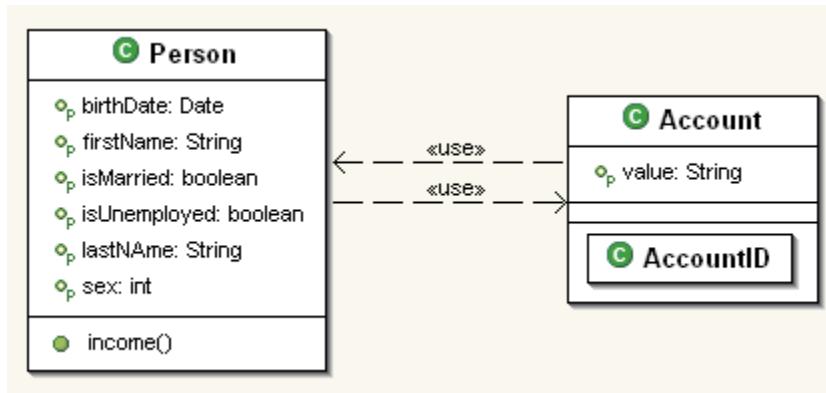


- **Dependency default explorer**

Select a class or an interface, open the popup menu and select **Open>Dependency explorer**

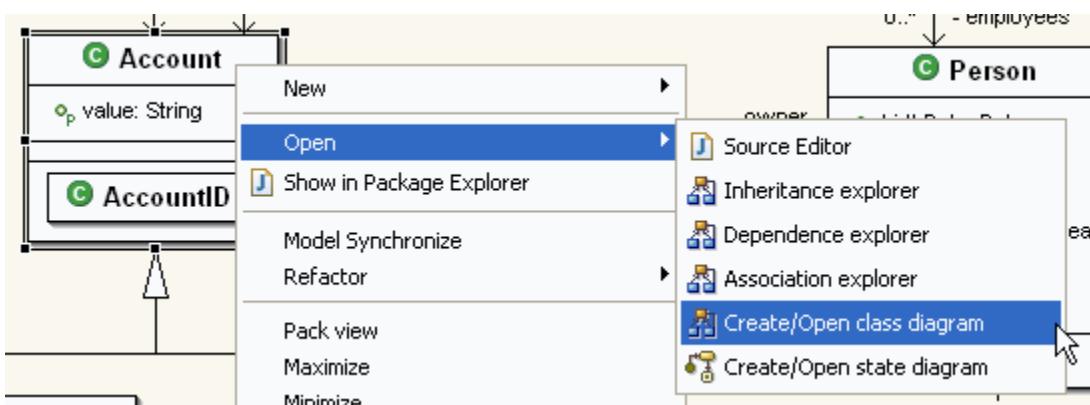


The default Dependency explorer diagram display the immediate (level 1) dependencies environment's of the selected class or interface.



- **Customizable Class-centric Diagram**

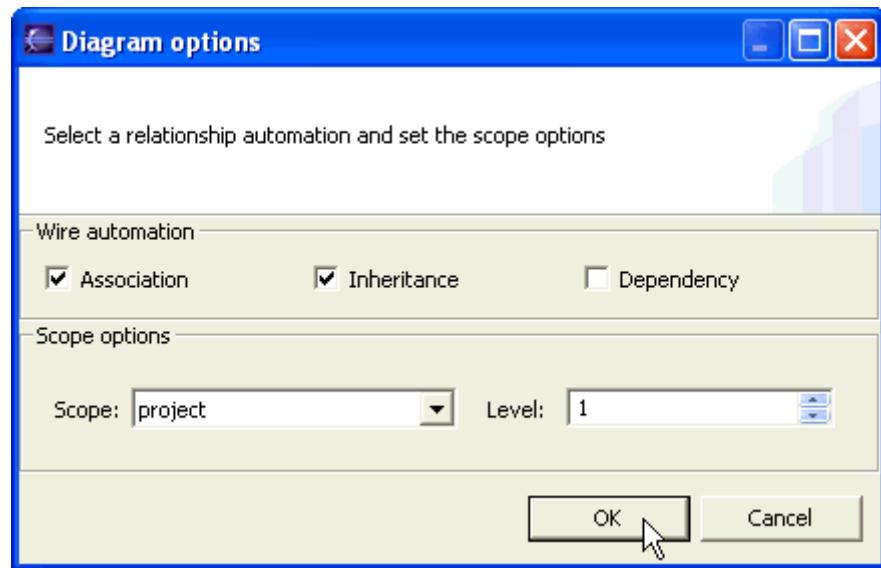
Open new class diagram and navigate within a Scope (package, project, all) and from -1 (infinite) to n levels. Select a class or an interface, open the popup menu and select **Open>Create/open class diagram**



Associations, interfaces and/or dependencies can be displayed in the diagram through the **Wire automation** option.

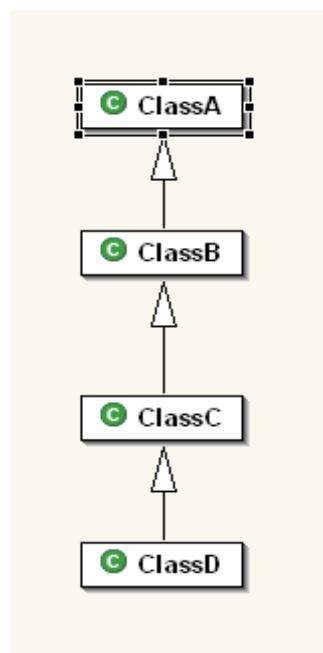
The diagram scope can be selected through the **Scope** option. Three options are available :

- package
- project
- all

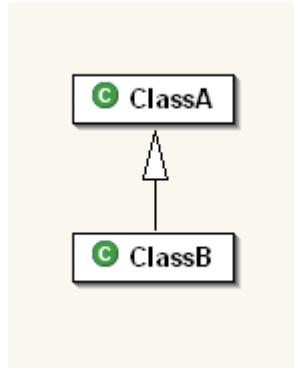


The diagram level can be selected through the **Level** option.

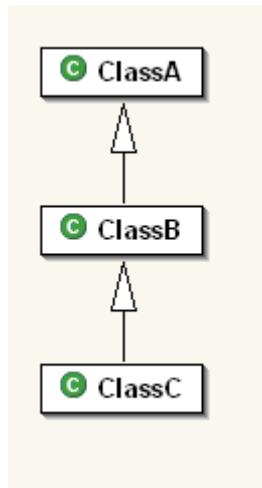
Here are some examples: Level -1 will analyze at infinite level and will then show all classes (Class A....Class D)



- 0 will analyze at the class level and will show Class A
- 1 will analyze at one level and will show ClassA and ClassB



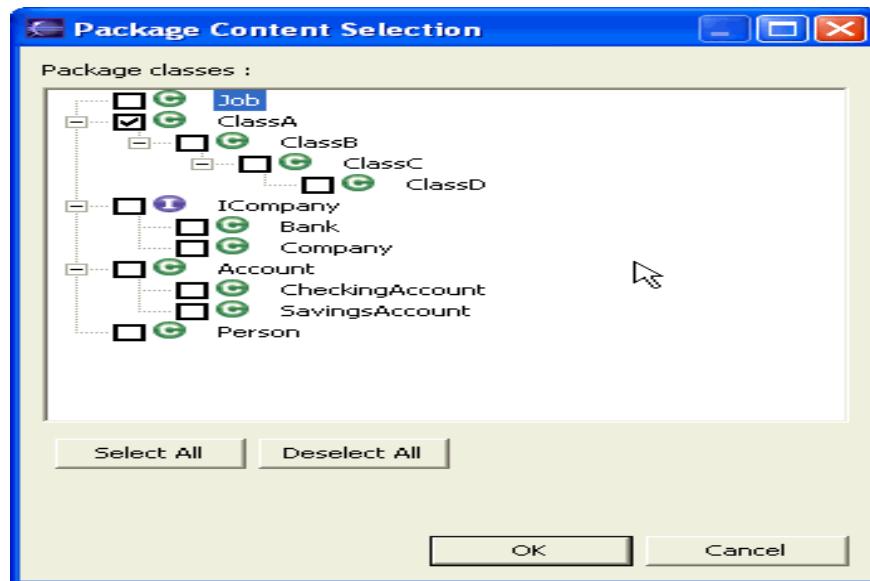
- 2 will analyze at depth 2



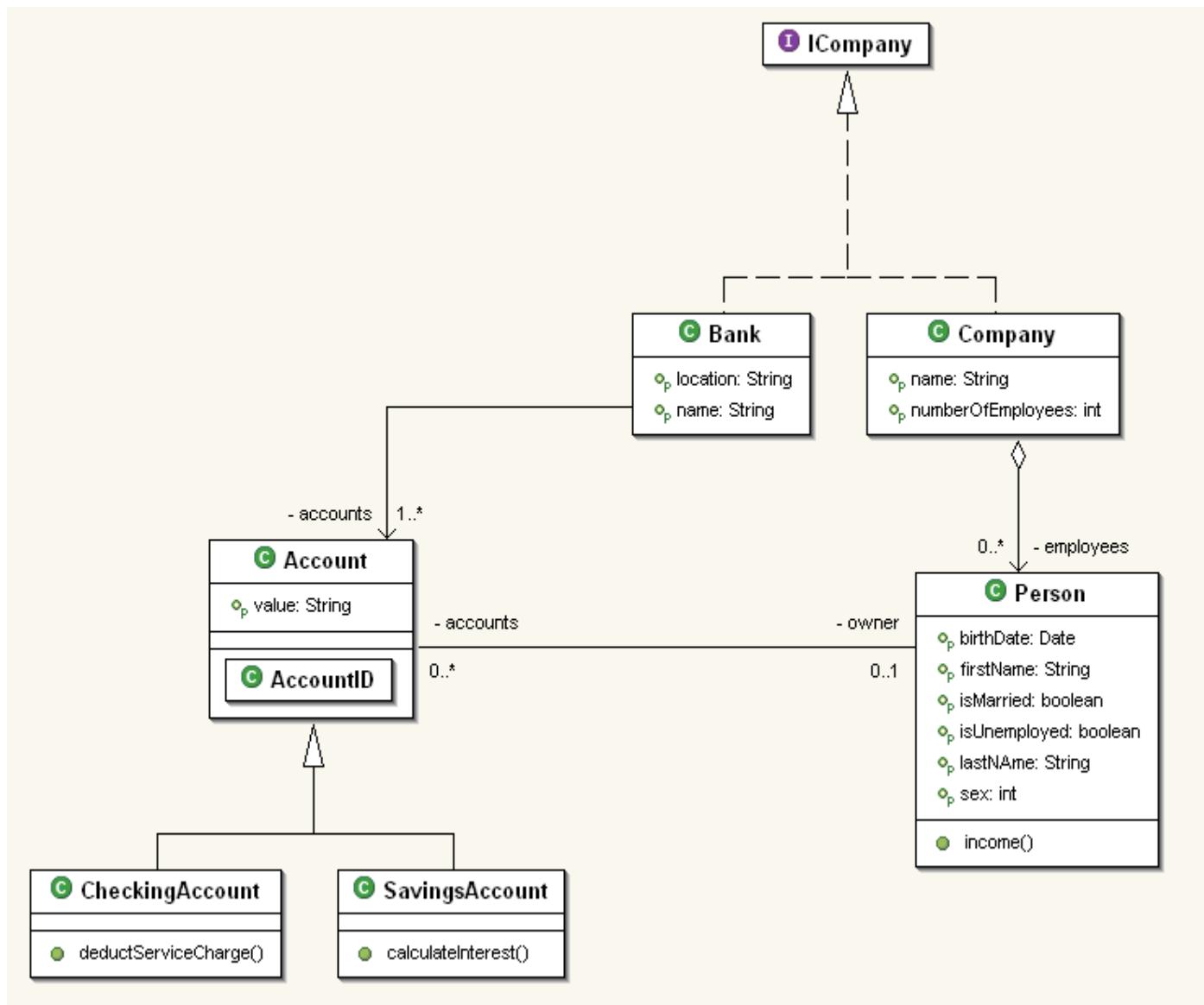
If you select an element, n level from the selected element will be displayed.

When working on large project, it is not recommended to create a diagram with scope "all" and level -1 which would be a full project diagram (the process could be heavy).

After selecting the option, a class selection dialog appears:



The expected diagram id created:



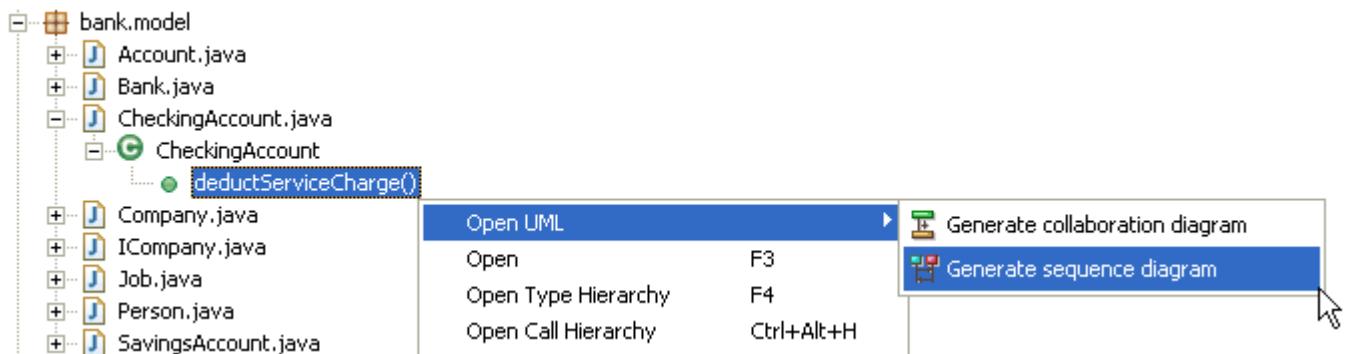
1.2 To a State Diagram

You can navigate from a class or an interface to a State diagram.
Select a class or interface, open the popup menu: **State diagram editor**

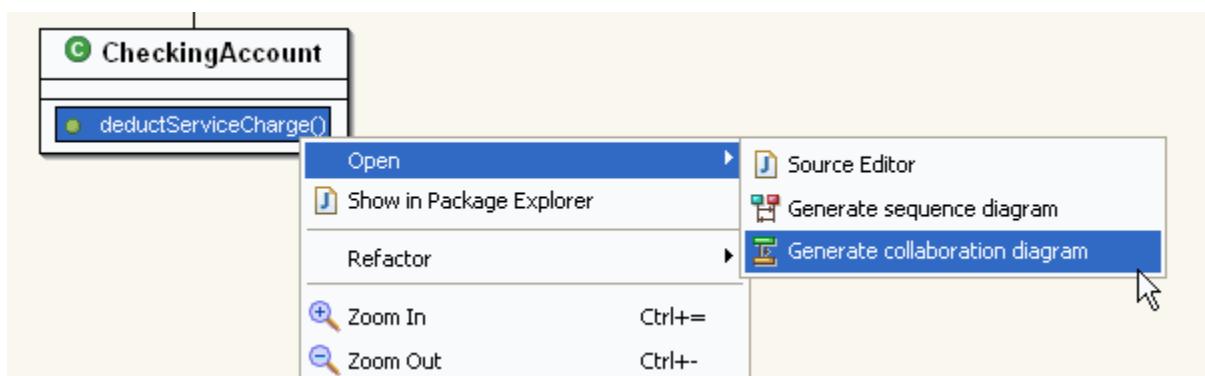


1.3 To a Sequence or Collaboration Diagram

You can navigate from a method to a Sequence/collaboration diagram.
Select a method, open the popup menu: **Generate sequence diagram**



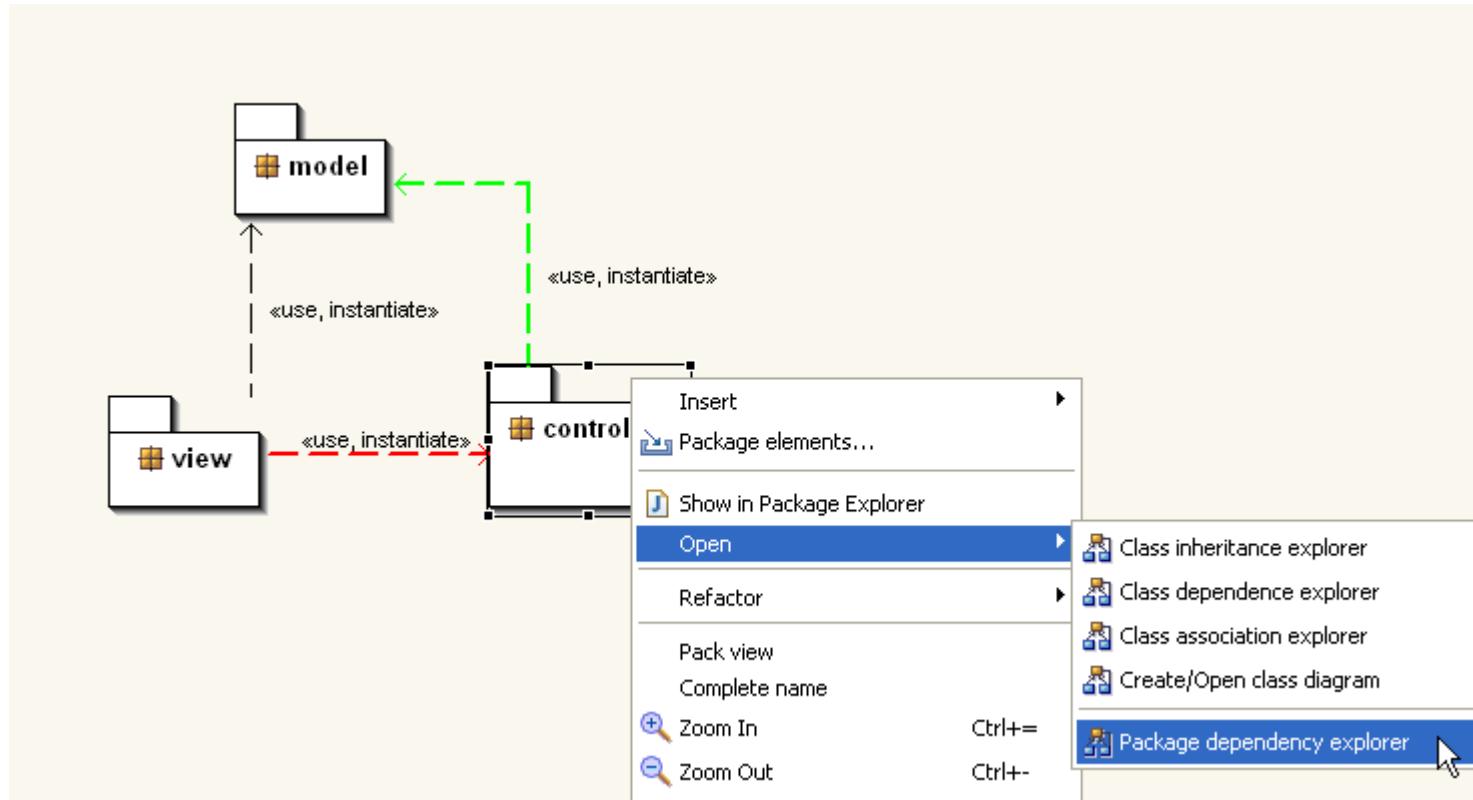
Select a method, open the popup menu: **Generate collaboration diagram**.



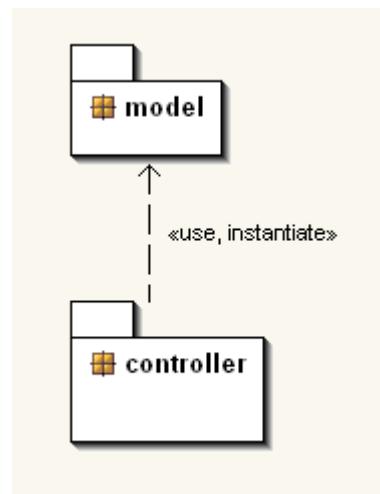
2. Navigation from a Package

2.1 To a Package-Centric Diagram

Select a package, open the popup menu: **Open>Package dependency explorer**



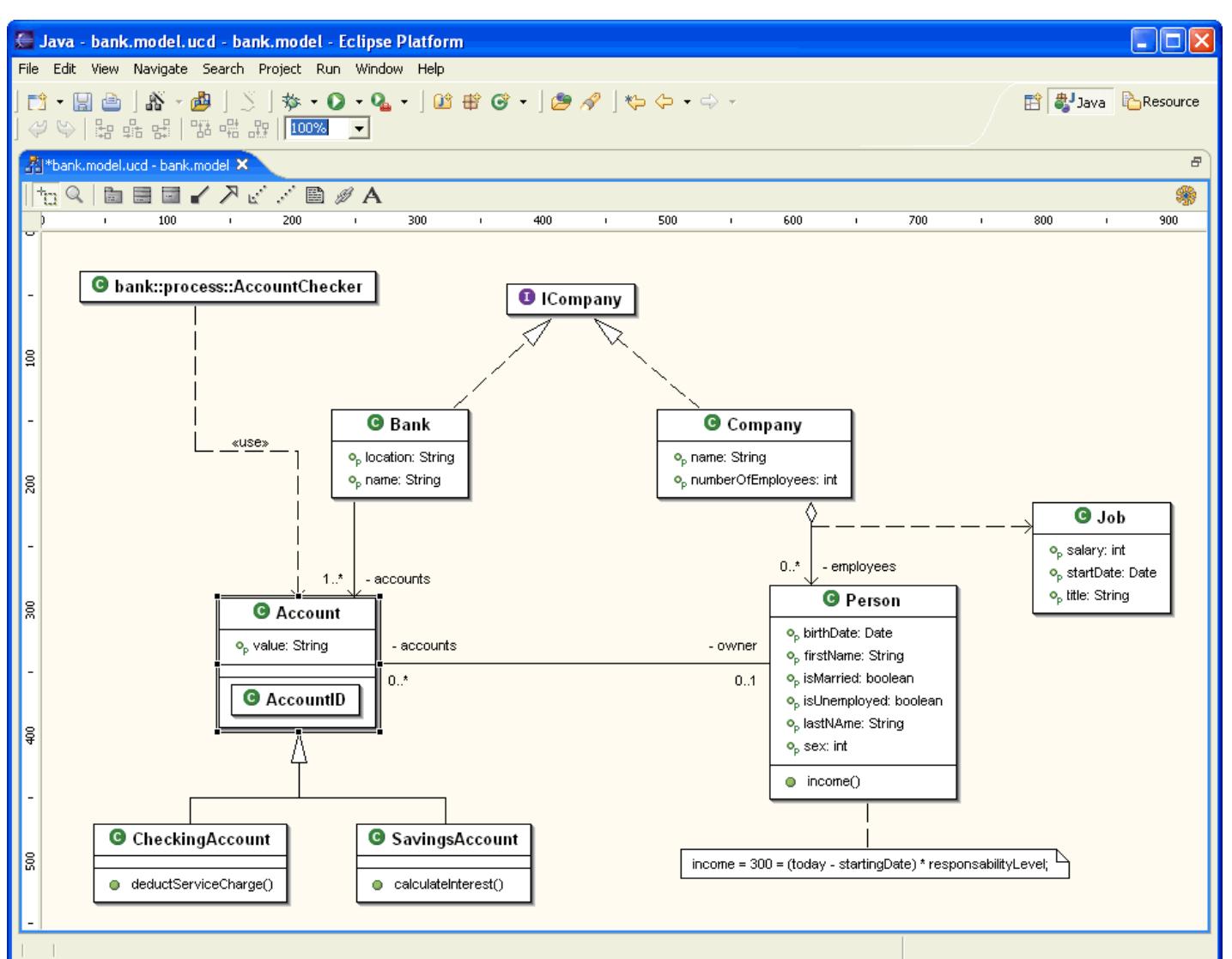
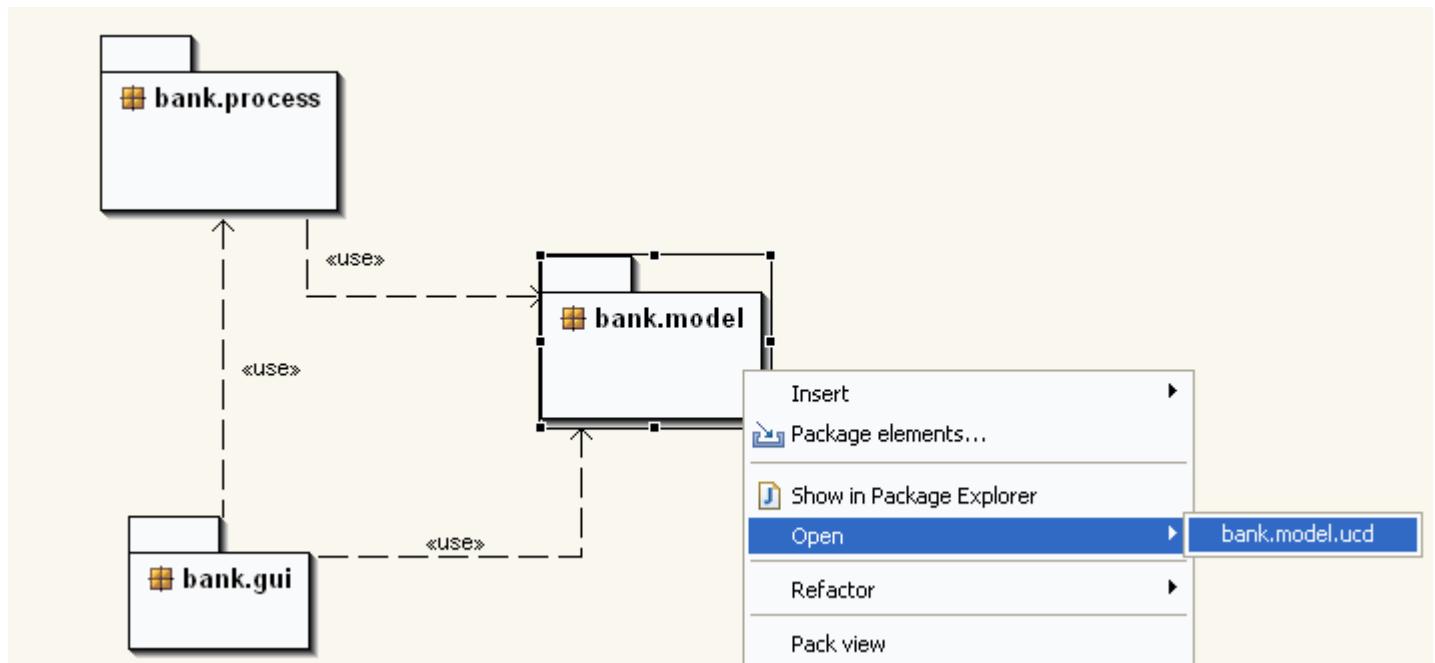
The default package dependency explorer focuses on the selected package and only displays its immediate outgoing dependencies environments.



2.2 To a Class Diagram

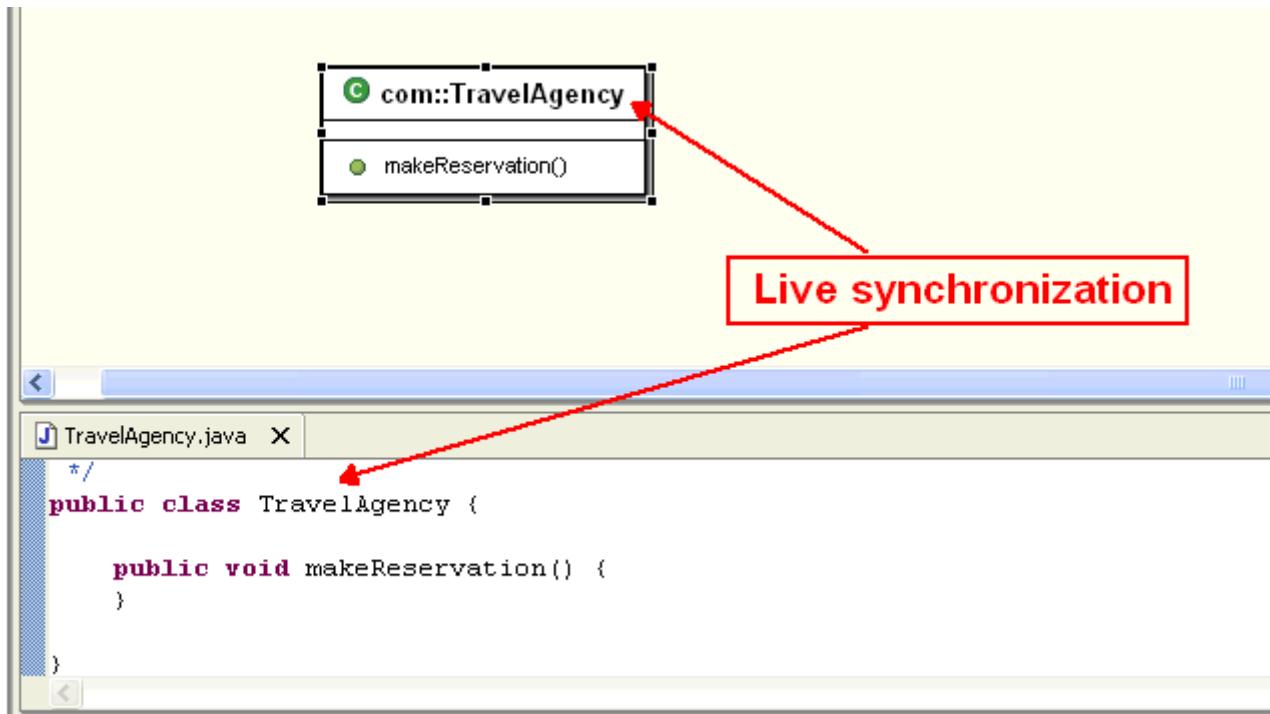
The navigation is the opportunity to browse several UML model. You can access this functionality through the contextual menu on a package.

- by clicking on *Open > myModel* where *myModel* is an existing model in the package.
- **double click on the selected package** will zoom inside and show classes.



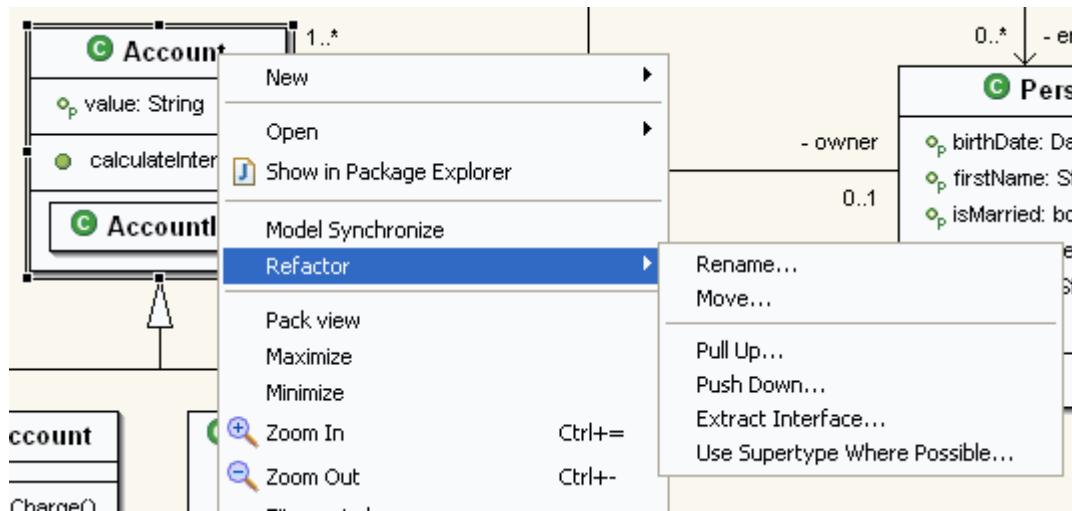
Real-Time Synchronization

The *EclipseUML plugin* is fully integrated into the Eclipse Framework. EclipseUML is a seamless integrated component in your IDE. You can modify your code either from the Java editor or through the UML view; both are always totally and immediately synchronized.



Refactoring

All Eclipse refactoring functions are available inside the Class Diagram Editor. To access these functions, select an element in the class diagram editor, right click to open the popup menu and deploy the refactor submenu.

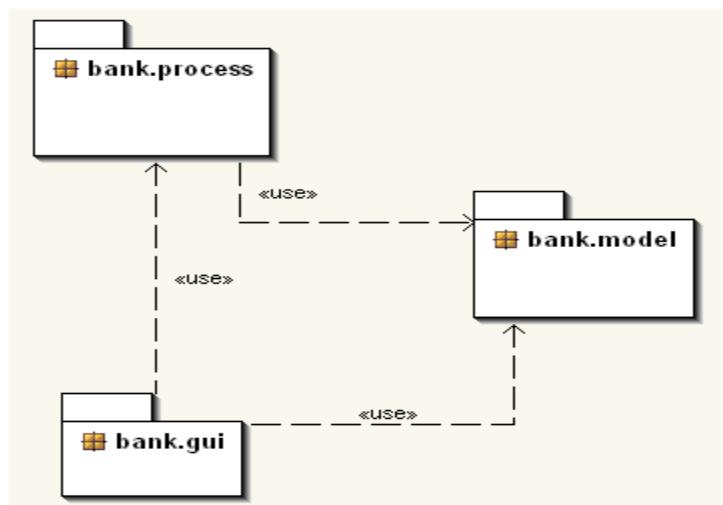


EclipseUML Class Diagram and Java code are immediately synchronized as soon as the refactor function is activated.

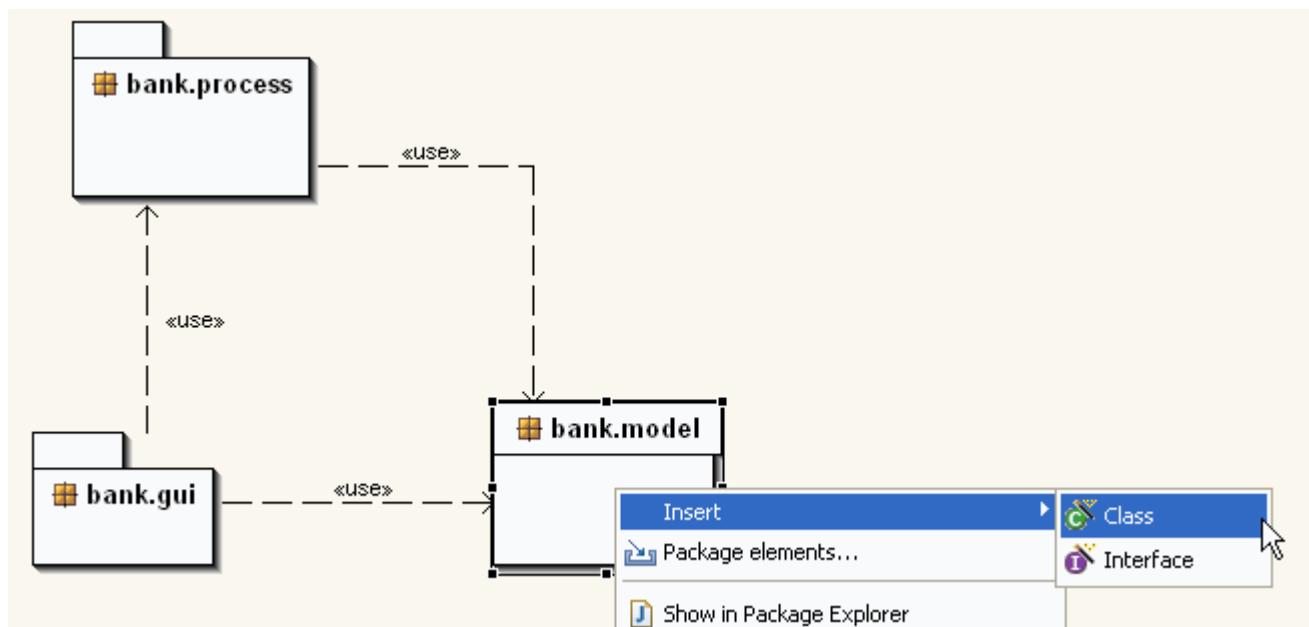
See more information in the [Refactoring Chapter](#).

Working At Package Level

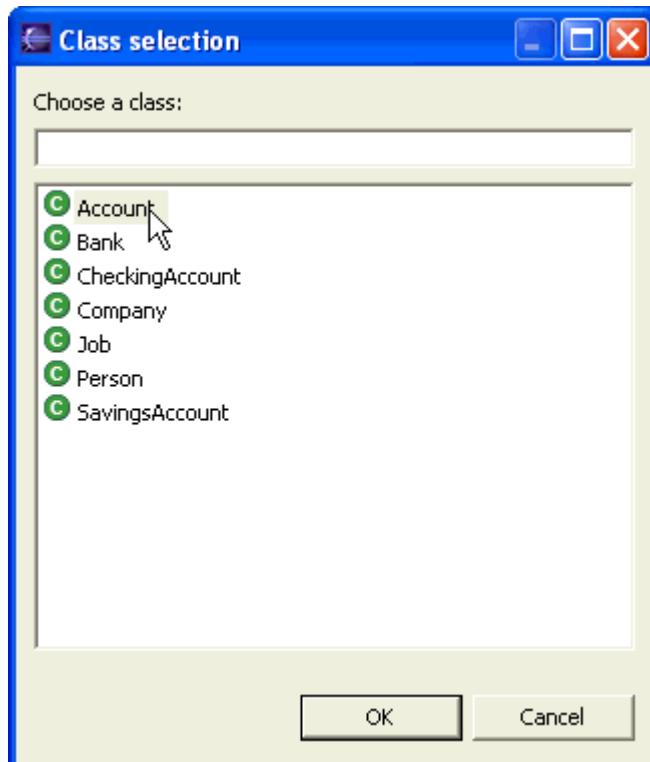
Any class diagram created in a package fragment root is a specific diagram dedicated to show the packages and their relationships. The other class diagrams (created in packages) can also work with packages but they must be explicitly inserted in the diagram. The following screenshot shows an example. Three packages are represented with their dependencies: model, process and gui. Here the package gui depends on both packages model and process, while the process package depends on the model package.



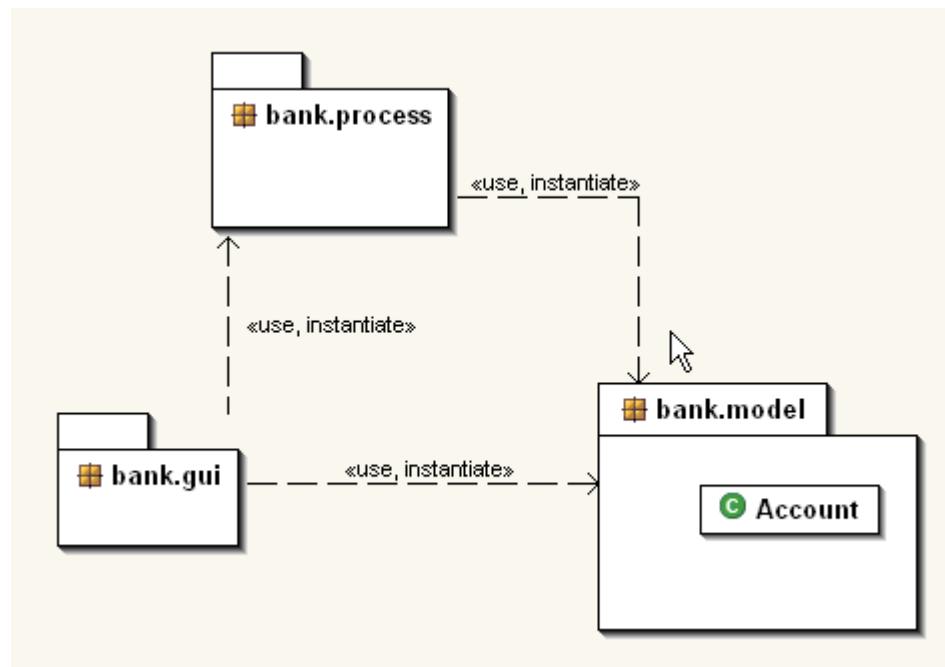
Classes (and Interfaces) can be inserted in the packages by using the contextual menu item *Insert > Class* (or *Insert > Interface*) on the package.



A dialog box proposes to choose one the packages classes (or interfaces).



Once chosen, the class (or interface) appears in the package view.



The layout can be applied to either the whole diagram or a single package with its inserted element.

View Selector

Stereotypes, attributes, methods, inner classes and inner interfaces can be shown or hidden in the Class Diagram. This is done through a window called *View selector*. You can access it from the Class Diagram or package, class or interface context menus > **View Selector**.

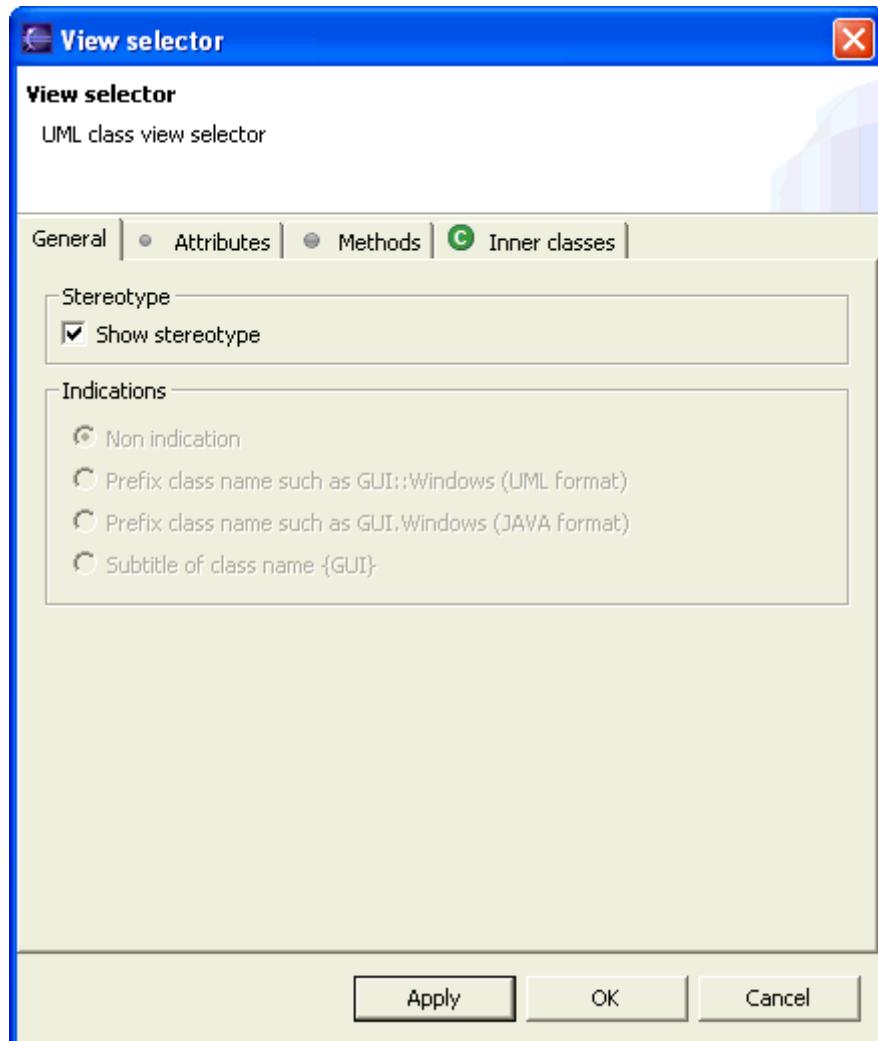
Four options are available:

1. [General](#)
2. [Attributes](#)
3. [Methods](#)
4. [Inner Classes](#)

1. General

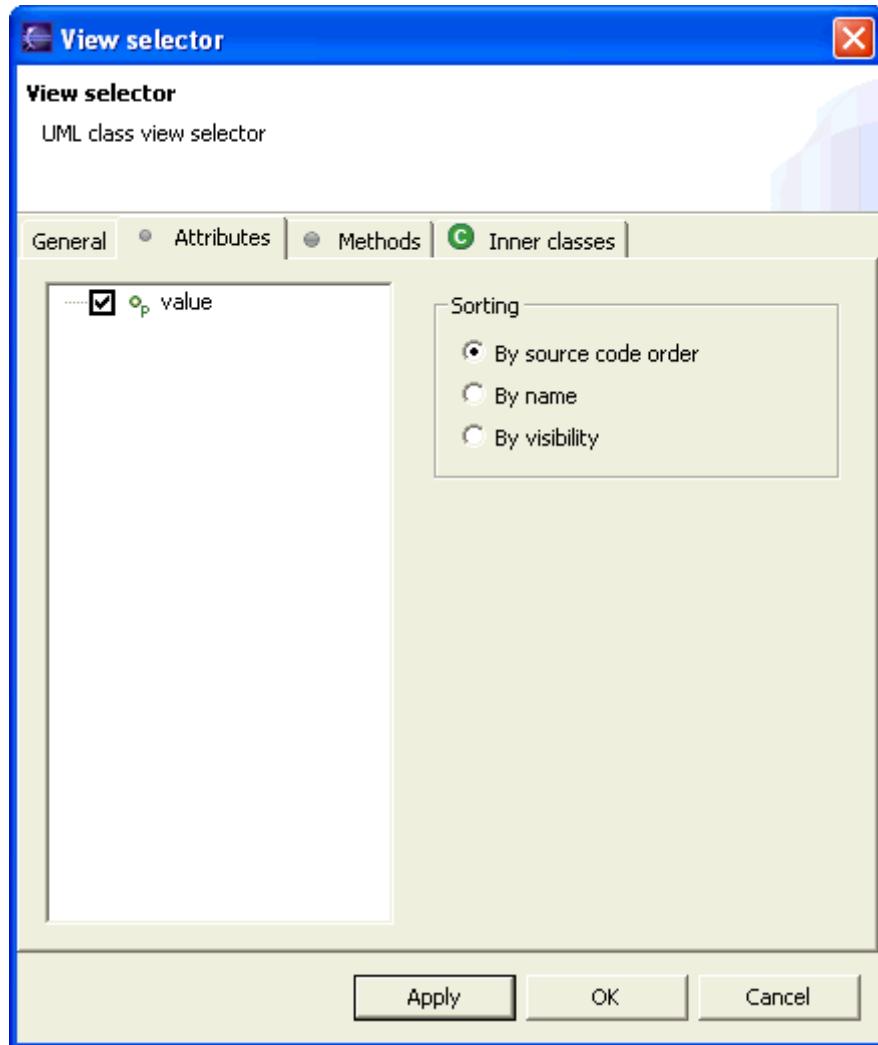
On classes and interfaces, you can configure:

- **Indications:** Allows you to show the class package name in the element. If the class package is the current diagram package, there is no indication.



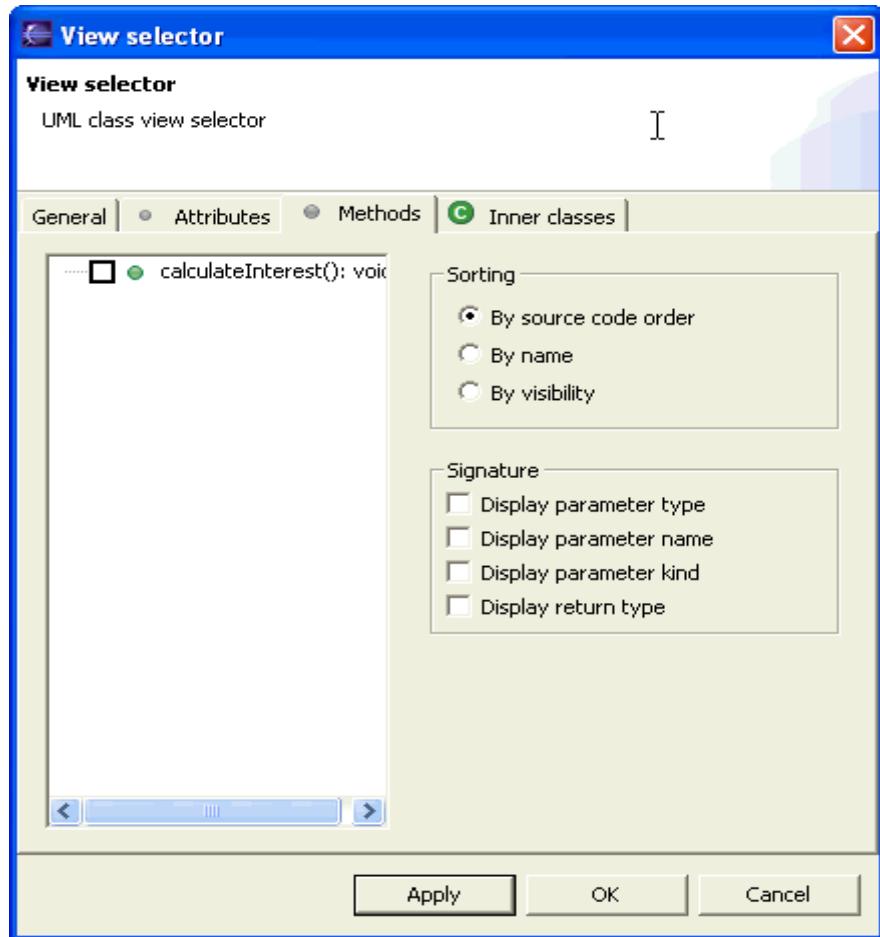
2. Attributes

- **Attributes** views in the class diagram are set according to the [preferences](#) values, but you can override the behavior for this class (hide or show attributes) either by (de)selecting directly the attribute, or through the attributes visibility.



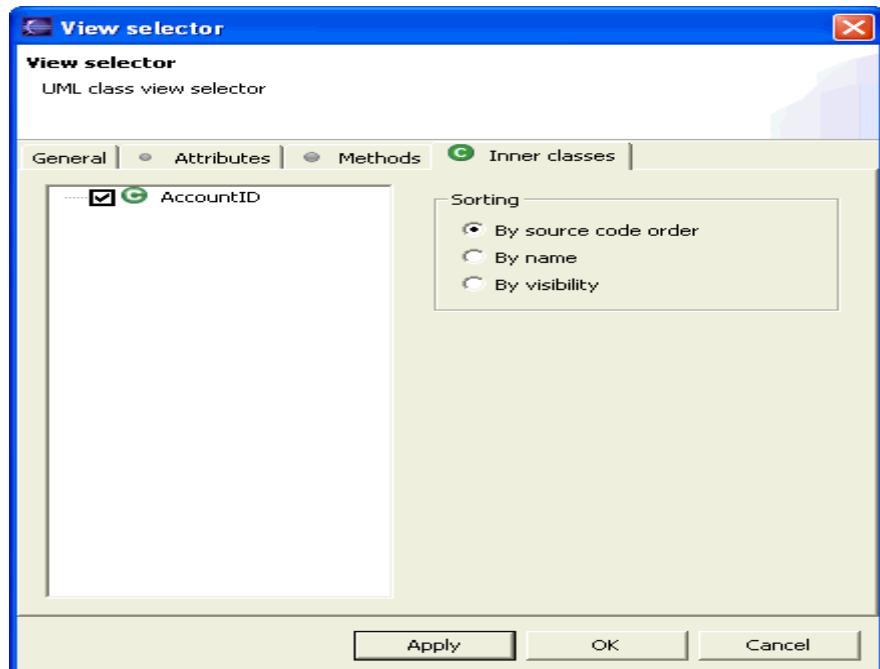
3. Methods

- **Methods** views in the class diagram are set according to the [preferences](#) values, but you can override the behavior for this class (hide or show methods) either by (de)selecting directly the method, or through the methods visibility.



4. Inner Classes

- **Inner classes** views in the class diagram are set according to the [preferences](#) values, but you can override the behavior for this class (hide or show methods) either by (de)selecting directly the method, or through the methods visibility.



Property Concept

The property concept has been improved inside EclipseUML.

The logic is that if you select Show Association Member then accessors created by the association will be displayed.

If you unselect both Show Association member and Show Accessors, then the Property Concept will be activated by hiding all accessors.

Property concept is now composed by two options:

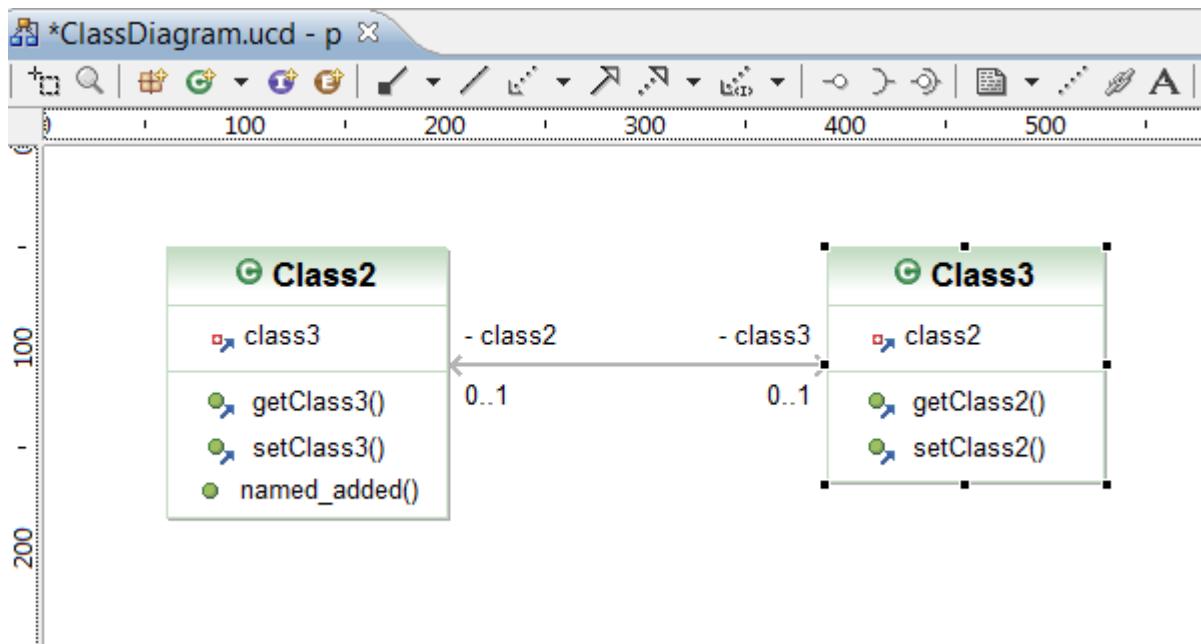
1. [Show Hide Association member](#)
2. [Show Hide Accessors](#)

1. Show Hide Association Members

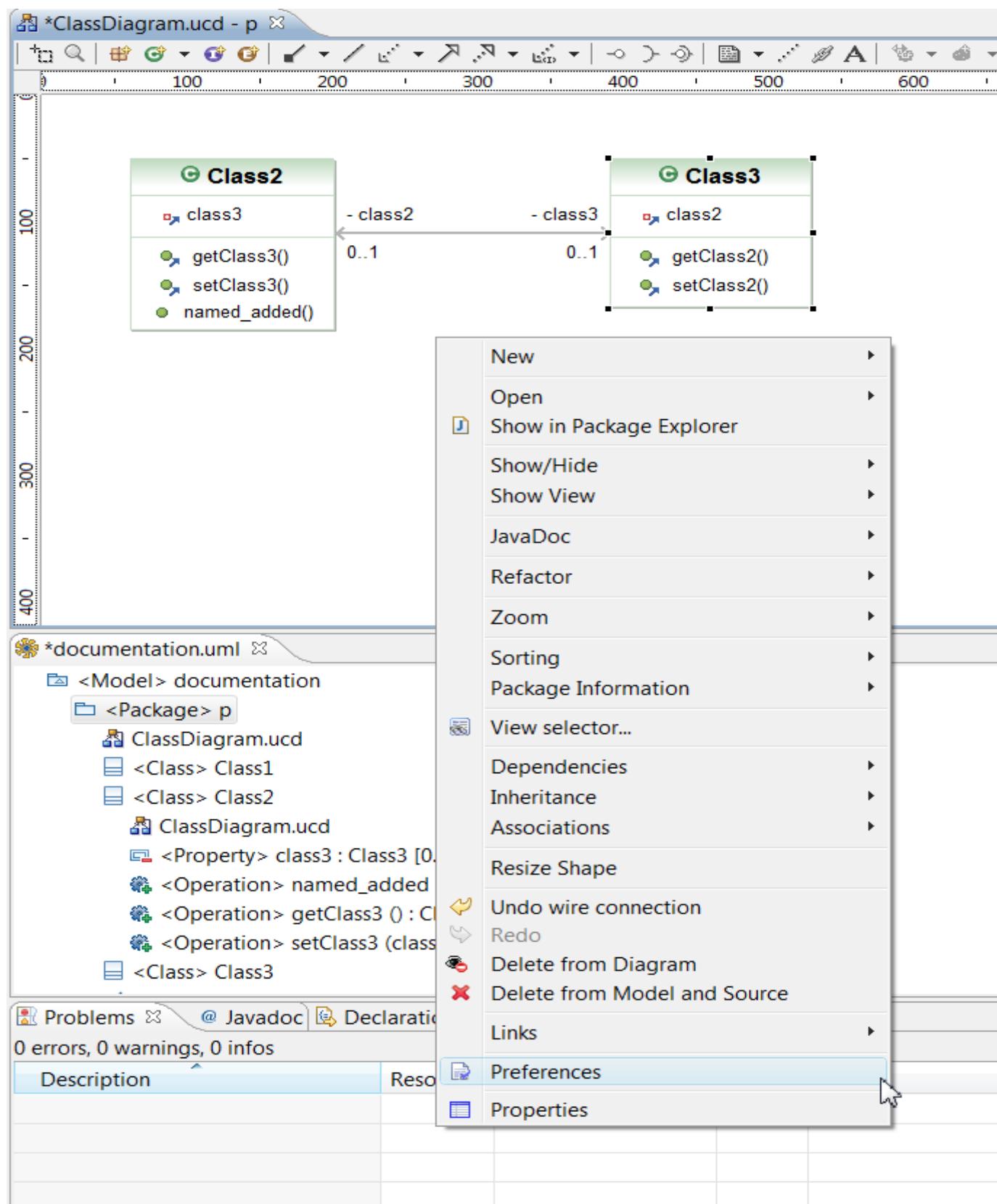
When working with a class diagram, it is possible to activate or not the Property concept related to association members.

These two Classes having a 0..1 to 0...1 association between them.

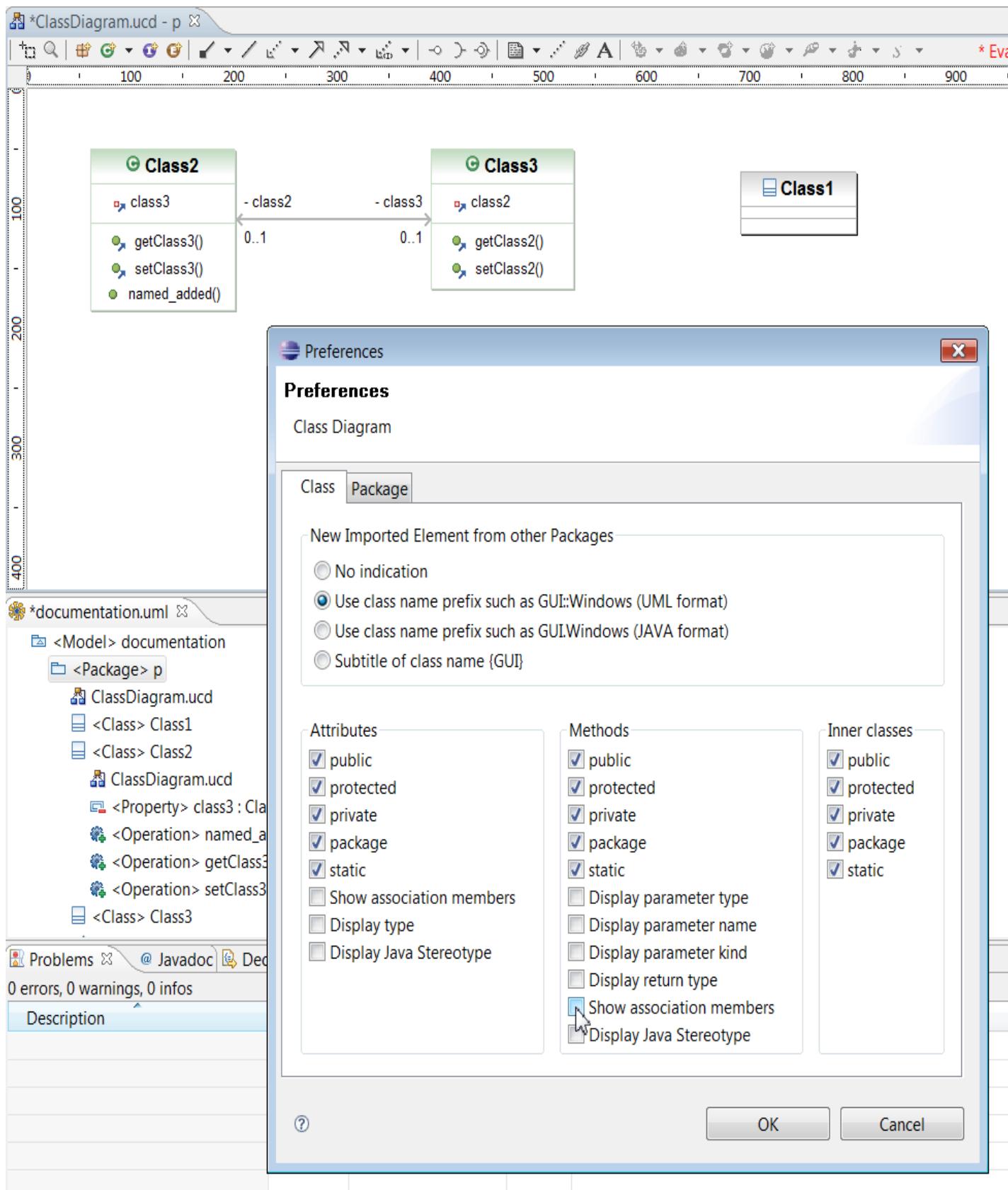
You can notice that Property and Operation which are related to an association have a small arrow under the Attribute or method icon.



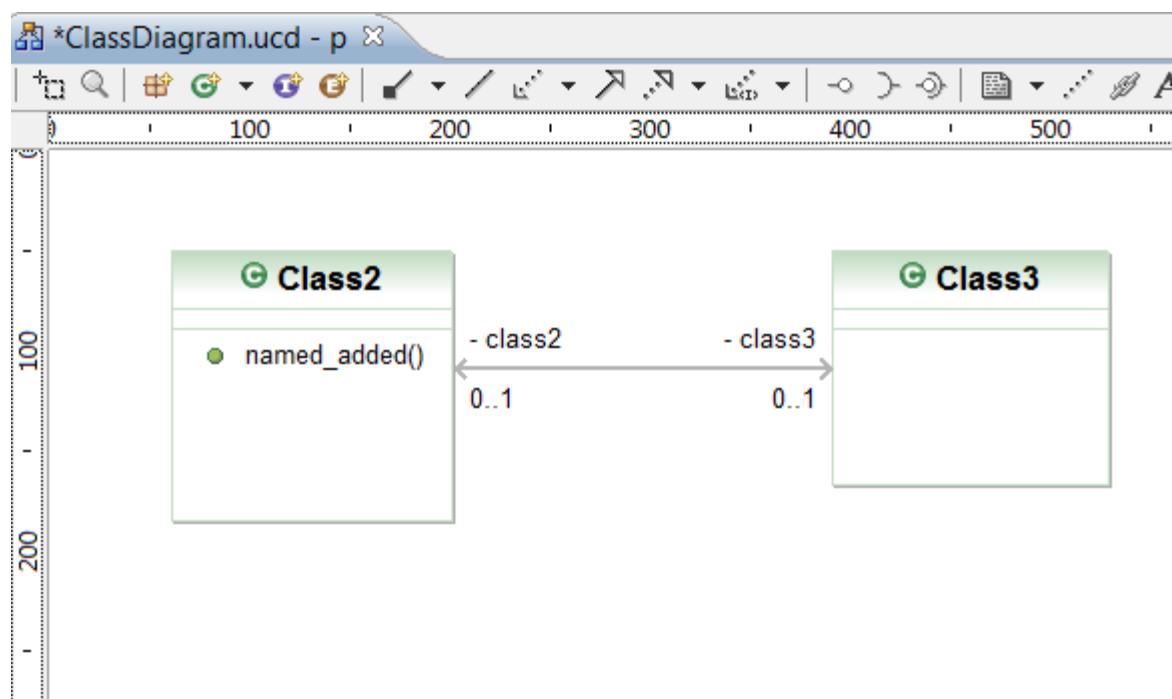
If you want to hide this association Property and accessors, then click on the **Class Diagram Background > Preferences**



Unselect Show association members field and click on the Ok button



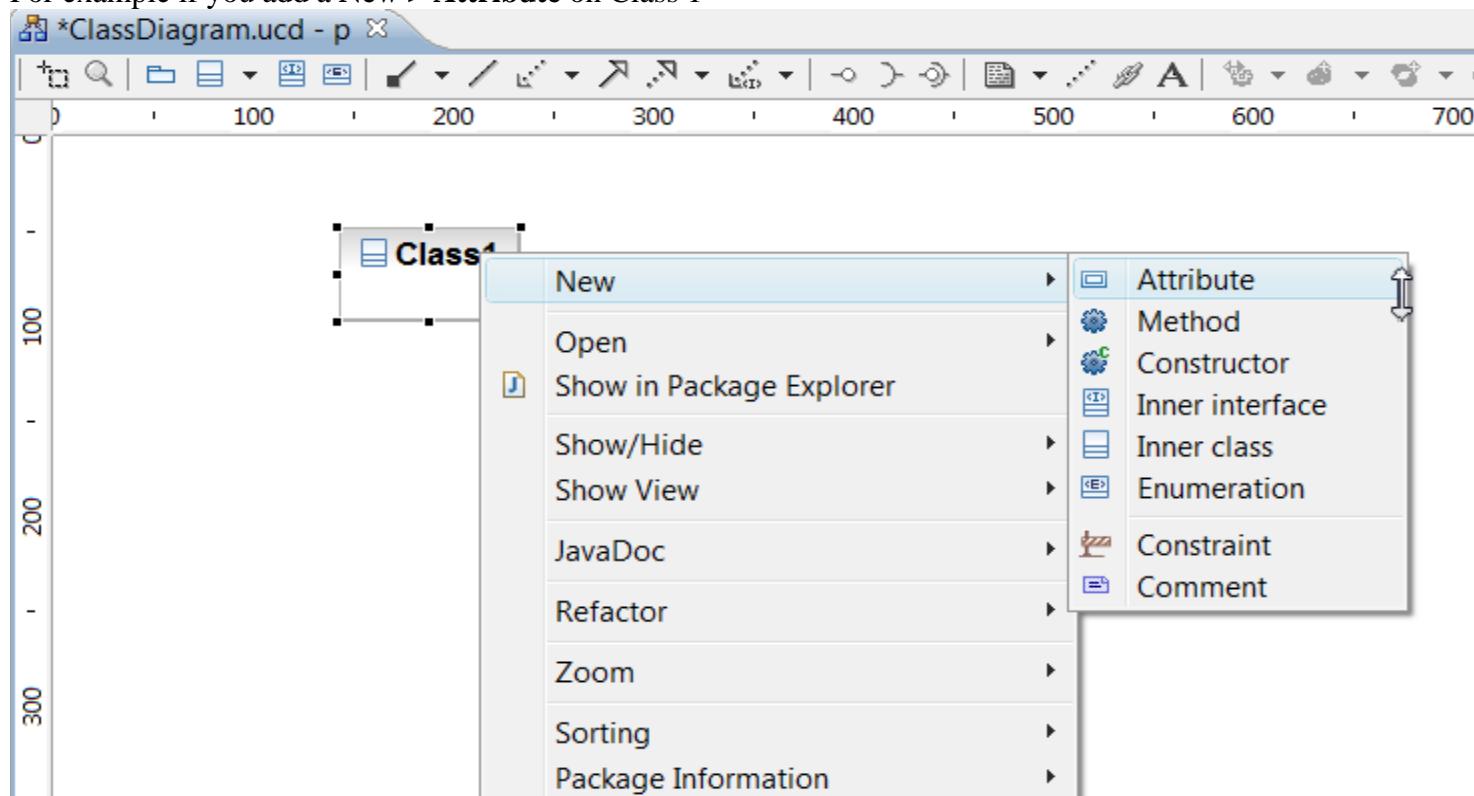
The association members have been immediately hidden.
Note that you can hide either Attributes or Methods independently.



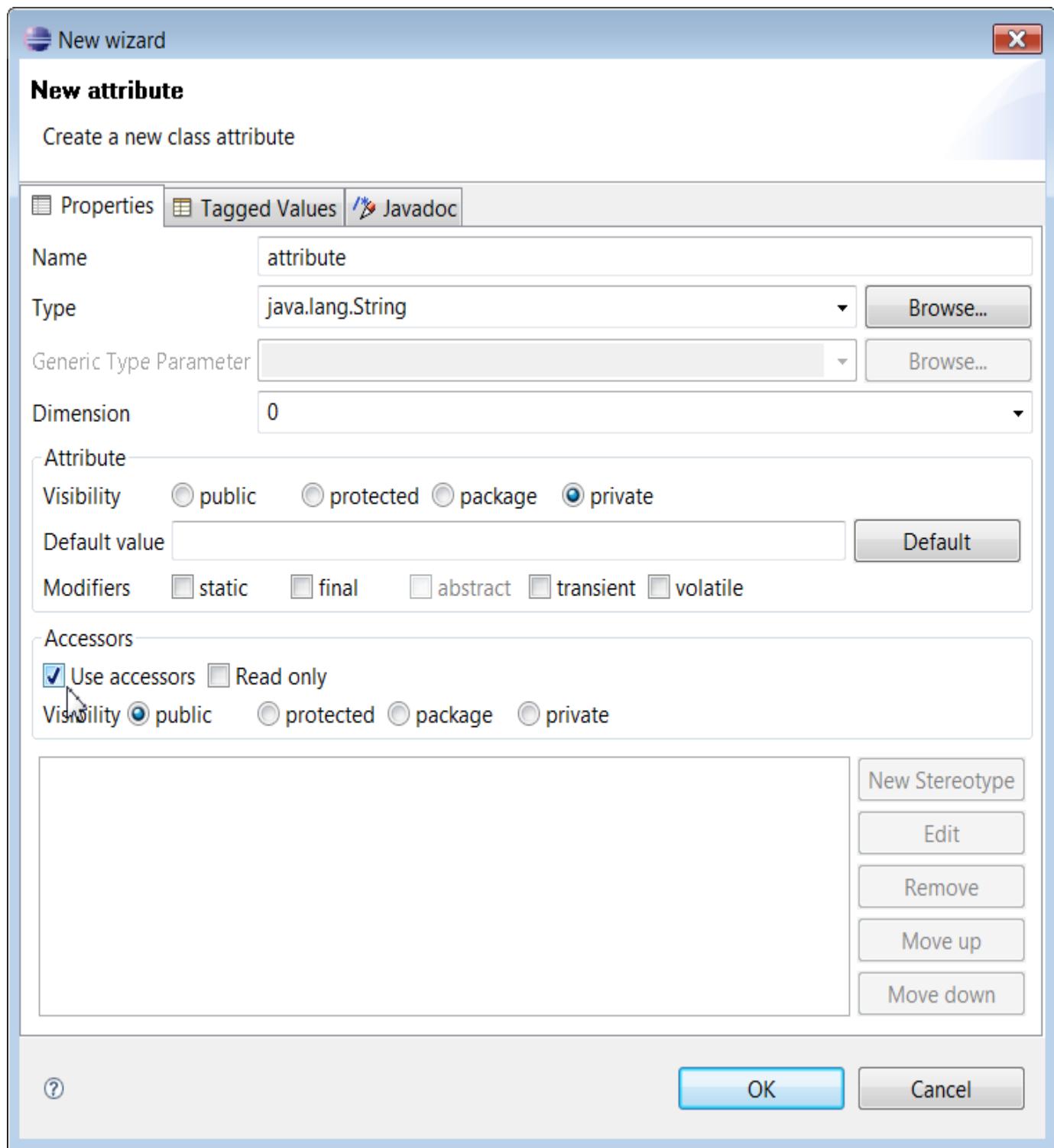
2. Show Hide Accessors

It is important in Java to generate accessors for every attribute, but it is not really important to display this accessors information in a UML Class Diagram.

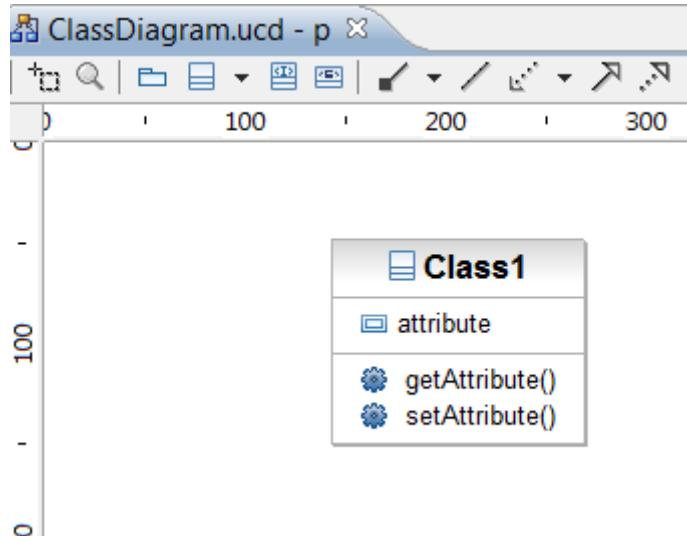
You can hide accessors by using the Show Hide accessors option in the Class Diagram contextual menu. For example if you add a **New > Attribute** on Class 1



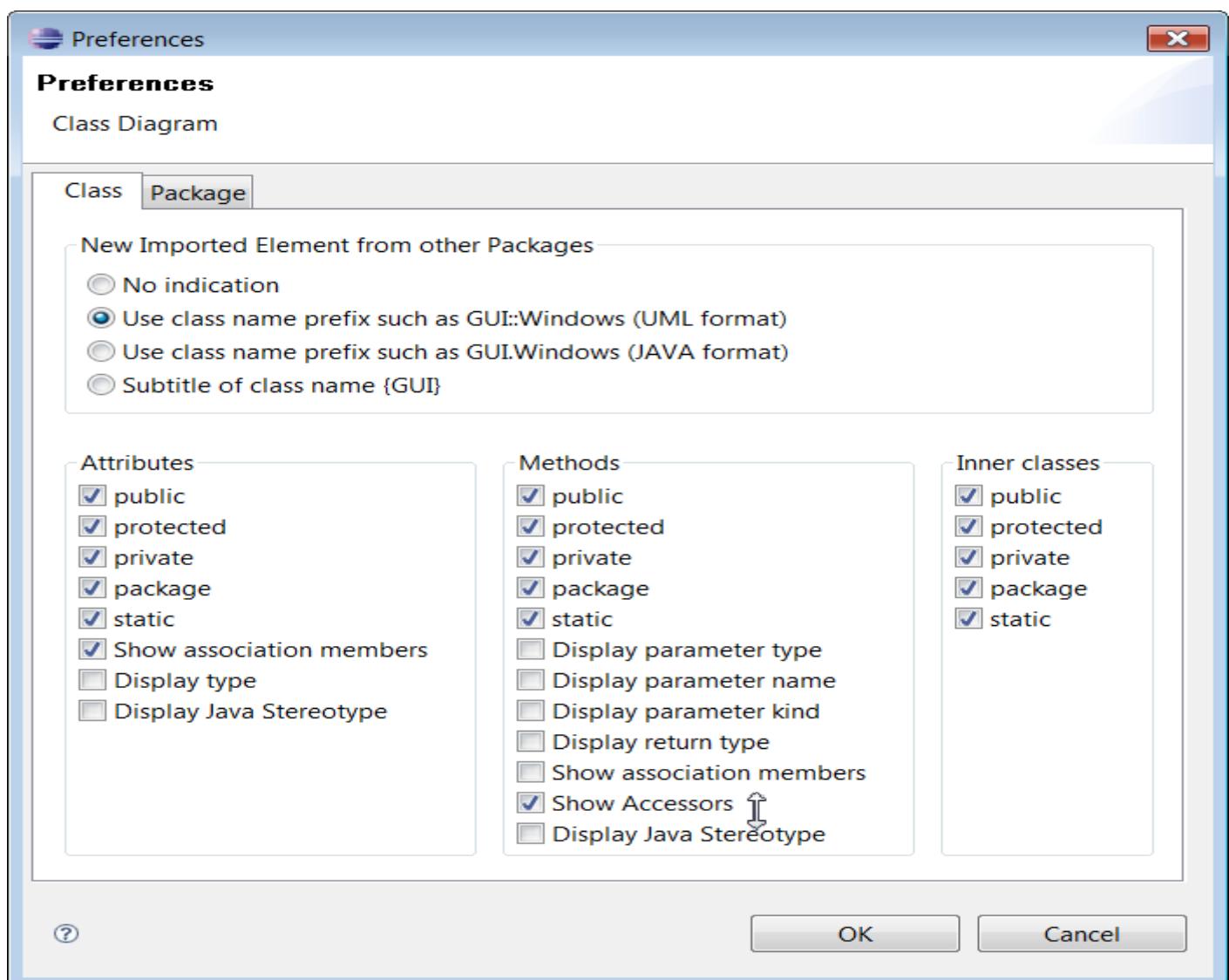
The following wizard will appear and allow to create or not accessors for this Attribute.



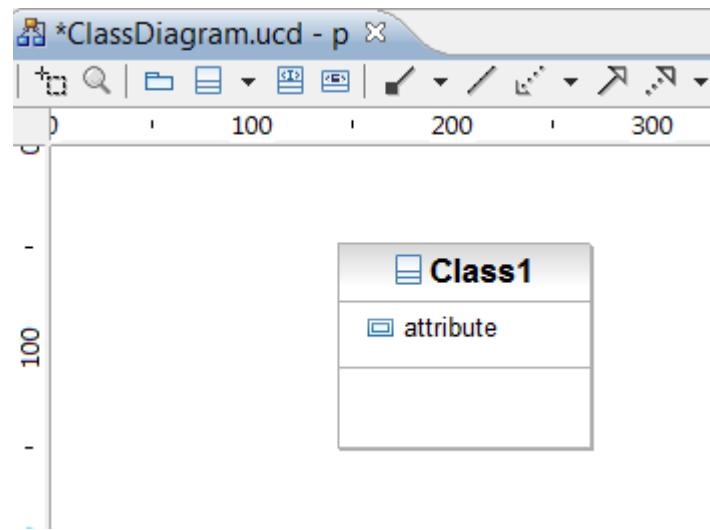
If we select to generate accessors, then the java code is clean but this information is also in the Class Diagram.



In order to avoid adding none necessary information in your class diagram and hide attribute accessors then **click on the Class Diagram background and unselect Show Accessors option.**



Here is your class after selecting this option



Association Class

In modeling an association, there are times when you need to include another class because it includes valuable information about the relationship.

For this you would use an *association class* that you tie to the primary association.

An association class is represented like a normal class. The difference is that the association line between the primary classes intersects a dotted line connected to the association class.

EclipseUML provides two kinds of association class:

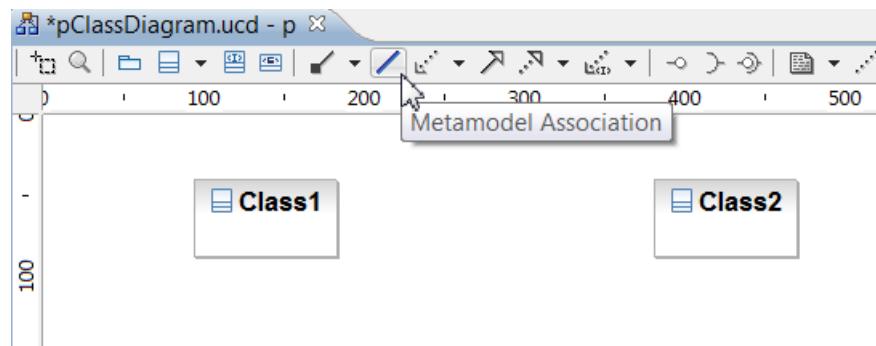
1. [Java Association Class](#)
2. [Metamodel Association Class \(MDA\)](#)

1. Java Association Class

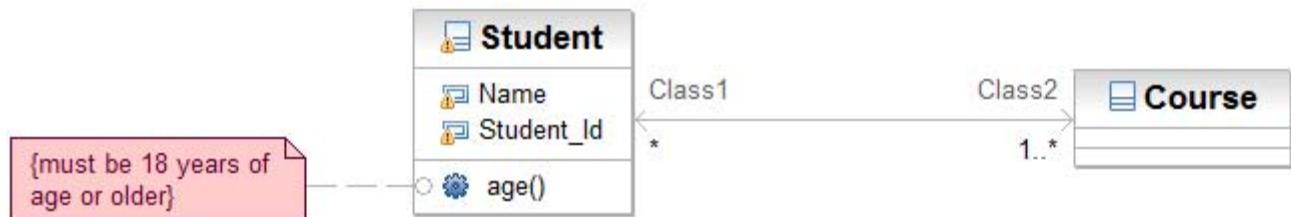
Figure below shows an association class for Student Enrollment example.

Note that Association Class is only working if using Metamodel Association menu.

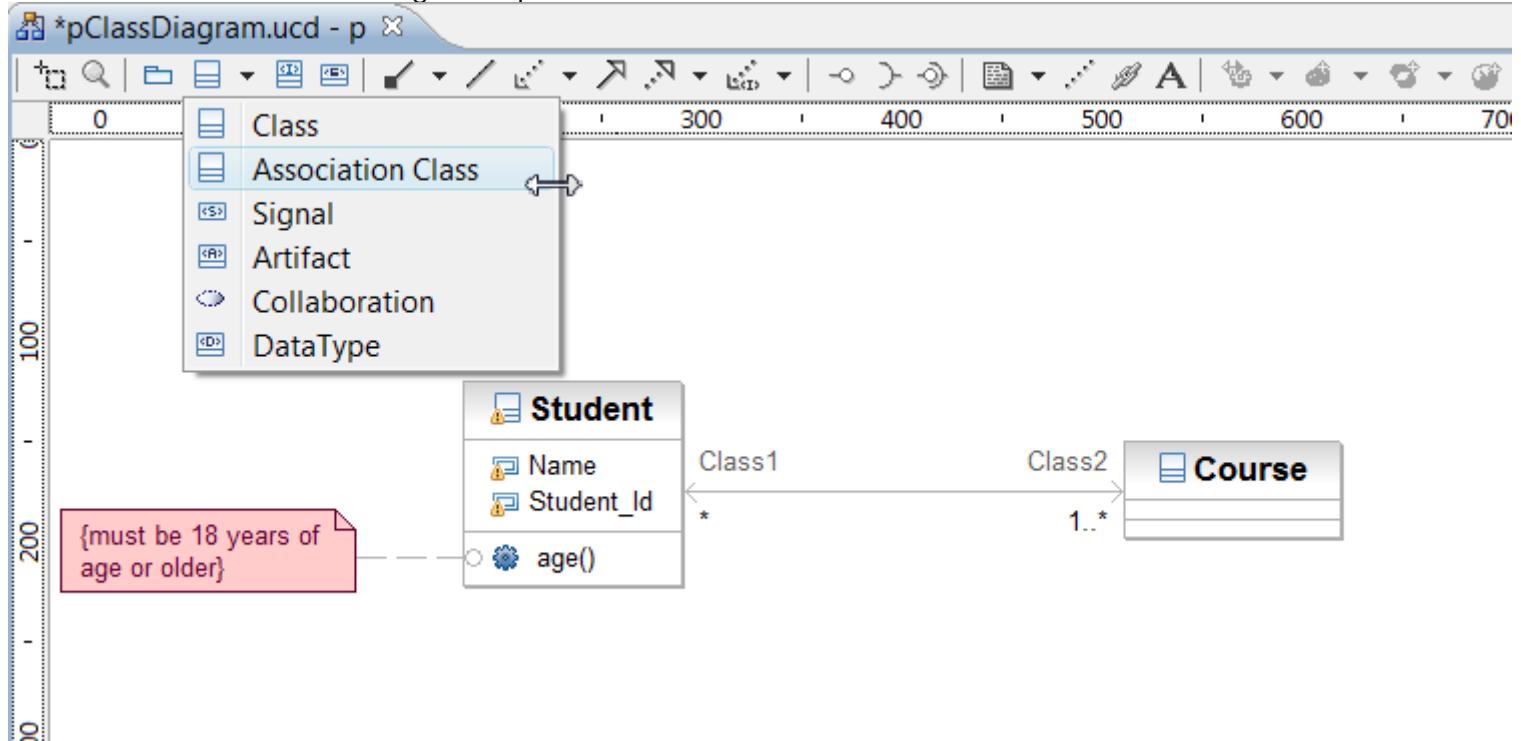
For example let's create two Java Classes in your Class Diagram and add a metamodel association between them



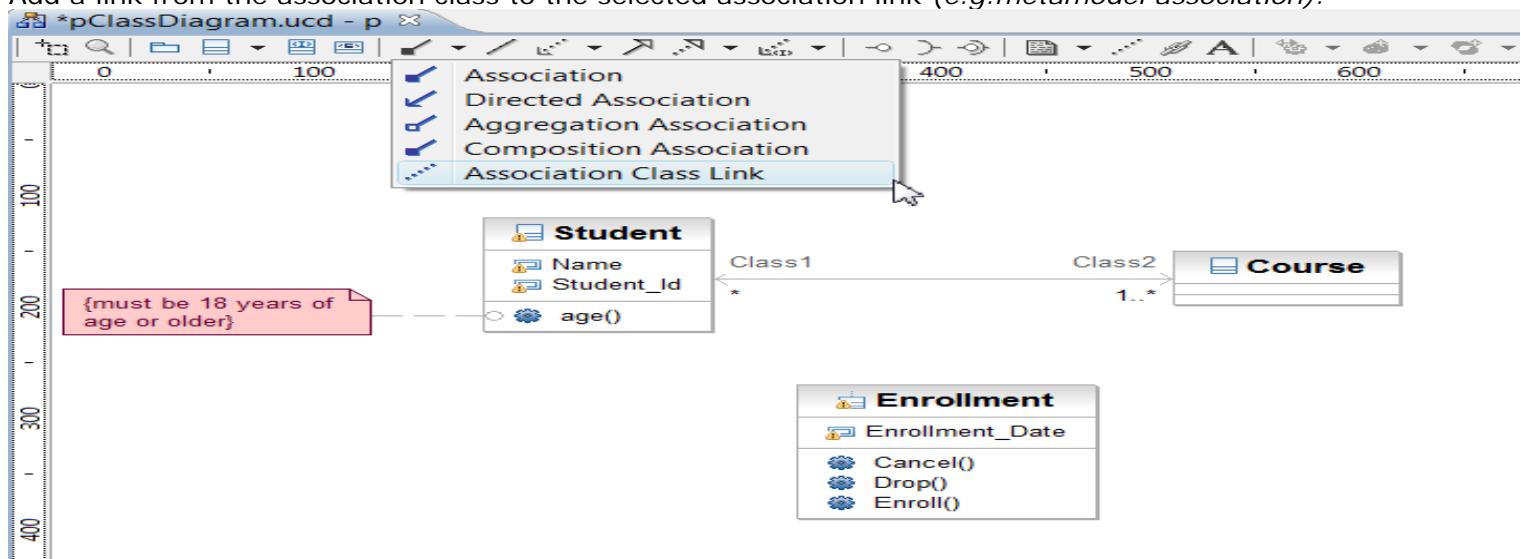
Rename both Classes to Student and Course, add attributes & methods in the Student Class, and add a constraint on the age.



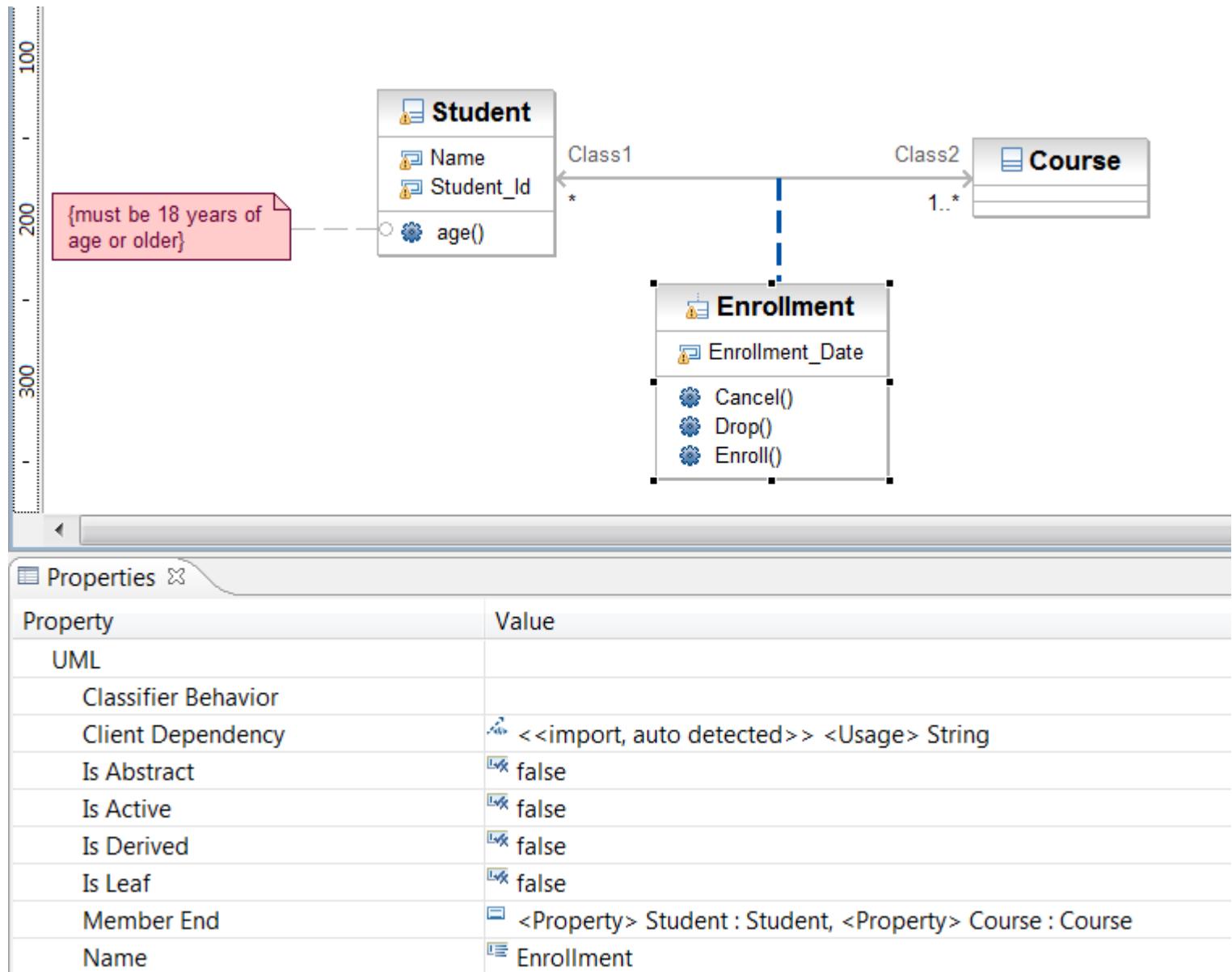
Create an association class using the top down menu.



Add a link from the association class to the selected association link (e.g. metamodel association).



The association class ends will be automatically created once the association class link is connected to the association.

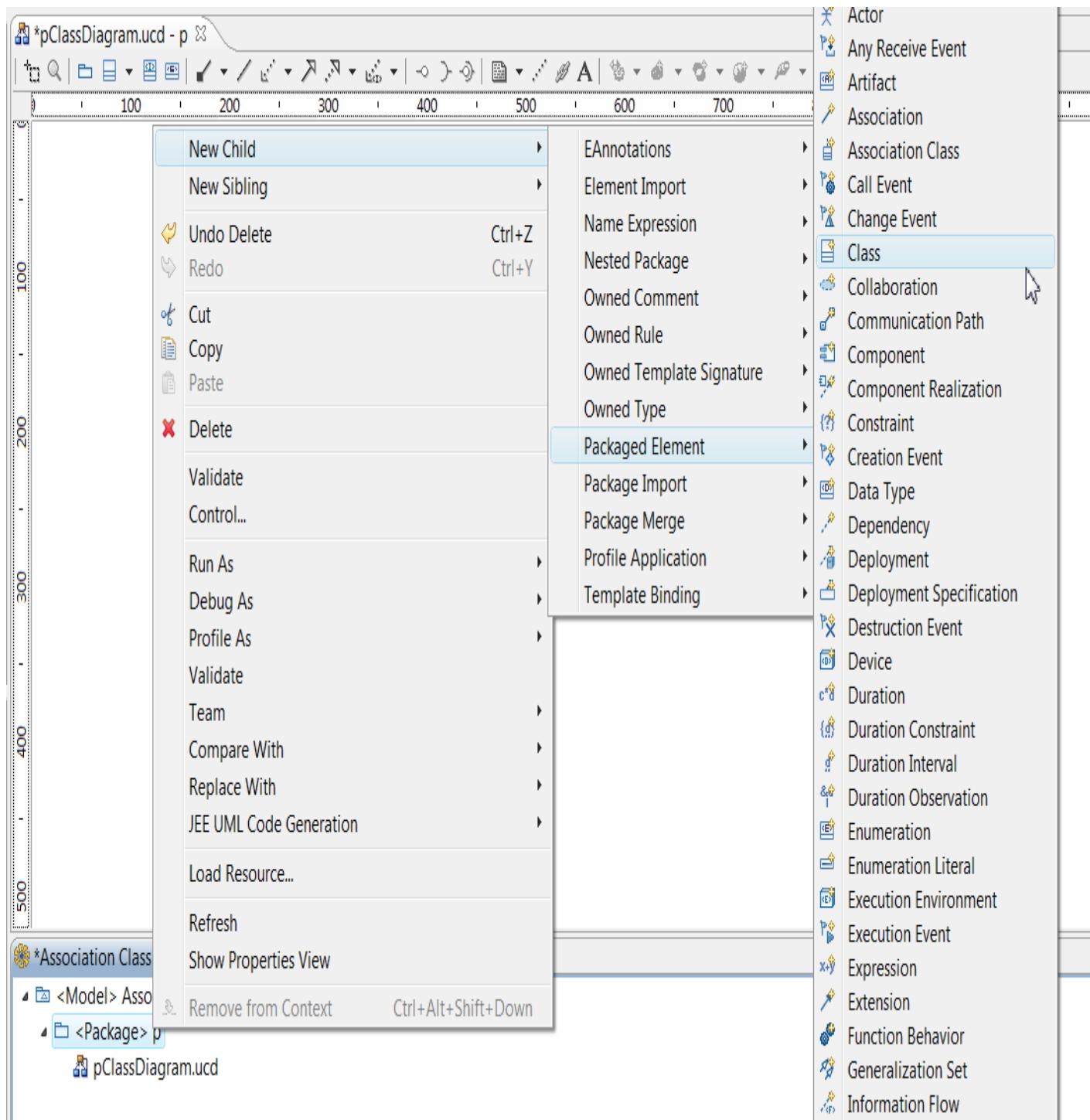


2. Metamodel Association Class

Let use the XMI Editor to create all class diagram elements and not generate any java code.

Figure below shows an association class for the airline industry example.

Create a new Class in the metamodel by selecting the **package > New Child > Package Element > Class**



Rename the created class to Flight using **the property view Name Field**.

The screenshot shows the UML property view on the left and the package browser on the right.

Property View (Left):

Property	Value
UML	
Classifier Behavior	
Client Dependency	
Is Abstract	false
Is Active	false
Is Leaf	false
Name	Flight
Owned Port	
Powertype Extent	
Redefined Classifier	
Representation	
Template Parameter	
Use Case	
Visibility	Public

Package Browser (Right):

- <Model> Association Class Example
- <Package> p
 - pClassDiagram.ucd
 - <Class> Flight

Create a new association class by selecting **the package > New Child > Packaged Element > Association Class**.

The screenshot shows the UML package browser with a context menu open over a package element.

Context Menu (Open):

- New Child
- New Sibling
- Undo New Packaged Element | Class
- Redo
- Cut
- Copy
- Paste
- Delete
- Validate
- Control...
- Run As
- Debug As
- Profile As
- Validate
- Team
- Compare With
- Replace With
- JEE UML Code Generation
- Load Resource...
- Refresh
- Show Properties View
- Remove from Context

Toolbox (Right):

- Actor
- Any Receive Event
- Artifact
- Association
- Association Class
- Call Event
- Change Event
- Class
- Collaboration
- Communication Path
- Component
- Component Realization
- Constraint
- Creation Event
- Data Type
- Dependency
- Deployment
- Deployment Specification
- Destruction Event
- Device
- Duration
- Duration Constraint
- Duration Interval
- Duration Observation
- Enumeration
- Enumeration Literal
- Execution Environment
- Execution Event
- Expression
- Extension
- Function Behavior
- Generalization Set
- Information Flow
- Information Item
- Instance Specification
- Instance Value

Drag and drop the metamodel classes from the XMI editor to the Class Diagram (e.g. No Java code will be created).

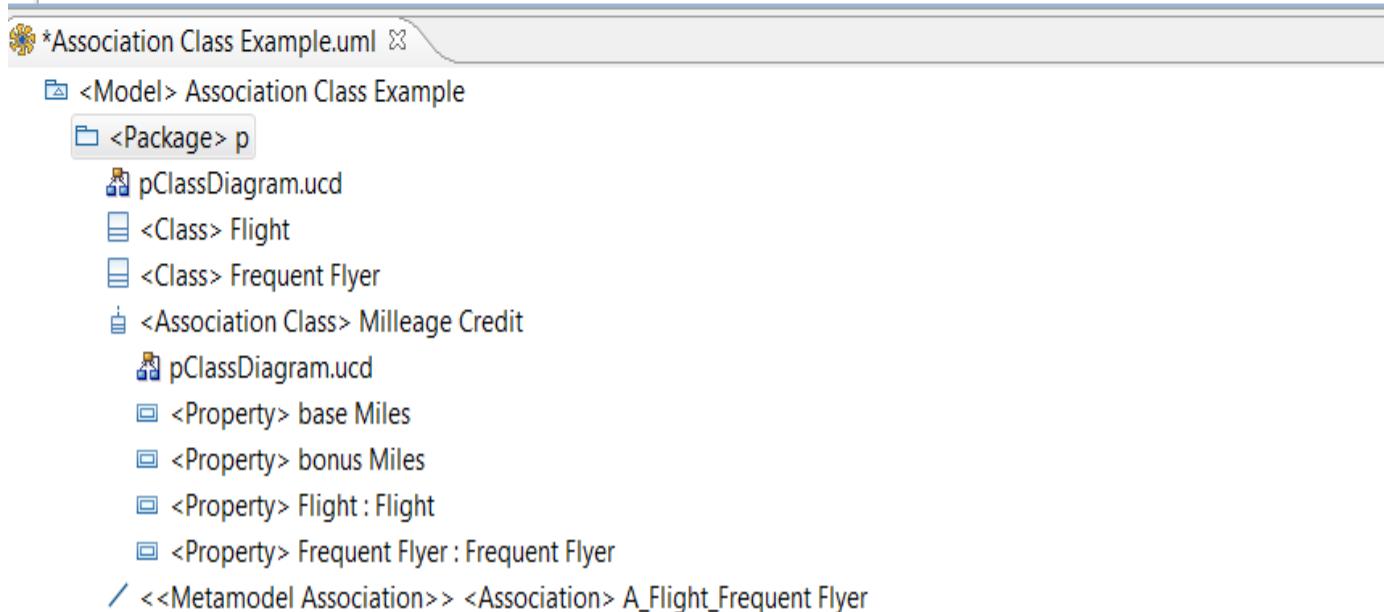
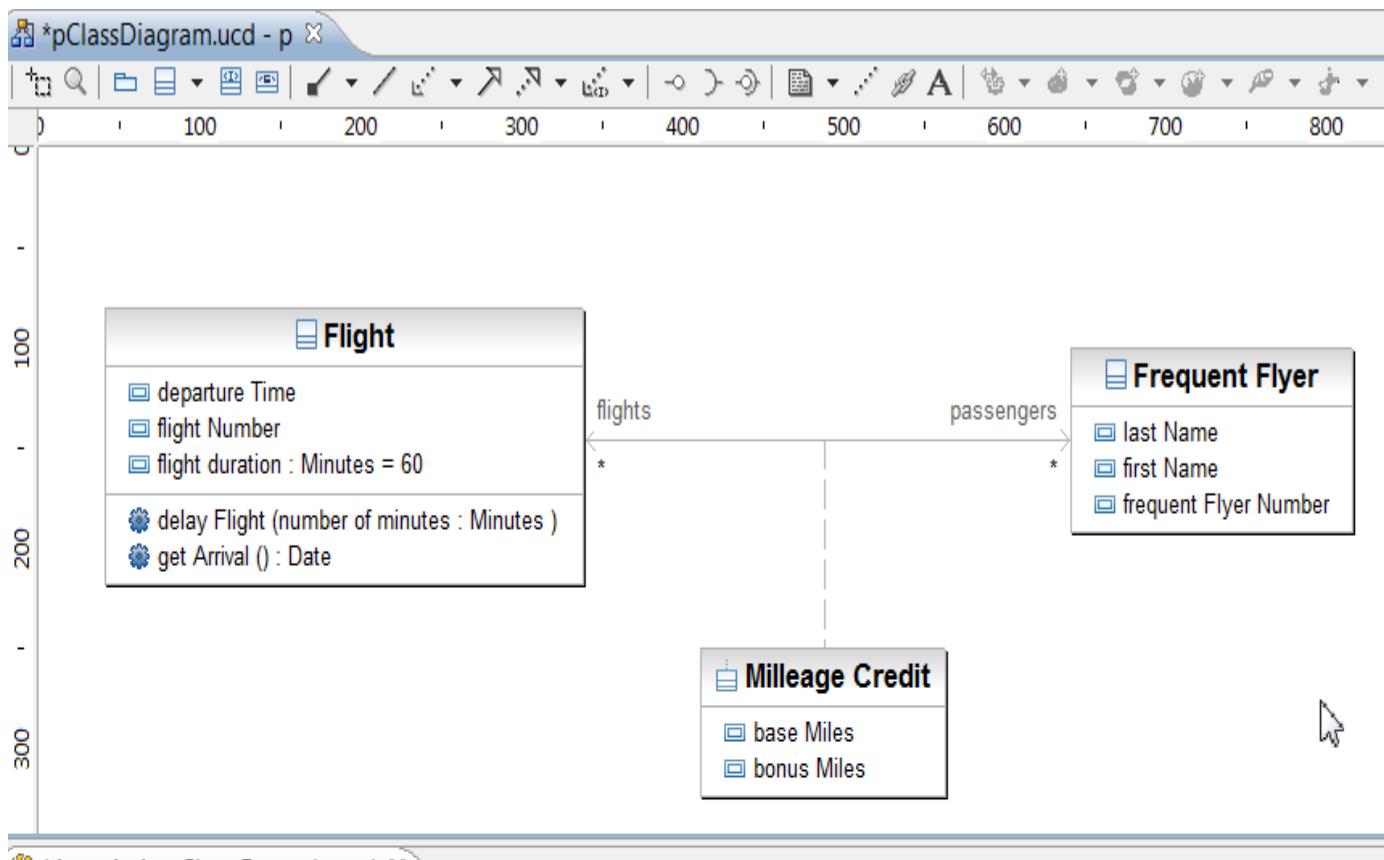
The screenshot shows the UML tool interface with two main windows. On the left is the 'Property' view, which displays various properties for a selected classifier. The 'Name' property is set to 'Flight'. On the right is the 'XMI Editor' showing a package named 'p' containing a class named 'Flight'.

Property	Value
UML	
Classifier Behavior	
Client Dependency	
Is Abstract	false
Is Active	false
Is Leaf	false
Name	Flight
Owned Port	
Powertype Extent	
Redefined Classifier	
Representation	
Template Parameter	
Use Case	
Visibility	Public

Add a property to your metamodel Class using the Class contextual menu > New > Property. (Note that you can add the name of the property with no syntax control using the Property View).

The screenshot shows the UML tool interface with a class named 'Flight' selected. A contextual menu is open, and the 'Property' option is highlighted.

Here is the created diagram and the metamodel view of the created class diagram.



Format

This section describes how to use the format menu.

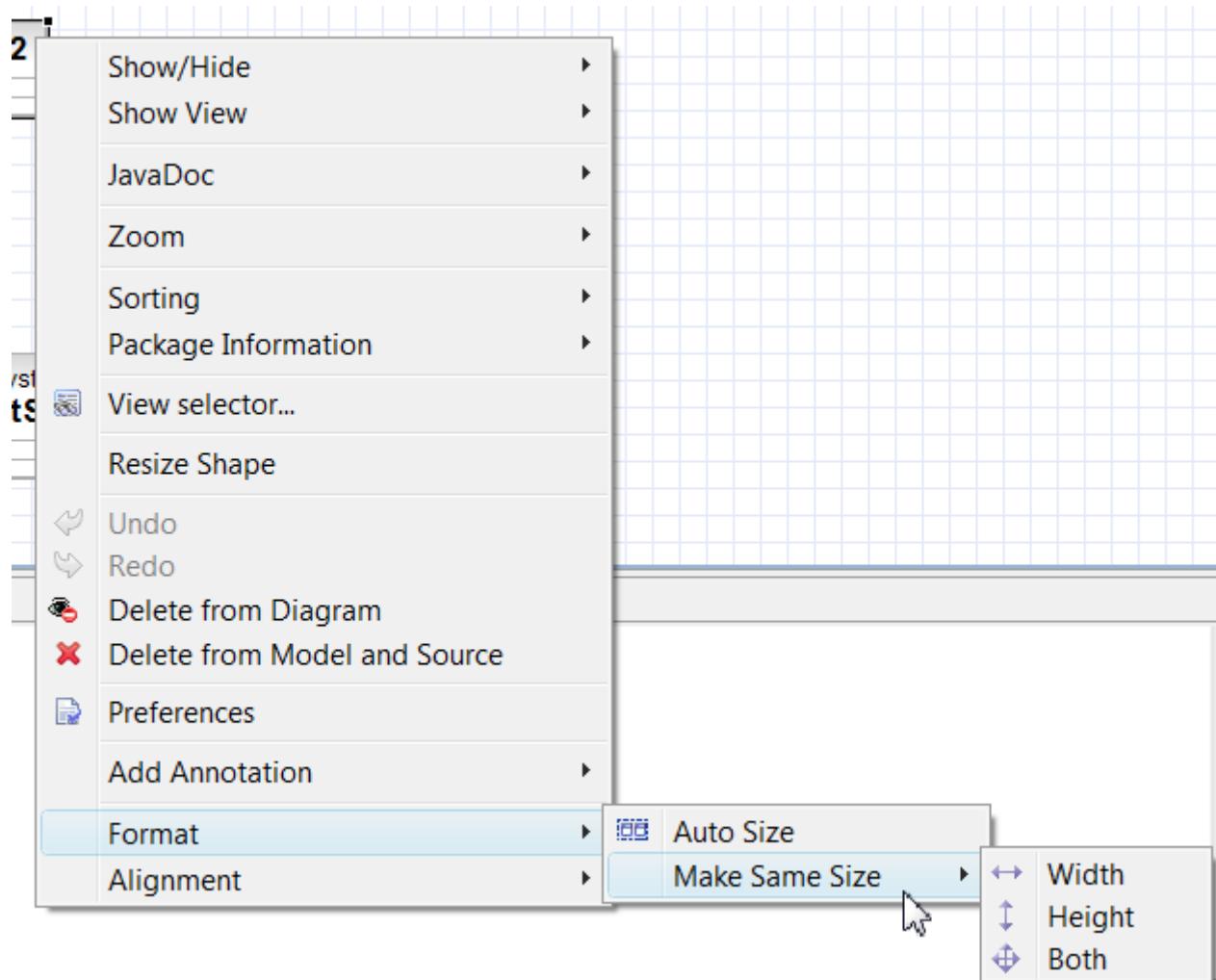
The format menu allows you to change the size of a group of elements.

This menu is only available if you select two or more elements.

To select a group of element click with the mouse on each graphical element while keeping the ctrl key pressed.

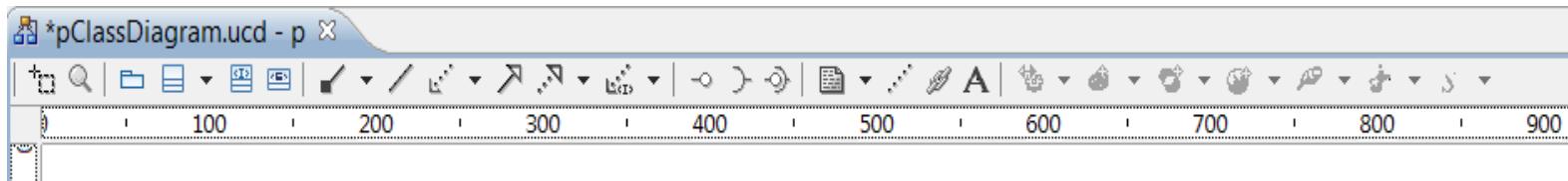
The following Format features are possible:

- **Auto Size** menu will give the same size as the last selected classifiers. Each group of classifiers (*e.g. class, interface and enumeration*) is differentiated.
- **Make same size** will allow to have the same **width**, **height** and **both** as the last selected classifiers.



Toolbar items

This section describes the tools provided in order to build a Class Diagram.



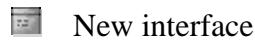
You can create the following elements:



Create a new package in the diagram editor



Create a new class



Create a new interface



Create a new association between two elements

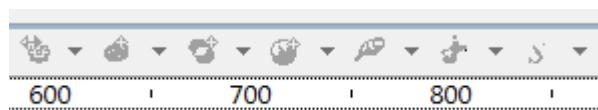


Create generalization between two elements



Create dependency between two elements

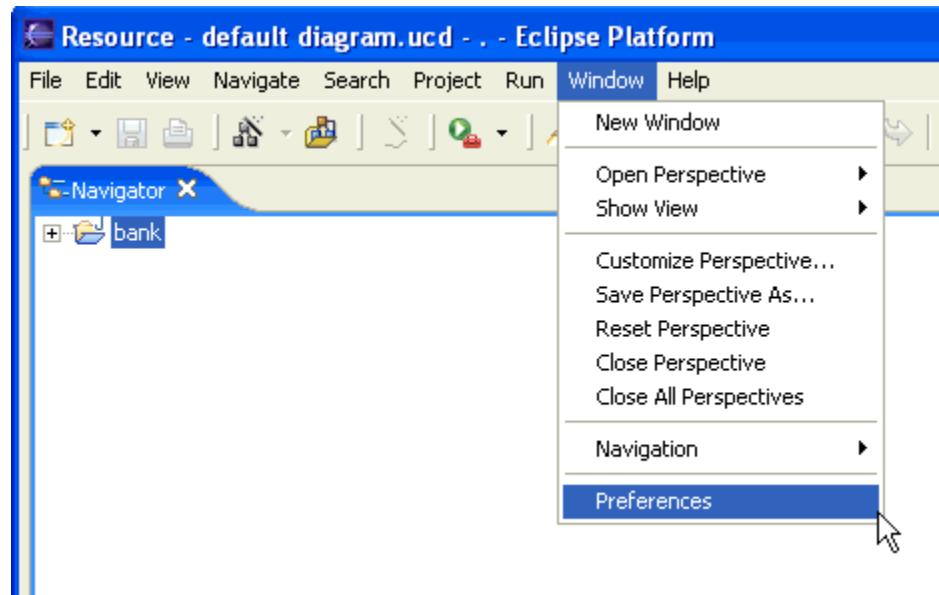
PSM elements



Class Diagram Example

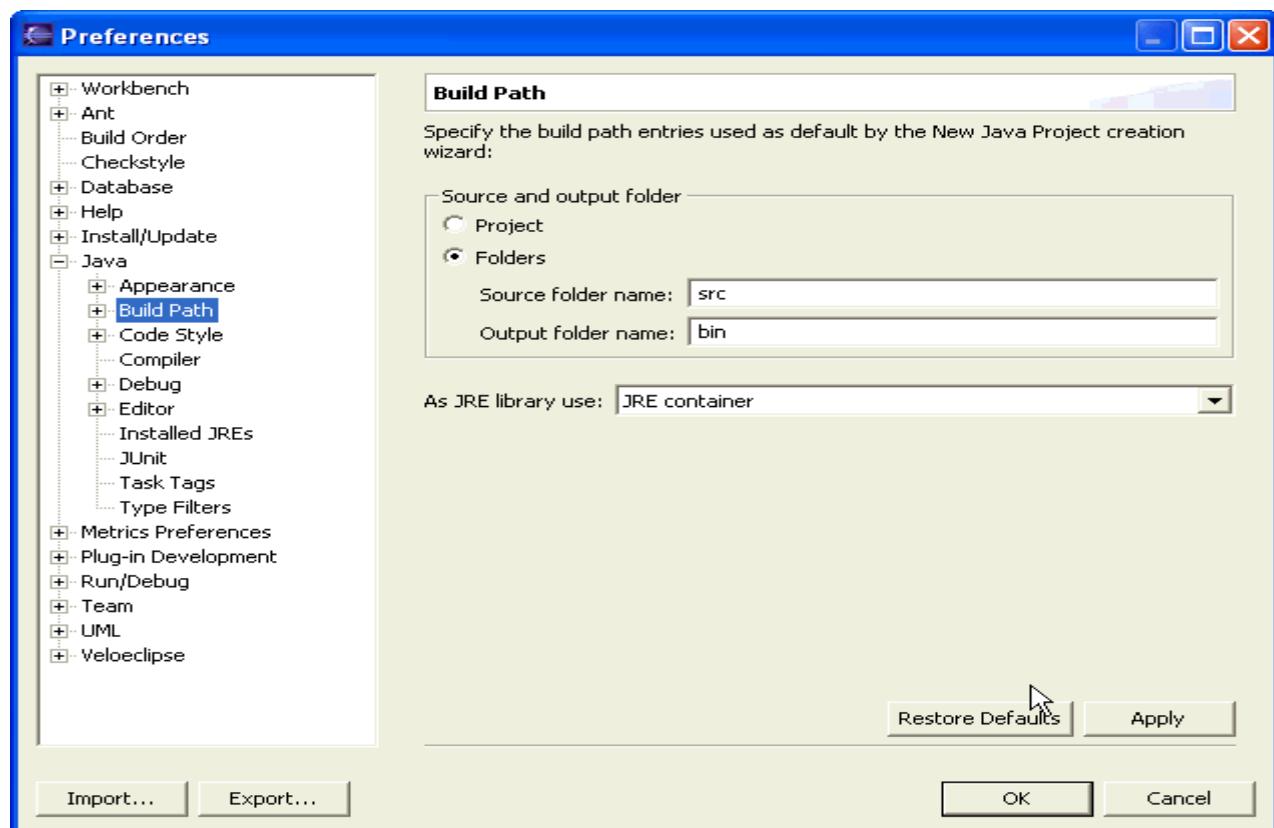
The following example shows a class diagram creation using Eclipse 2.1.

We are going to add general Eclipse Preferences, which will be used in our example.
Click on **Window>Preferences**.



We decided to create a src and a bin output for this project.

Select **Java>Build Path** and click on Folders checkbox.



We are going to set up general UML preferences.

Click on UML; wait for the launch of EclipseUML.

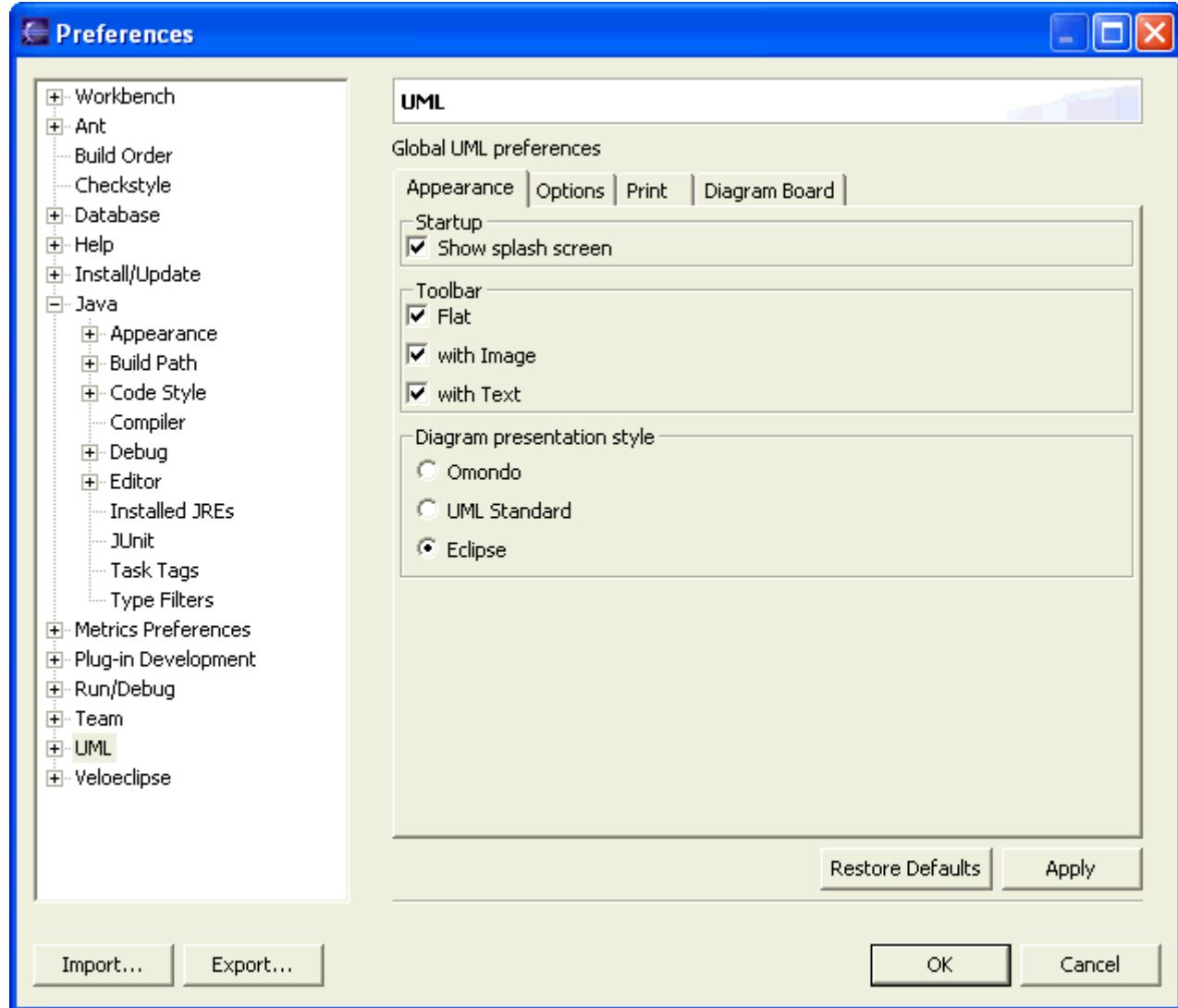
Select the following options:

Toolbar:

- Flat
- with image
- with text

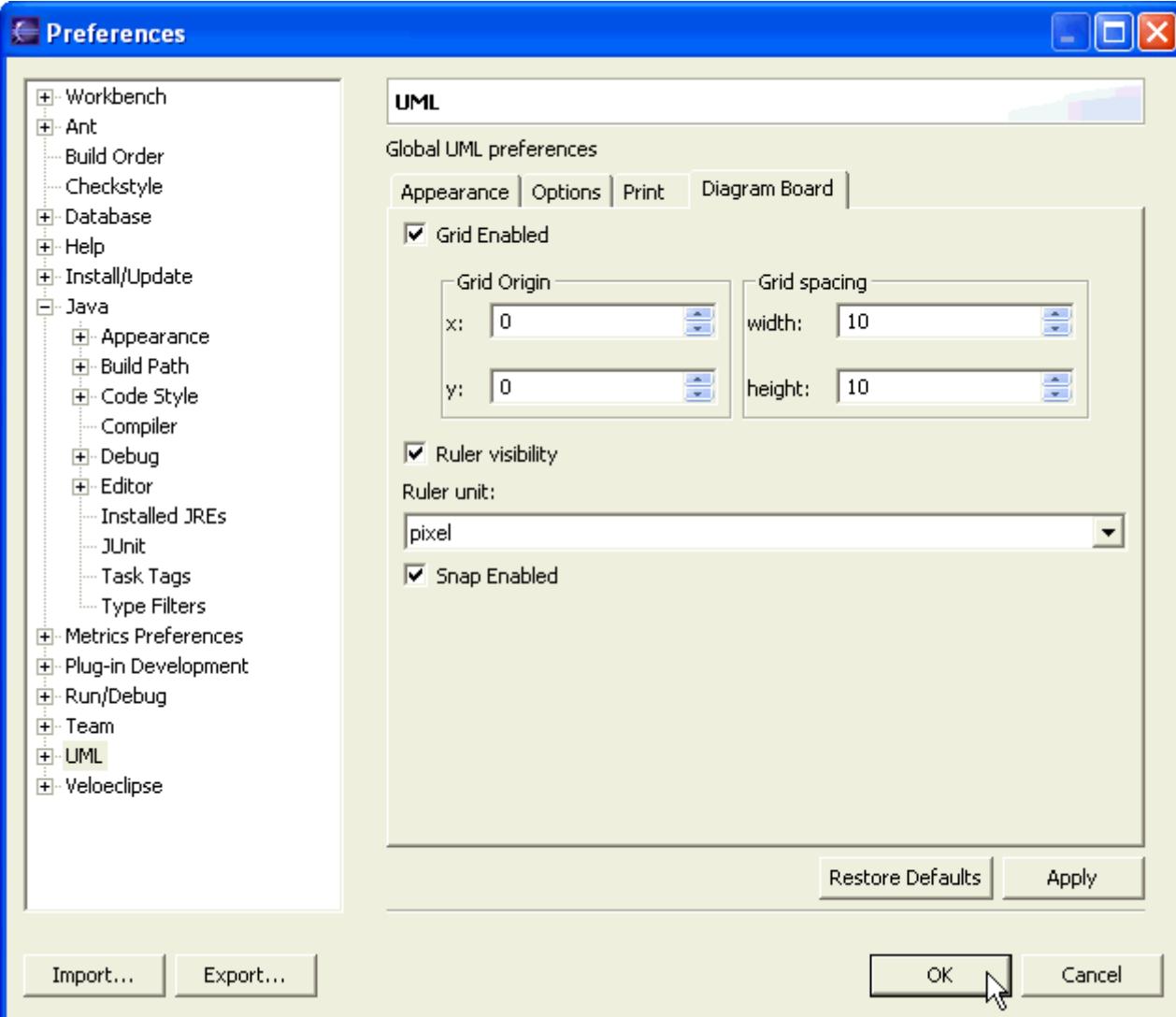
Diagram Presentation:

- Eclipse



Select Diagram Board menu to activate the following options:

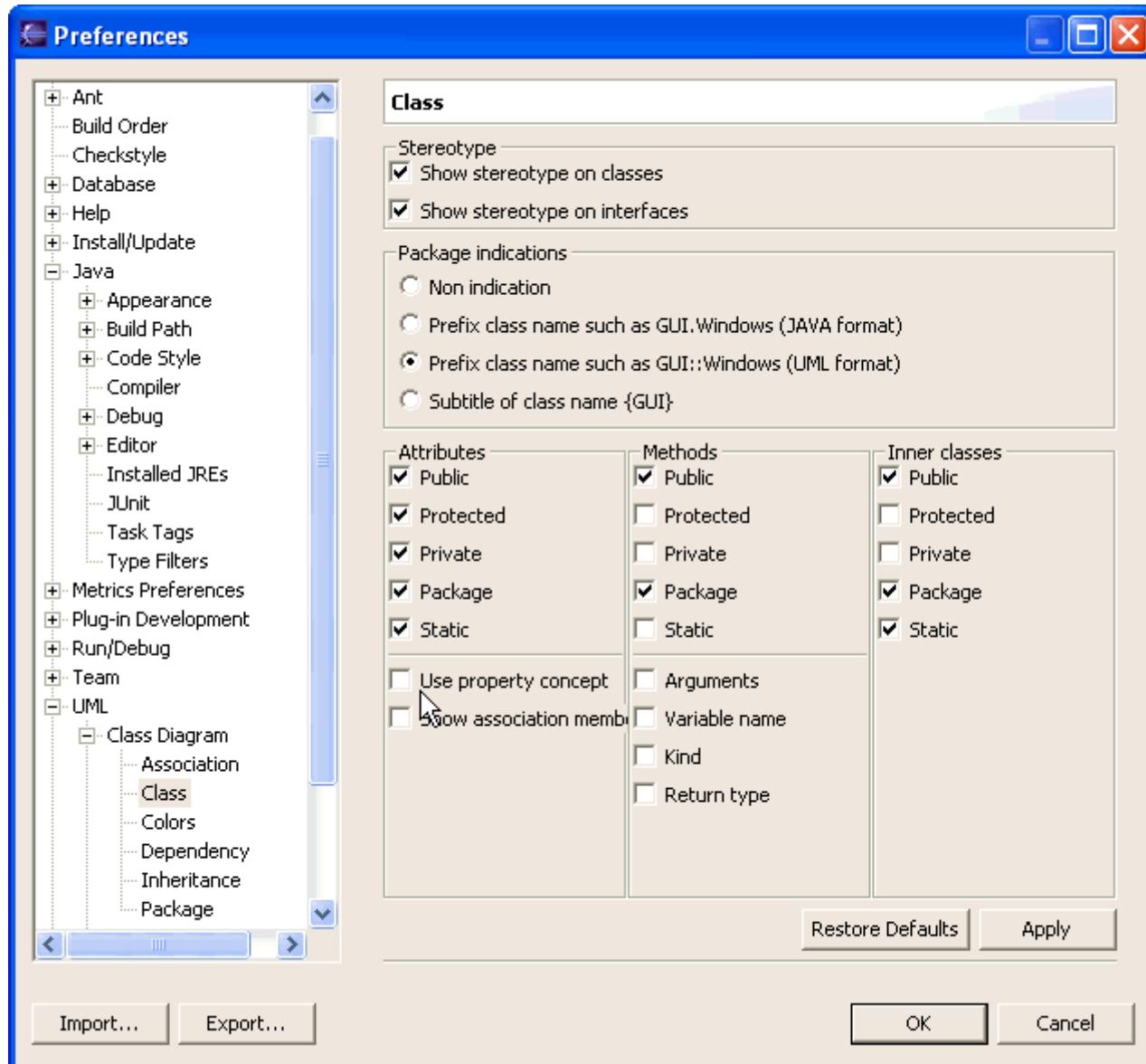
- Grid Enabled
- Ruler visibility > pixel
- Snap Enabled



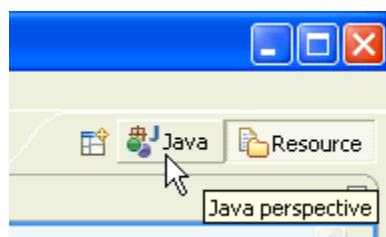
We want to show Public, Protected, Private, Static and Package Attributes.

We also want to show Public and Package Methods.

We unselect the Use property concept checkbox.

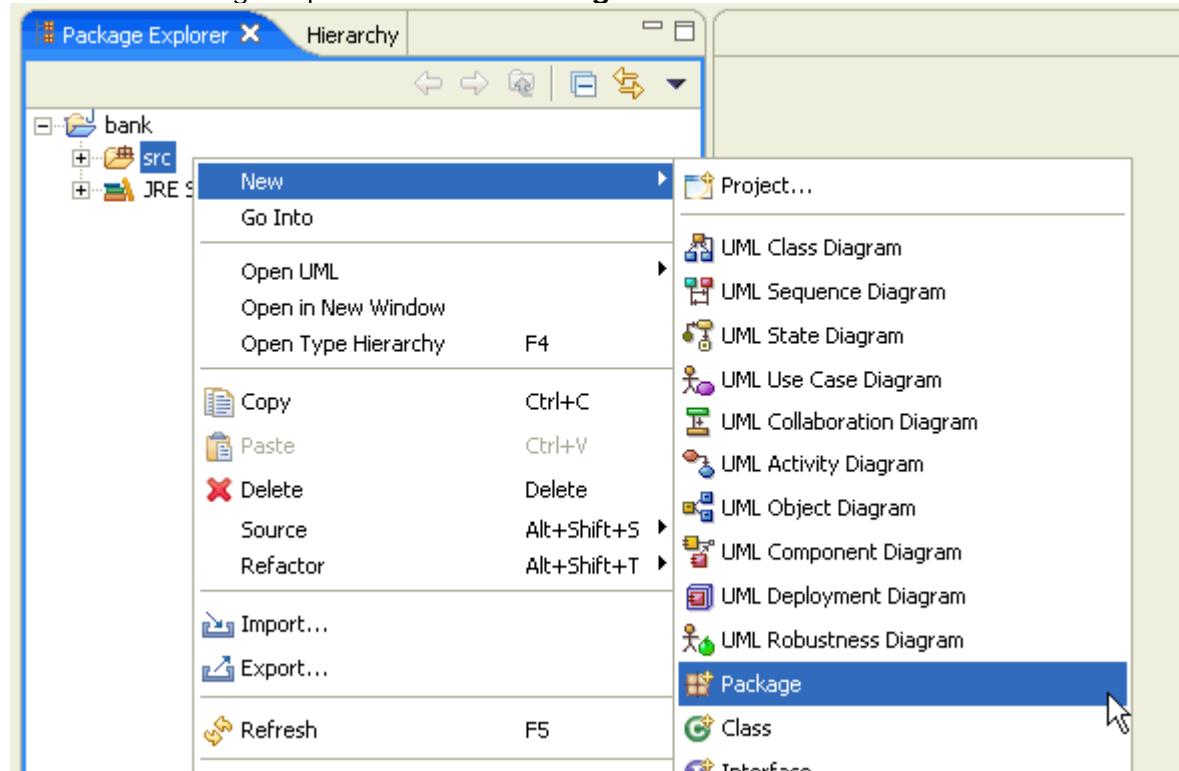


We would like to use a Java Perspective.
Click on the top icon to **select Java perspective**.

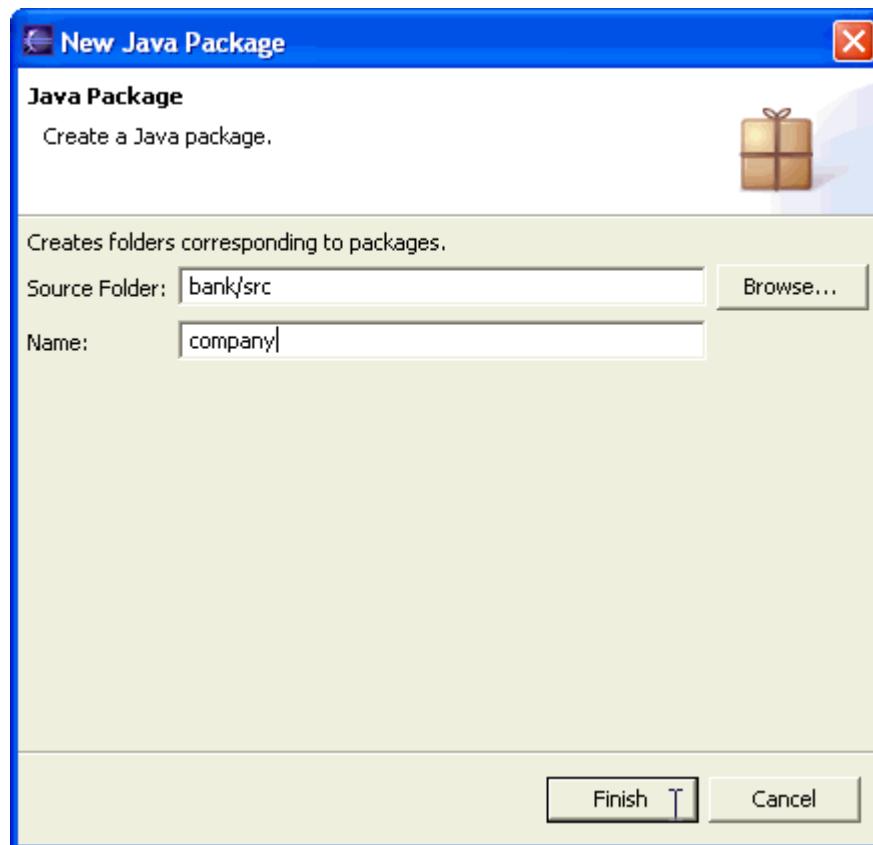


We are going to create a new Java Package.

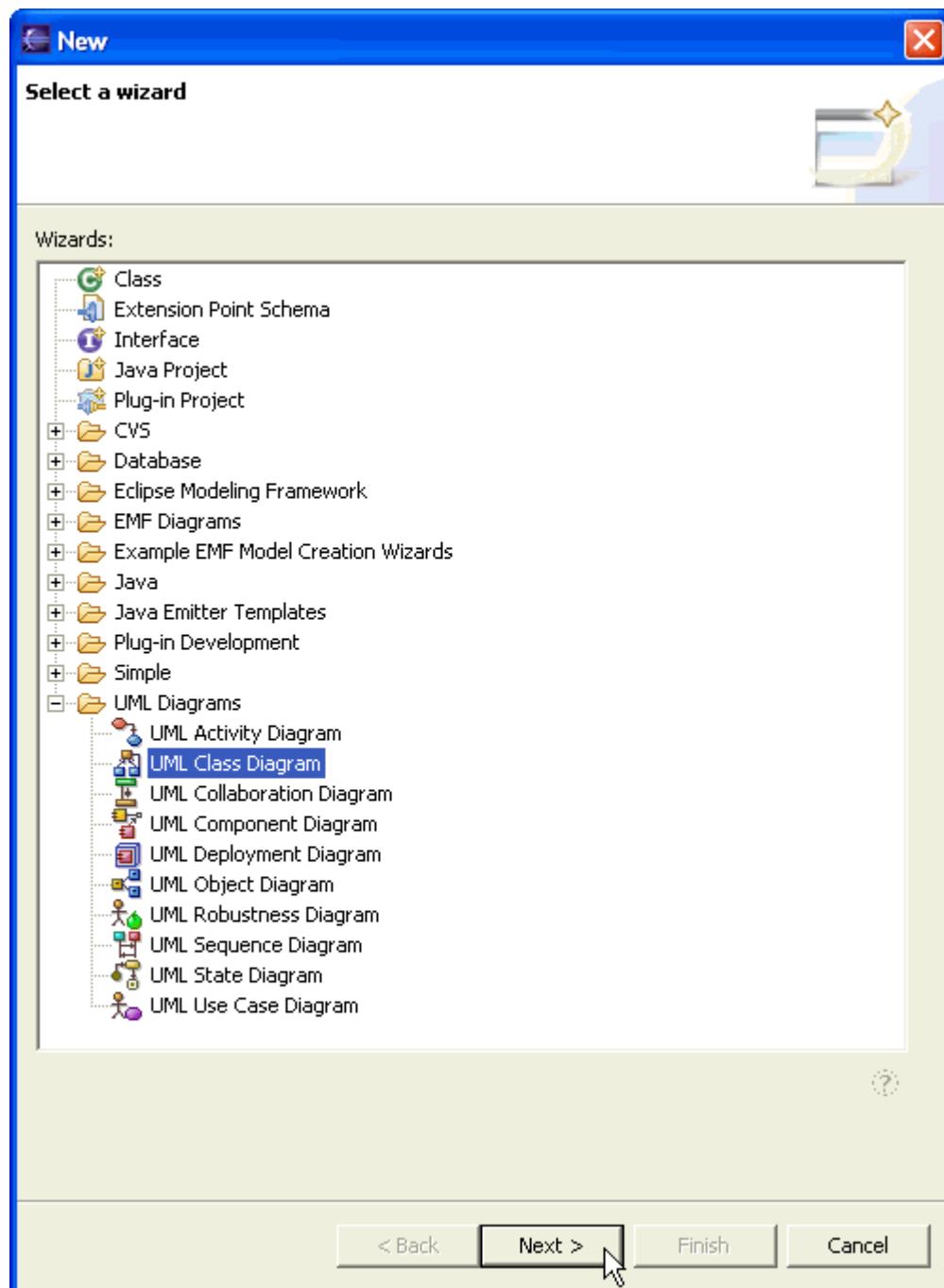
Select src in the Package Explorer **New > Package**.



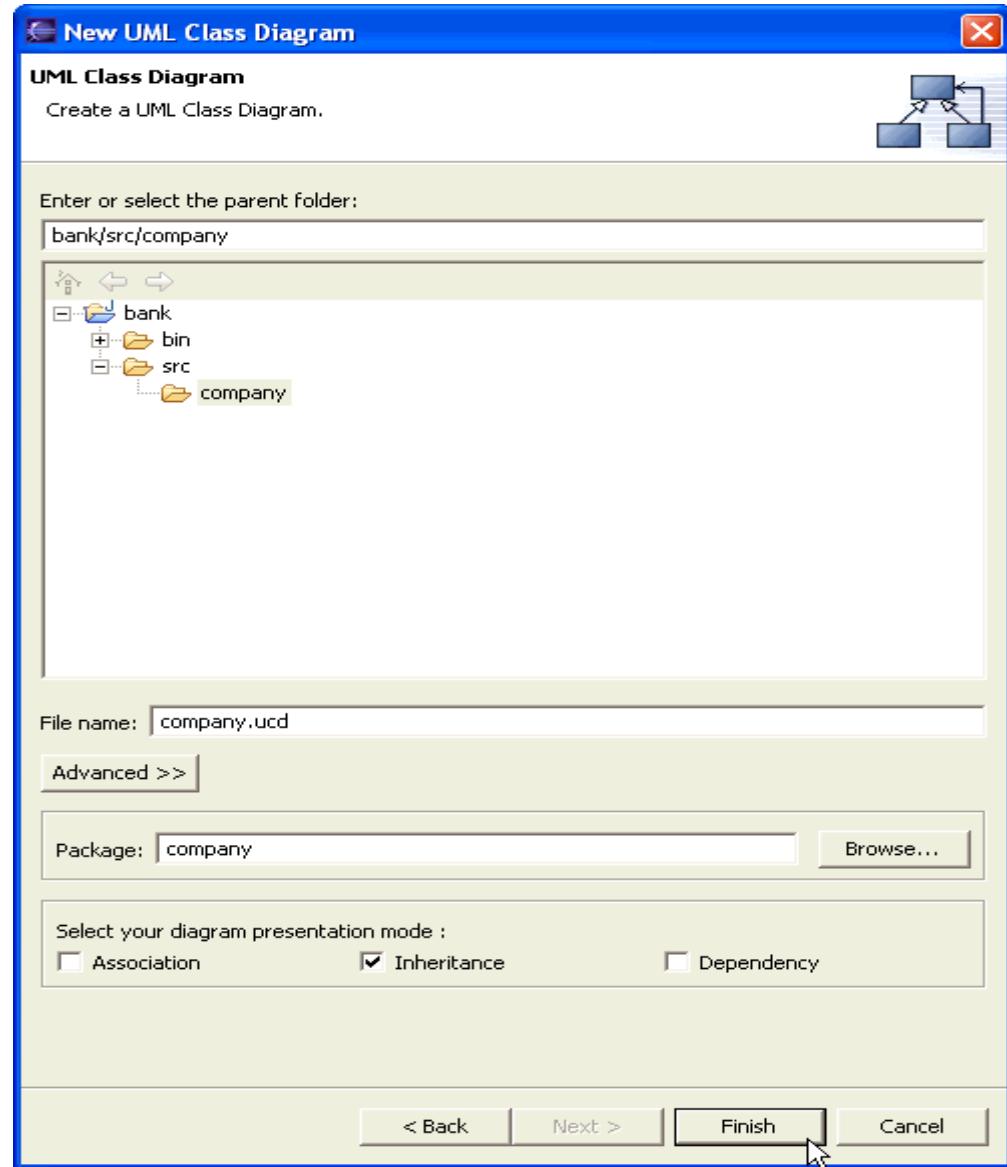
Enter the name of the Package.



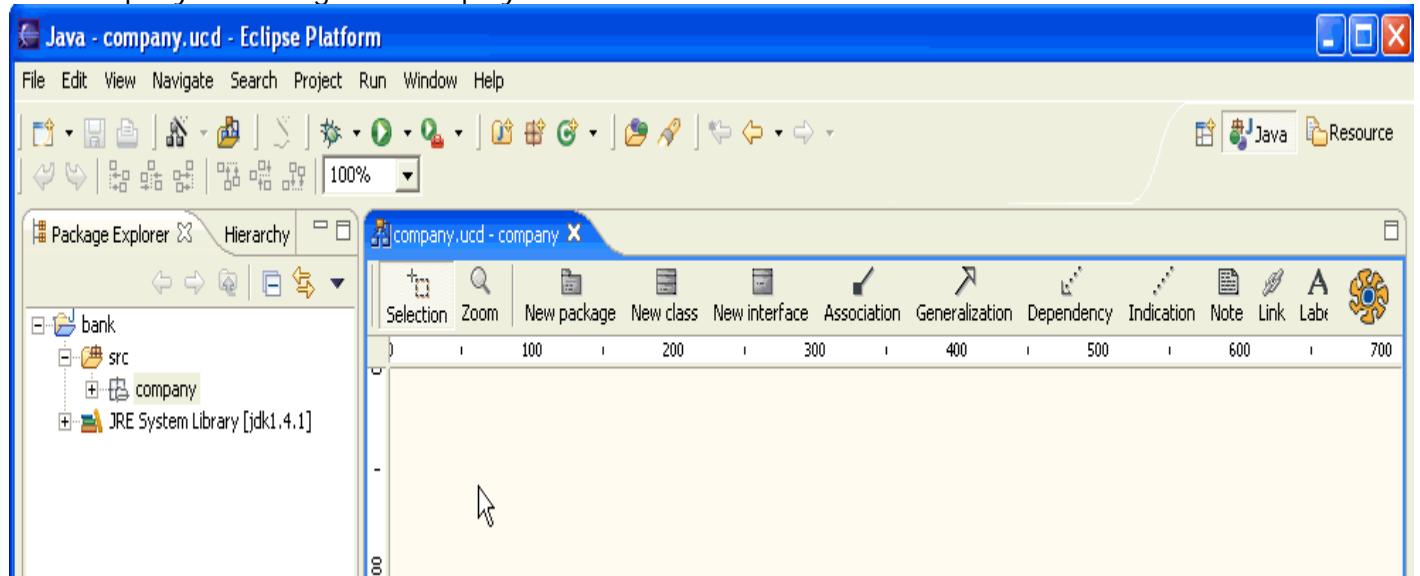
Select company package in the Package Explorer, open the popup menu **New > Other**
Select **UML Diagrams > UML Class Diagram** then click on the next button.



Enter the name of the class diagram file in the File name field.
Enter company, the ucd (UML Class Diagram) extension is related to the class diagram.
Click on the finish button.

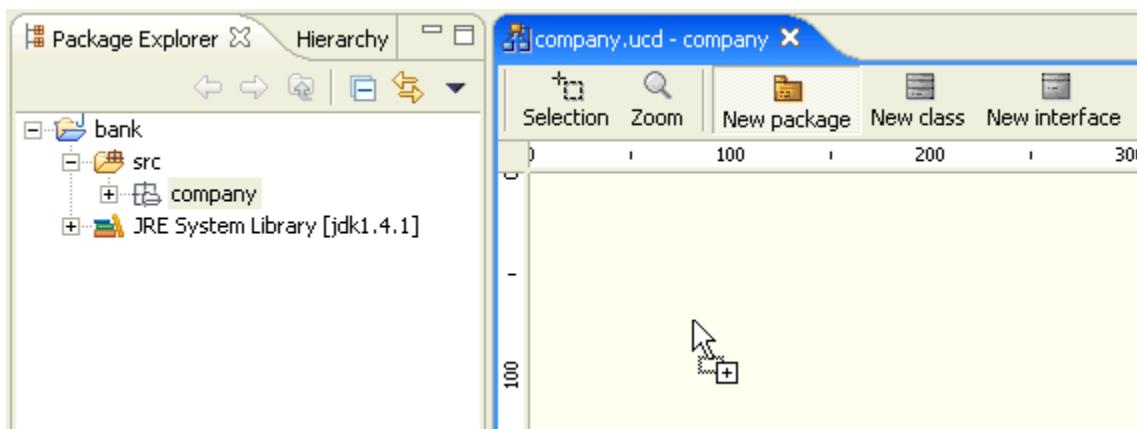


The company class diagram is displayed.

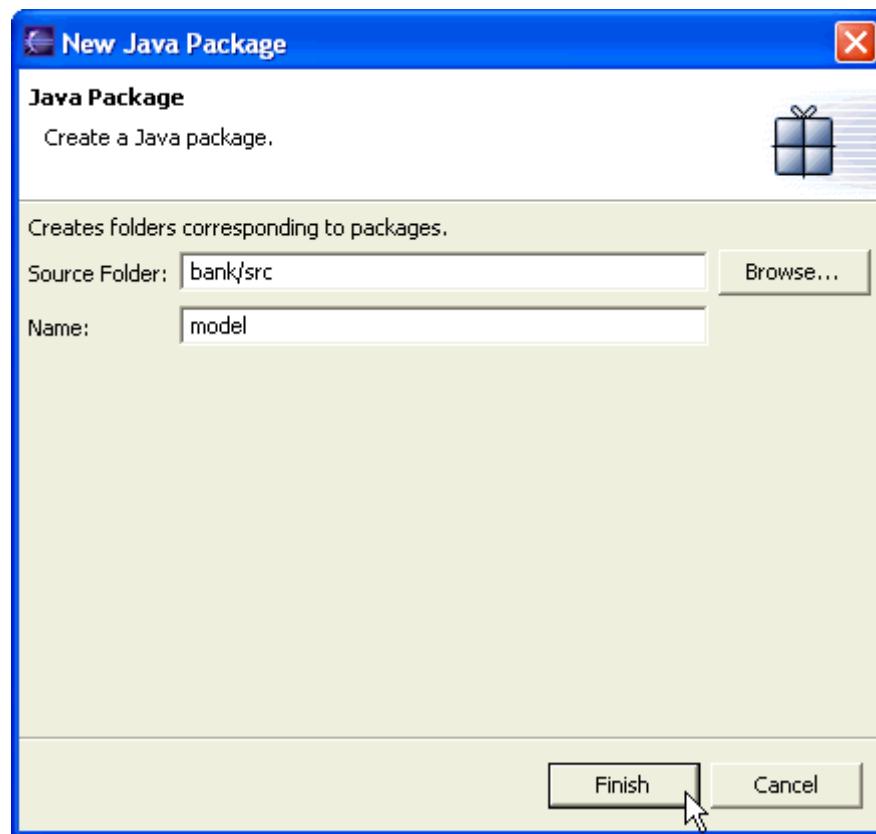


Create a New Package.

Select new package in the toolbar and click on the diagram.

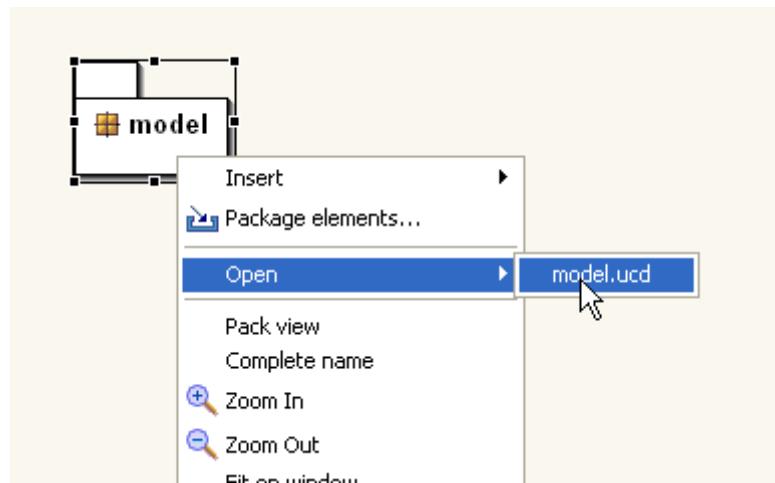


Enter the name of the package in the Name field.

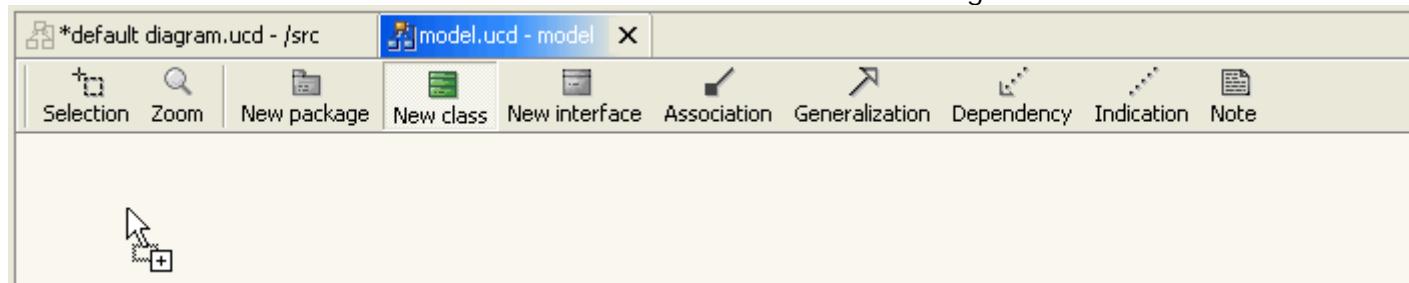


Create a new Class Diagram inside the model Package.

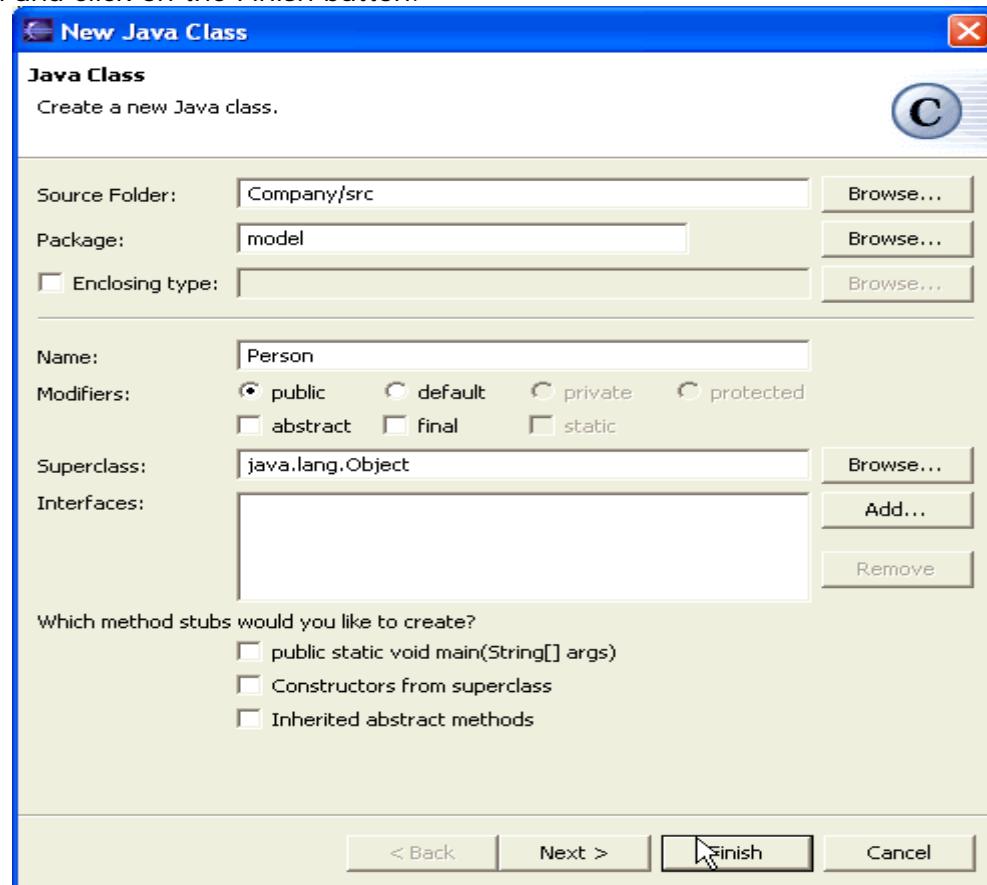
Click on model>Open>model.ucd (ucd is the class diagram extension).



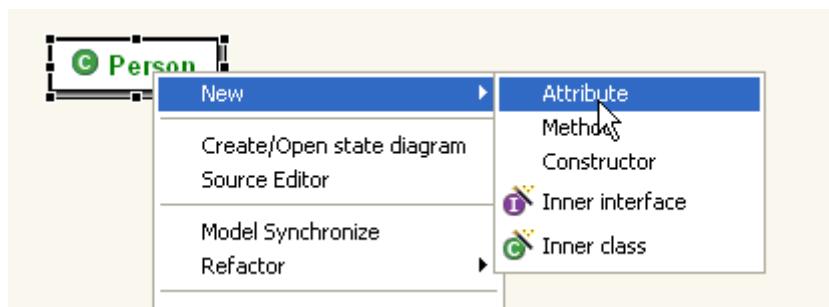
Create a New class. Select New class in the toolbar and click on the diagram.



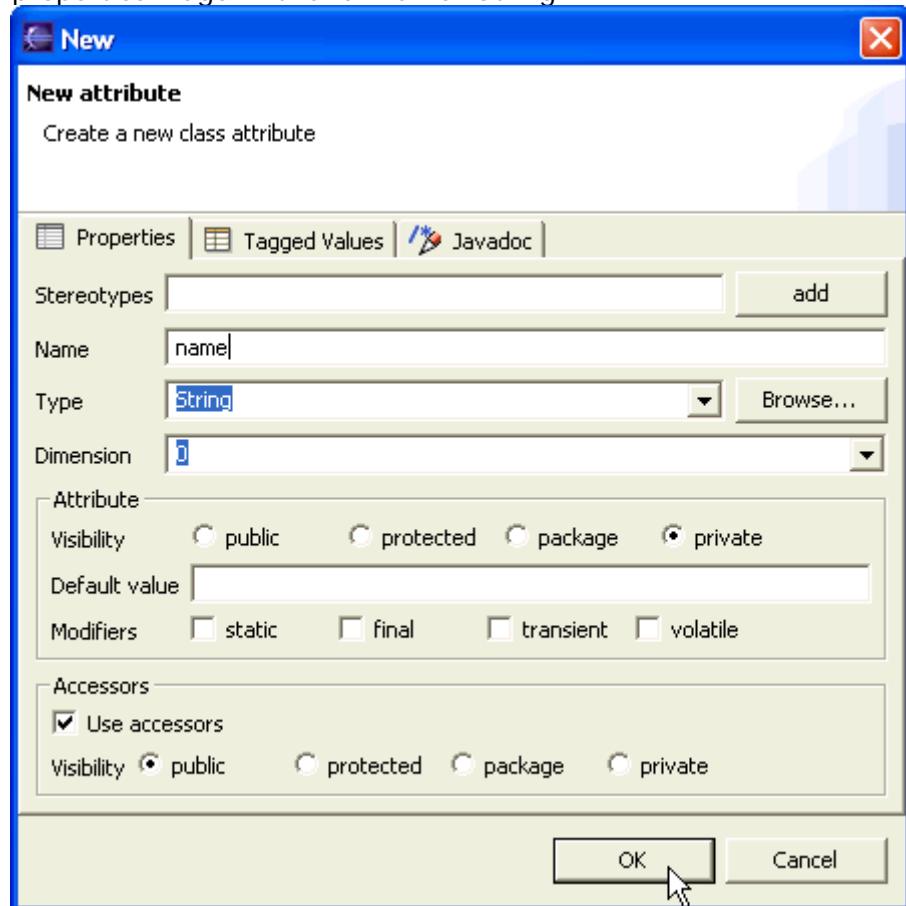
Enter Person and click on the Finish button.



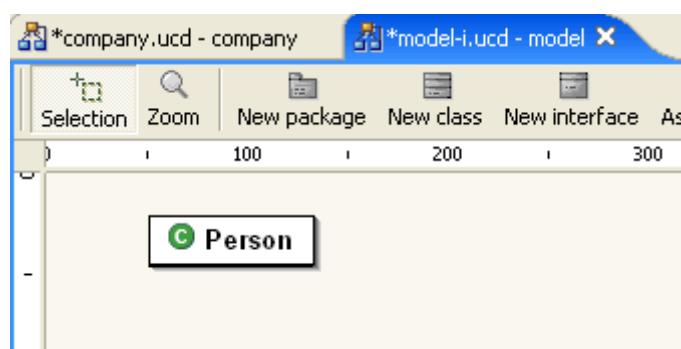
We are going to add two new Attributes.
Click on Person>New>Attribute.



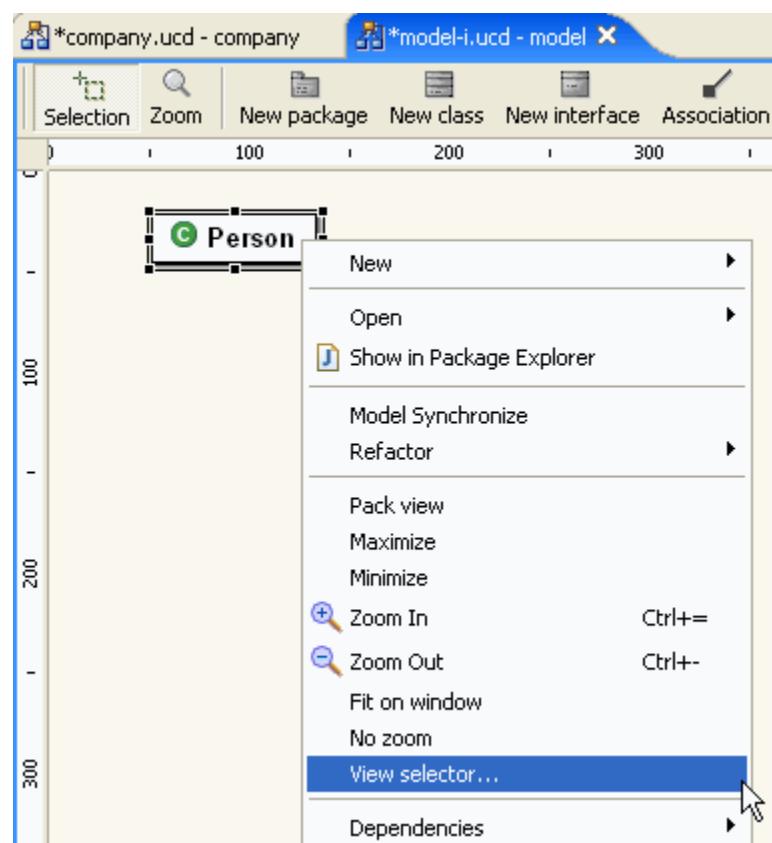
Enter Attributes properties: "age: int" and "name: String".



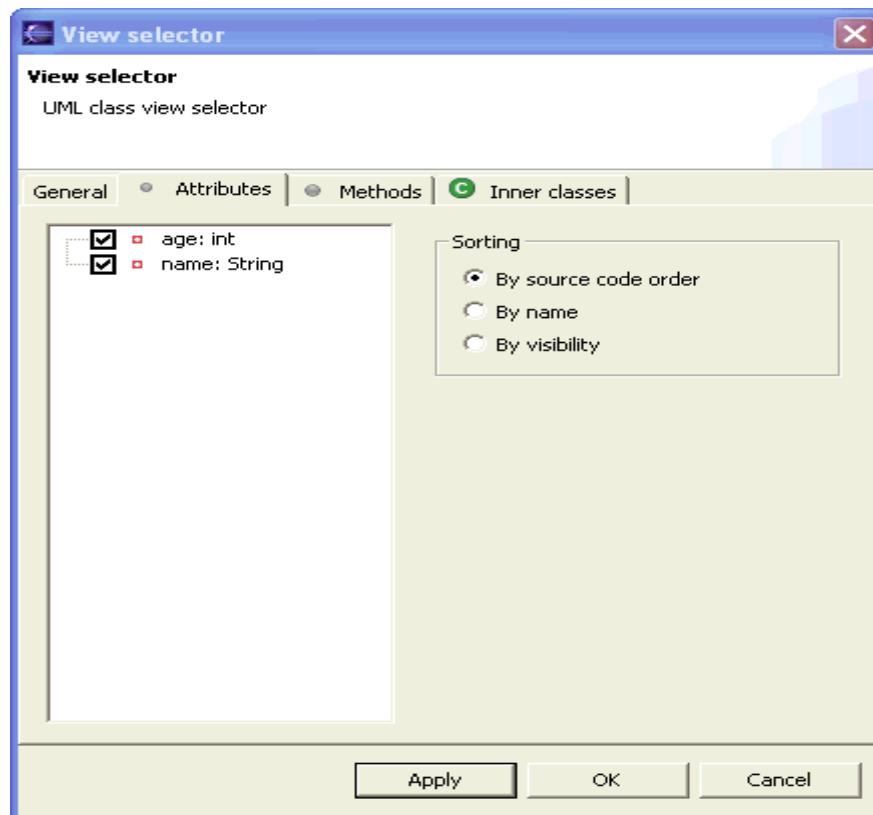
We discover that attributes are not visible in the class diagram editor.



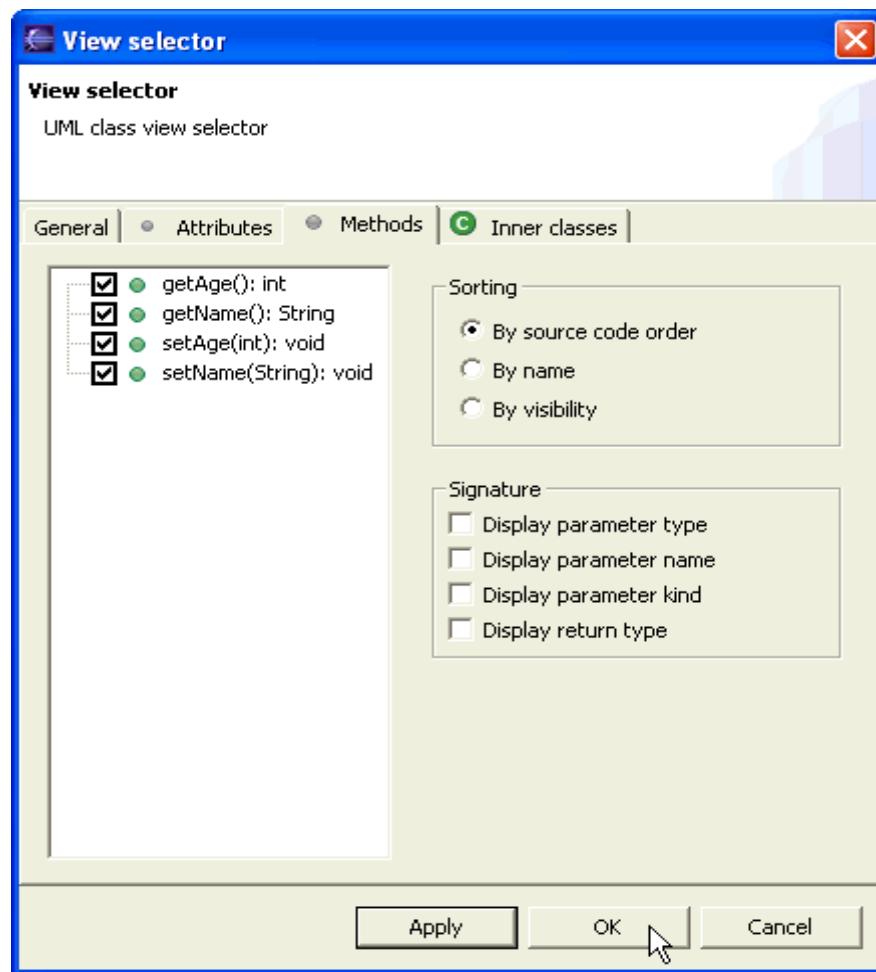
We want to show attributes and Methods in the Class diagram.
Select a class and open the popup menu then **select View selector**.



Select Attributes tab and attributes checkbox in the left pane.



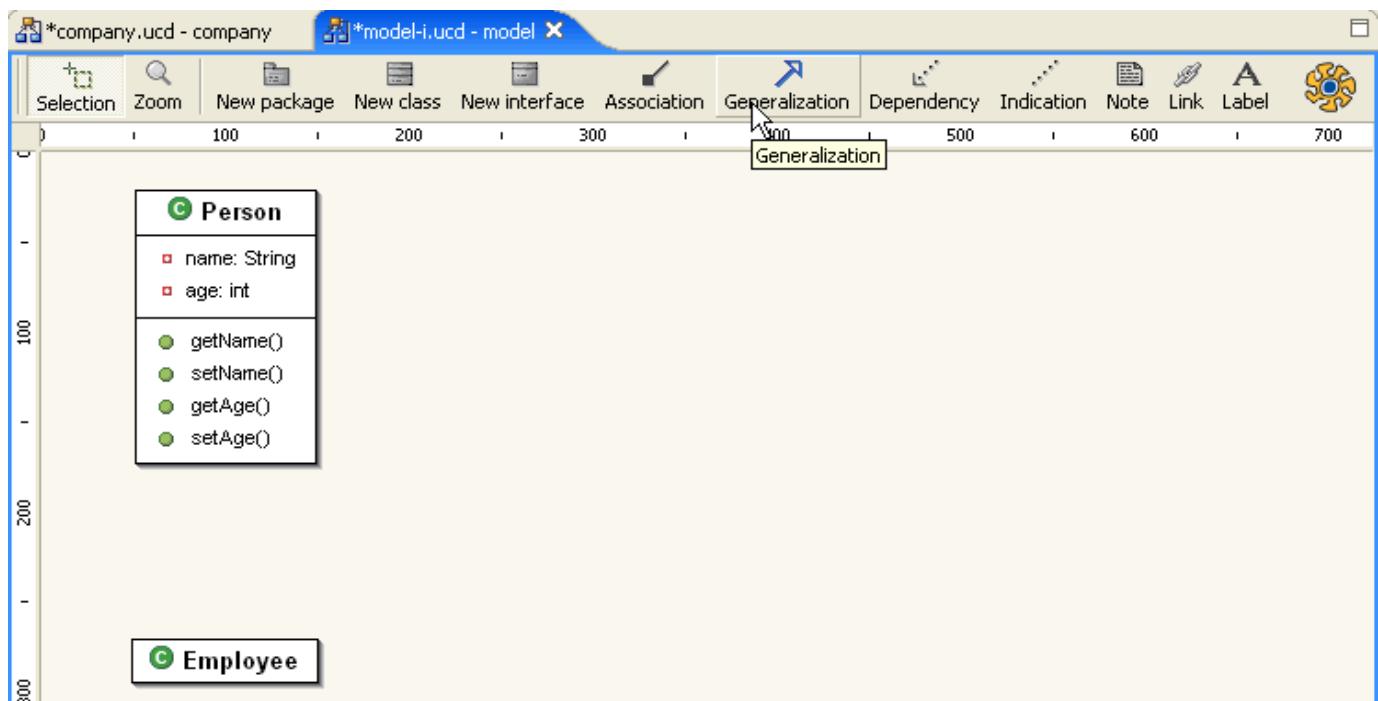
Select Methods tab and Methods checkbox in the left pane.



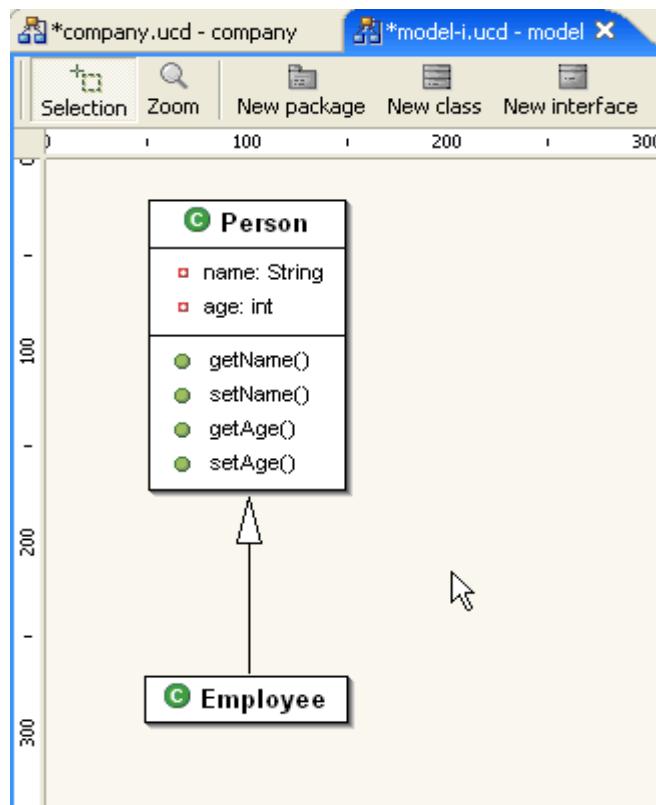
Create a New class named Employee.

We would like to add inheritance from Employee to Person.

Select Generalization in the Toolbar.

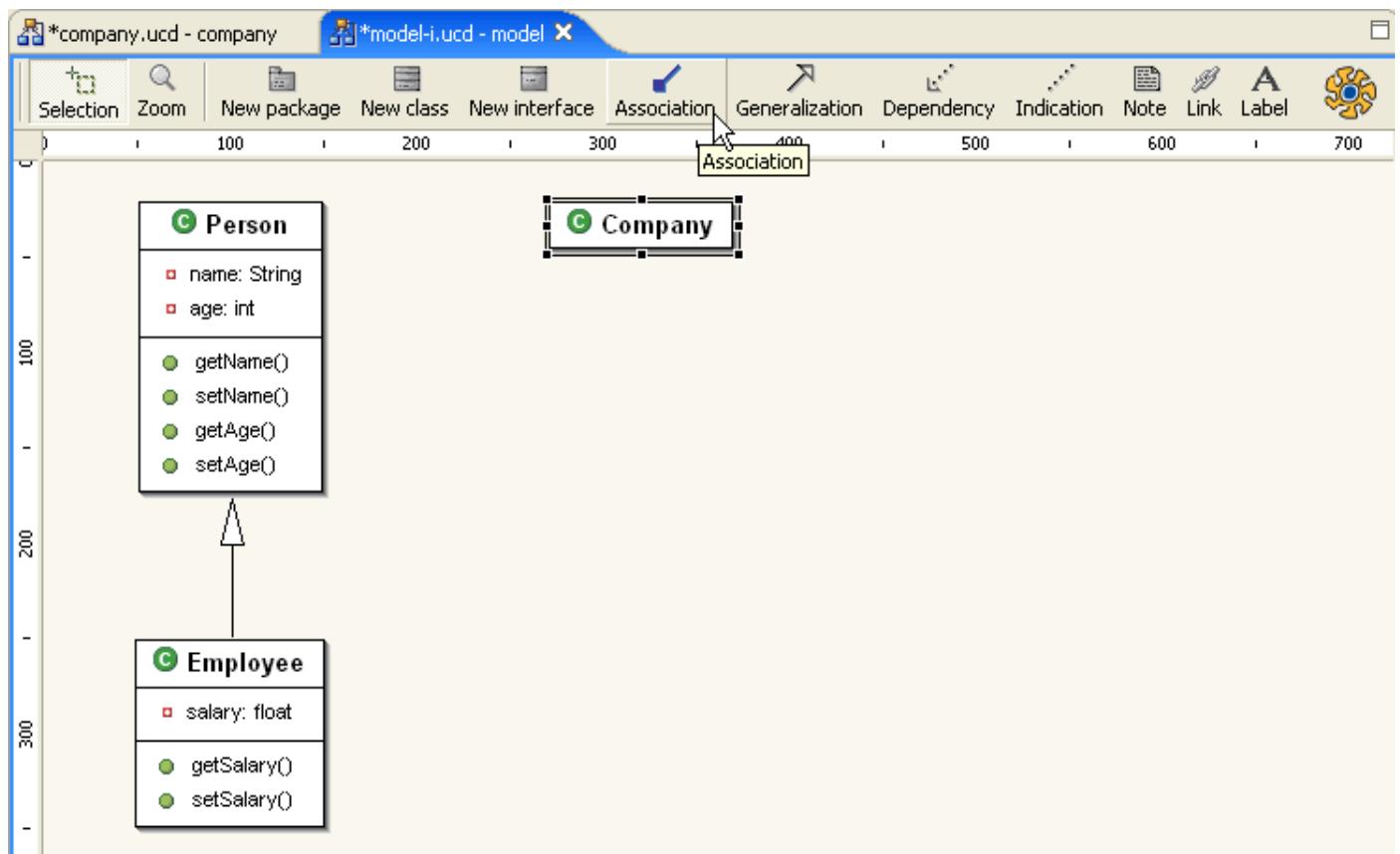


Drag the Inheritance from Employee to Person.

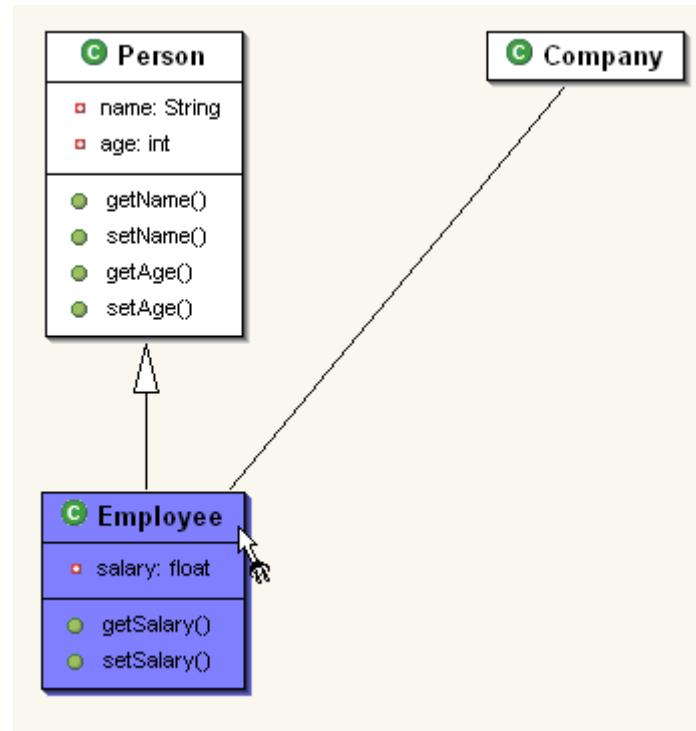


Add "salary: float" attribute to Employee.

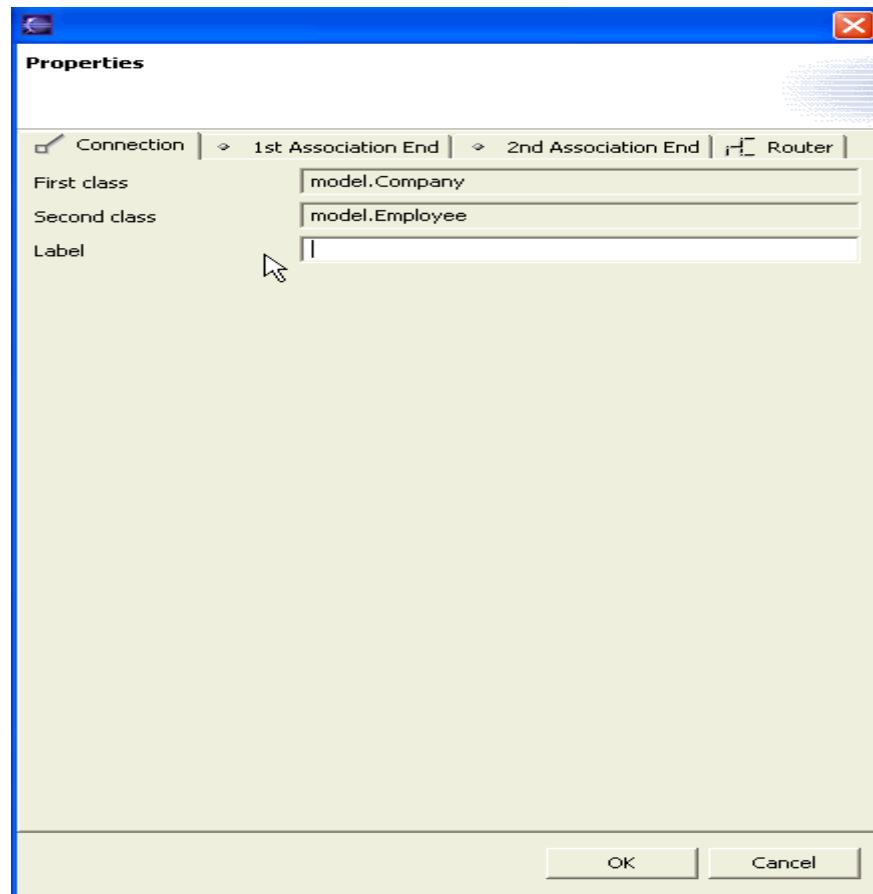
Create a New class named Company.



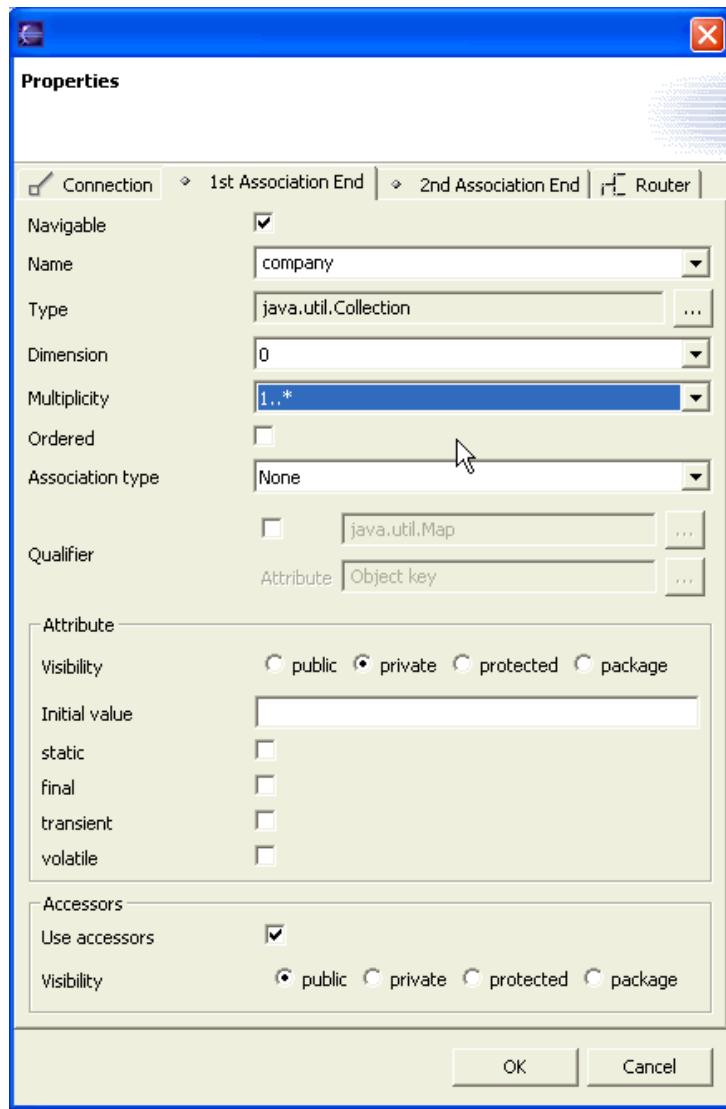
We want to create a new Association between Company and Employee.
Select Association in the Toolbar then drag from Company to Employee.



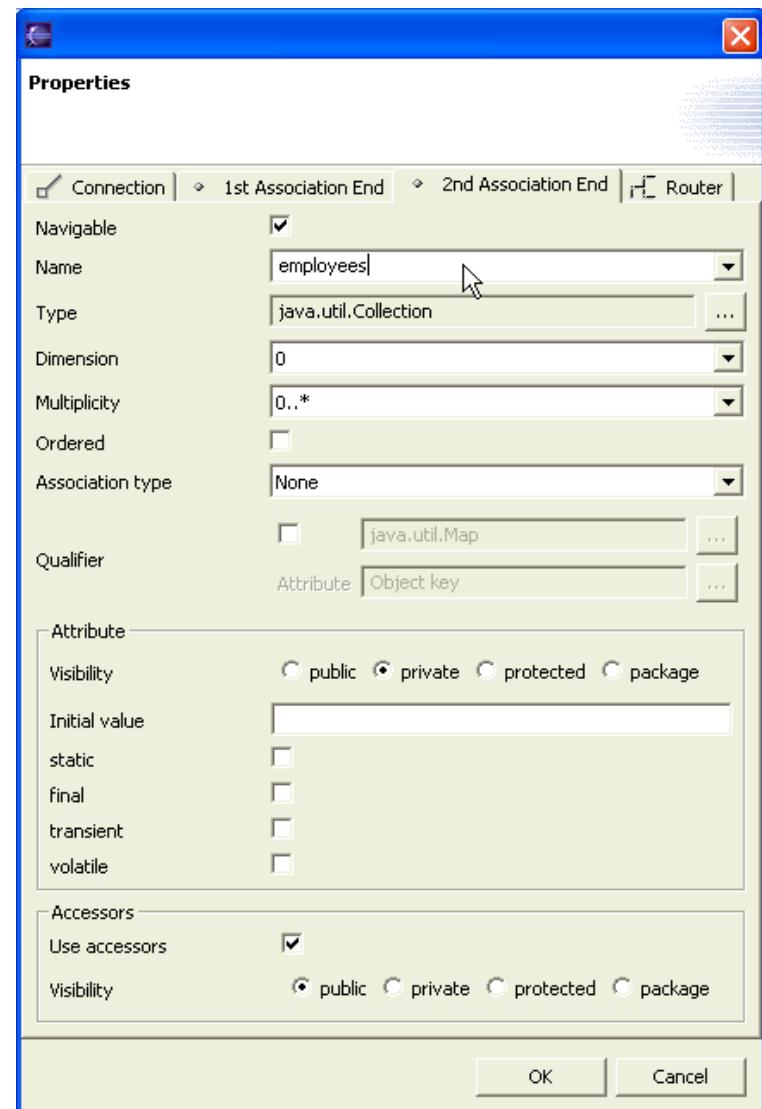
It is possible to add a name to the Label, but we will not in this example.



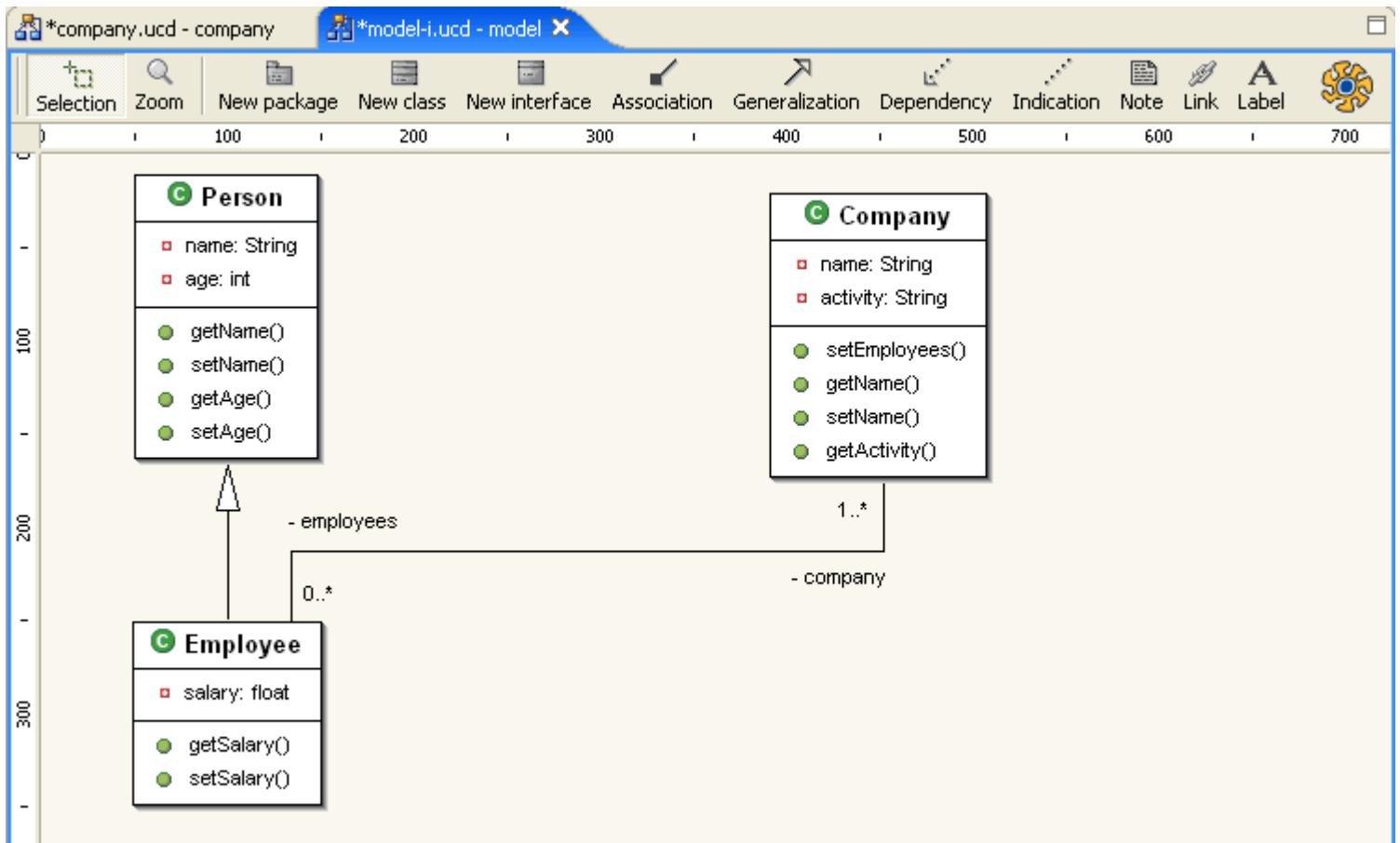
Enter the 1st Association End properties.
Select 1...* Multiplicity.



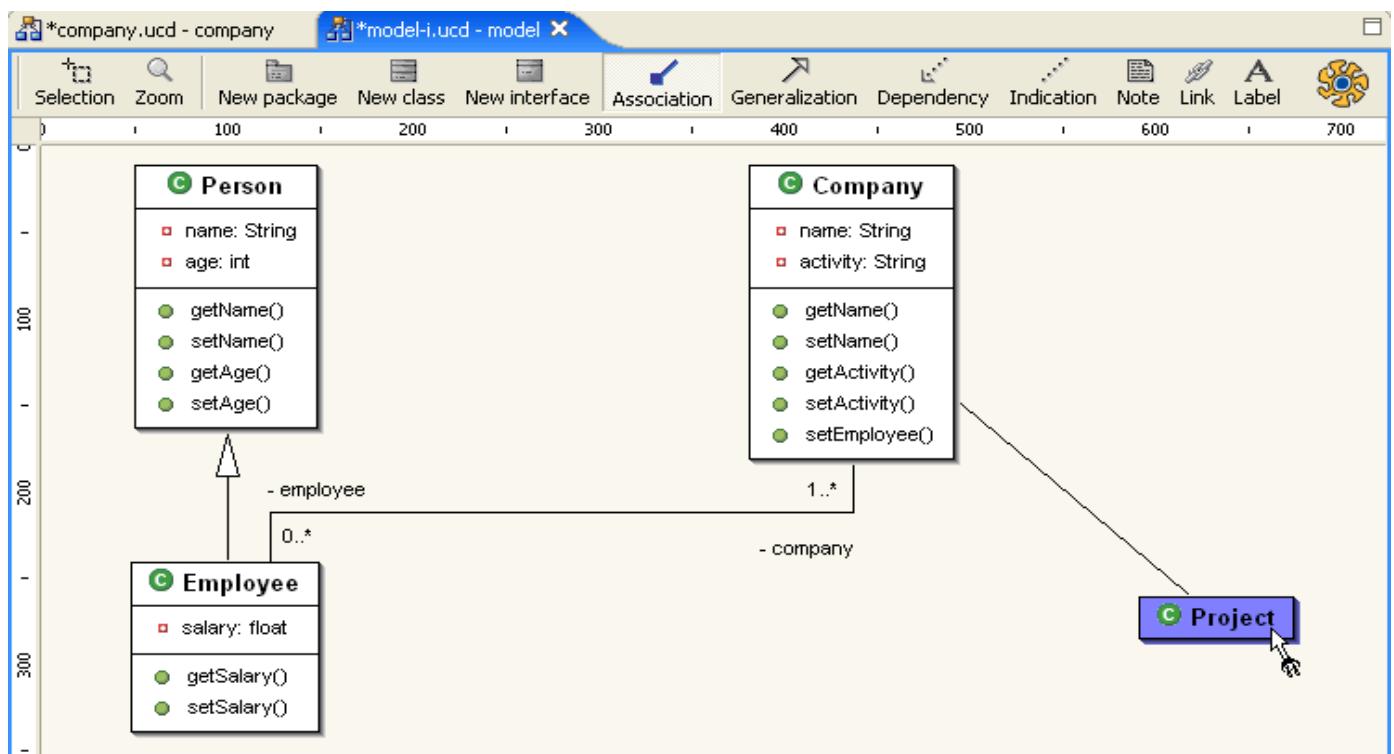
Enter the 2nd Association End properties.



Click on Company and Create two new Attributes: "name: String" and "activity: String".



Create a New class named Project and add an association between Company and Project.



Enter Association Properties.
Label: leave blank.

Properties

<input checked="" type="checkbox"/> Connection	1st Association End	2nd Association End	<input checked="" type="checkbox"/> Router
First class	model.Company		
Second class	model.Project		
Label	<input type="text"/>		

Enter the 1st Association End properties.

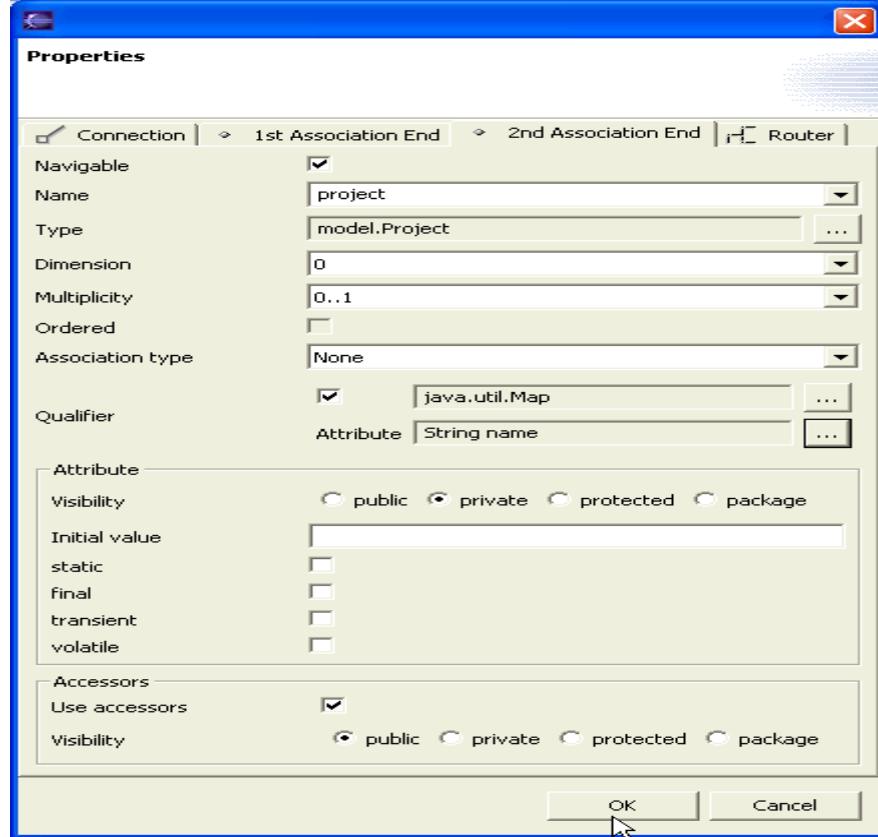
Properties

<input checked="" type="checkbox"/> Connection	1st Association End	2nd Association End	<input checked="" type="checkbox"/> Router	
Navigable	<input checked="" type="checkbox"/>			
Name	company			
Type	model.Company			
Dimension	0	<input type="button"/>		
Multiplicity	1	<input type="button"/>		
Ordered	<input type="checkbox"/>			
Association type	None			
Qualifier	<input type="checkbox"/>	java.util.Map	<input type="button"/>	
Attribute	Object key			
Attribute	<input type="checkbox"/>			
Visibility	<input type="radio"/> public	<input checked="" type="radio"/> private	<input type="radio"/> protected	<input type="radio"/> package
Initial value	<input type="text"/>			
static	<input type="checkbox"/>			
final	<input type="checkbox"/>			
transient	<input type="checkbox"/>			
volatile	<input type="checkbox"/>			
Accessors	<input checked="" type="checkbox"/>			
Use accessors	<input type="checkbox"/>			
Visibility	<input checked="" type="radio"/> public	<input type="radio"/> private	<input type="radio"/> protected	<input type="radio"/> package
<input type="button"/> OK <input type="button"/> Cancel				

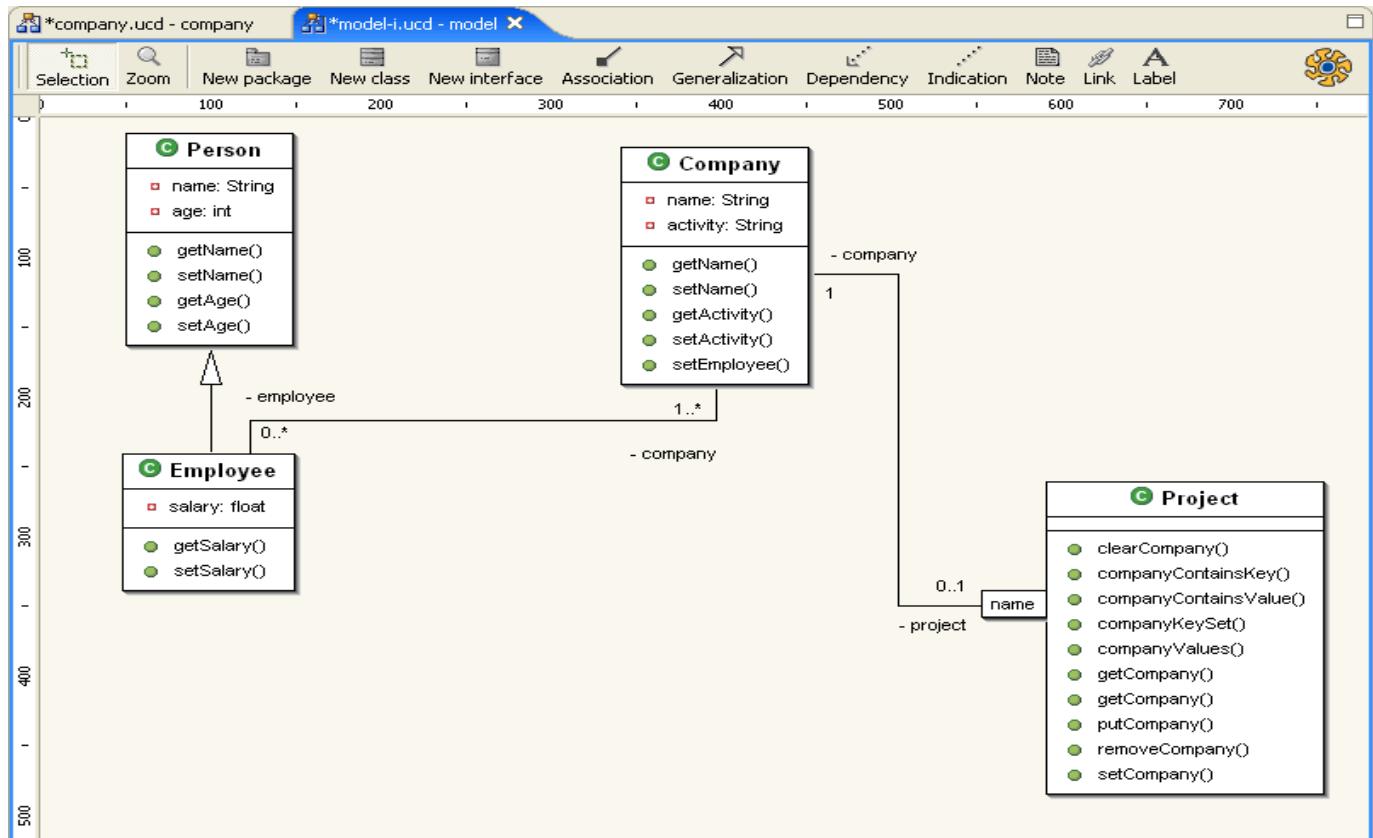
Enter the 2nd Association End properties.

Select Qualifier in the 2nd Association End.

Select the Qualifier checkbox and enter name in the name field and Attribute String name.

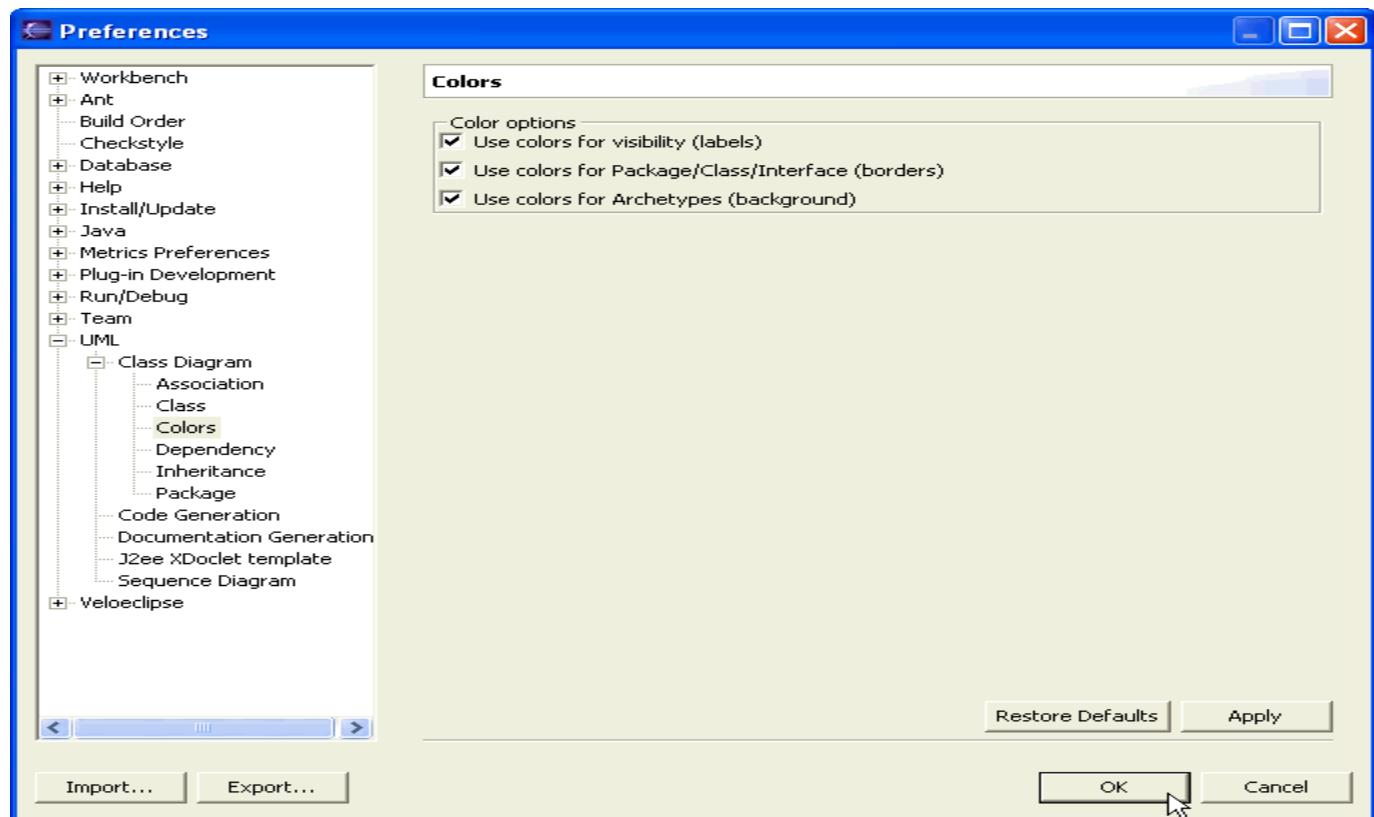


Please check that your Class diagram is the same as below.

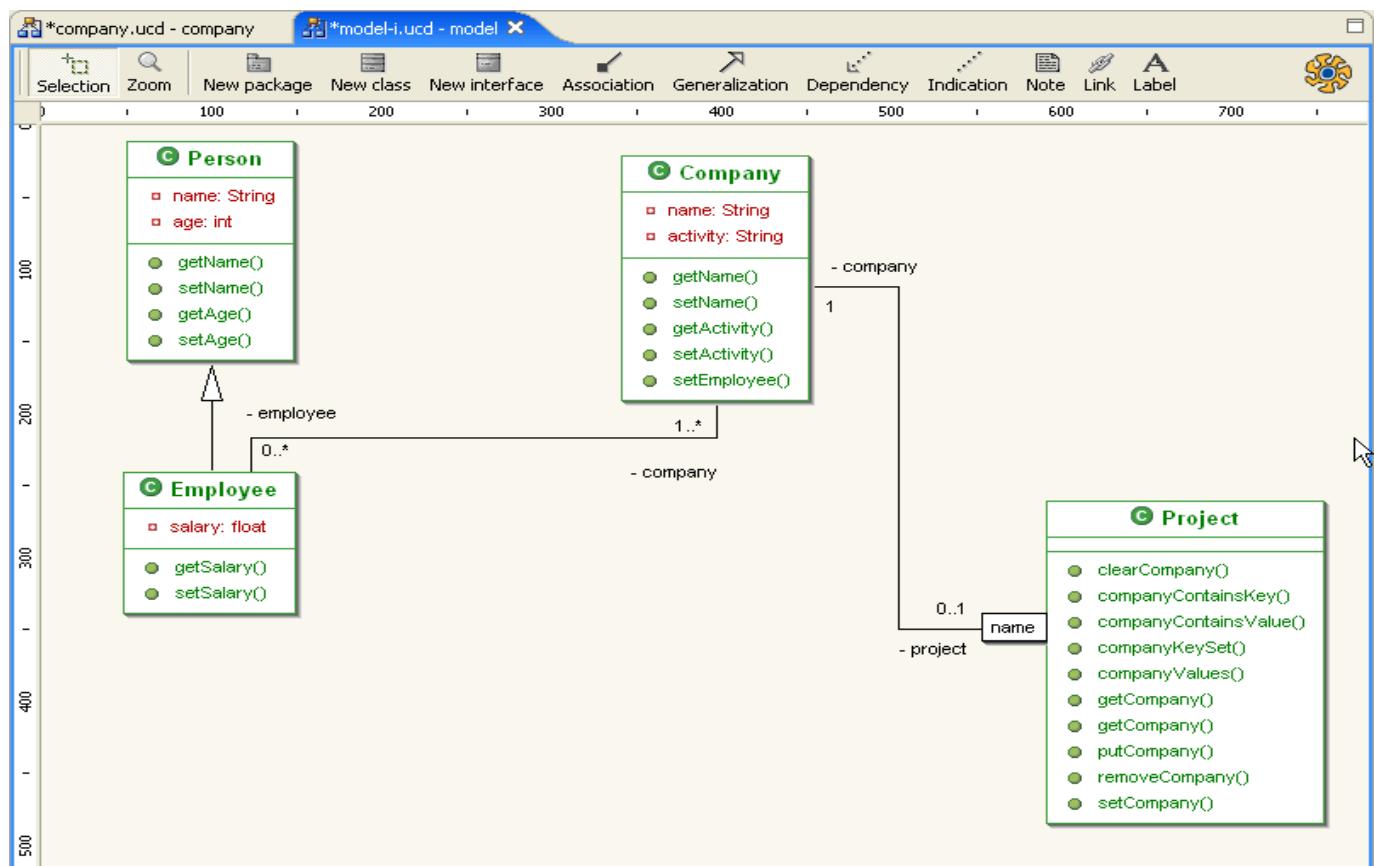


We can use colors to have a better understanding of our diagram.

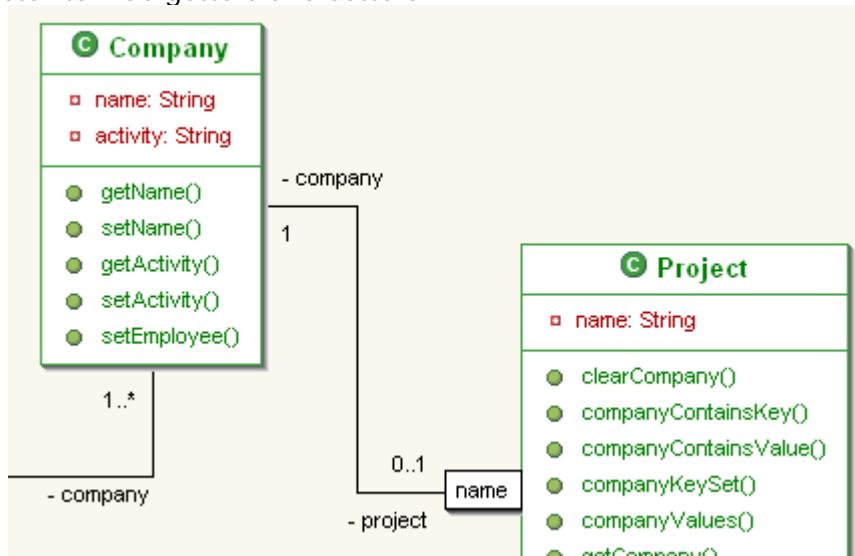
Select in the menu bar Windows > Preferences > UML > Class Diagram > Colors



Colors have been included in the class diagram below.



Add "name: String" new Attribute to Project class.
Use the view selector to hide getters and setters.



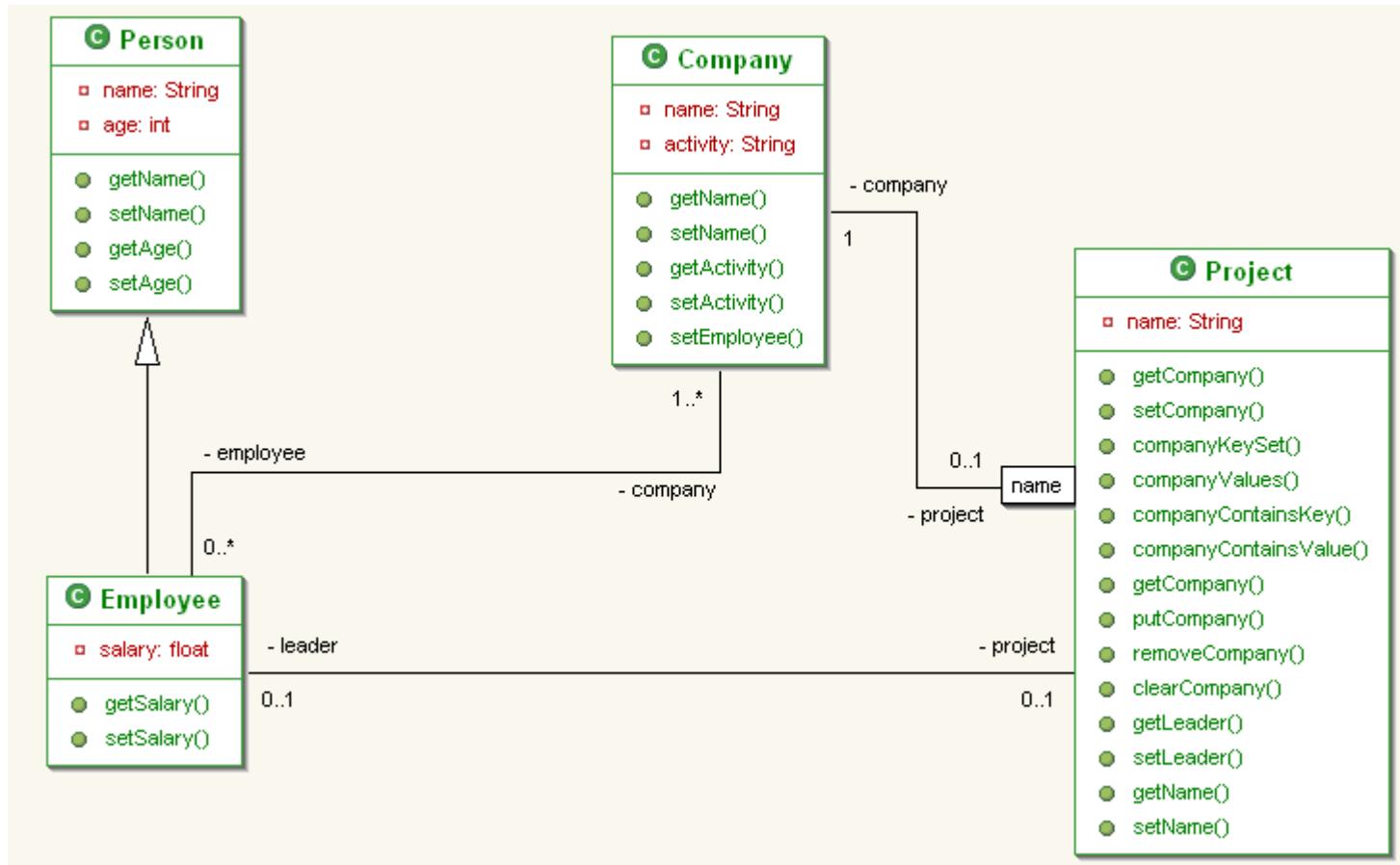
Add an Association between Employee and Project.
Select Association and drag from Employee to Project.

In the properties menu:

Connection: leave blank

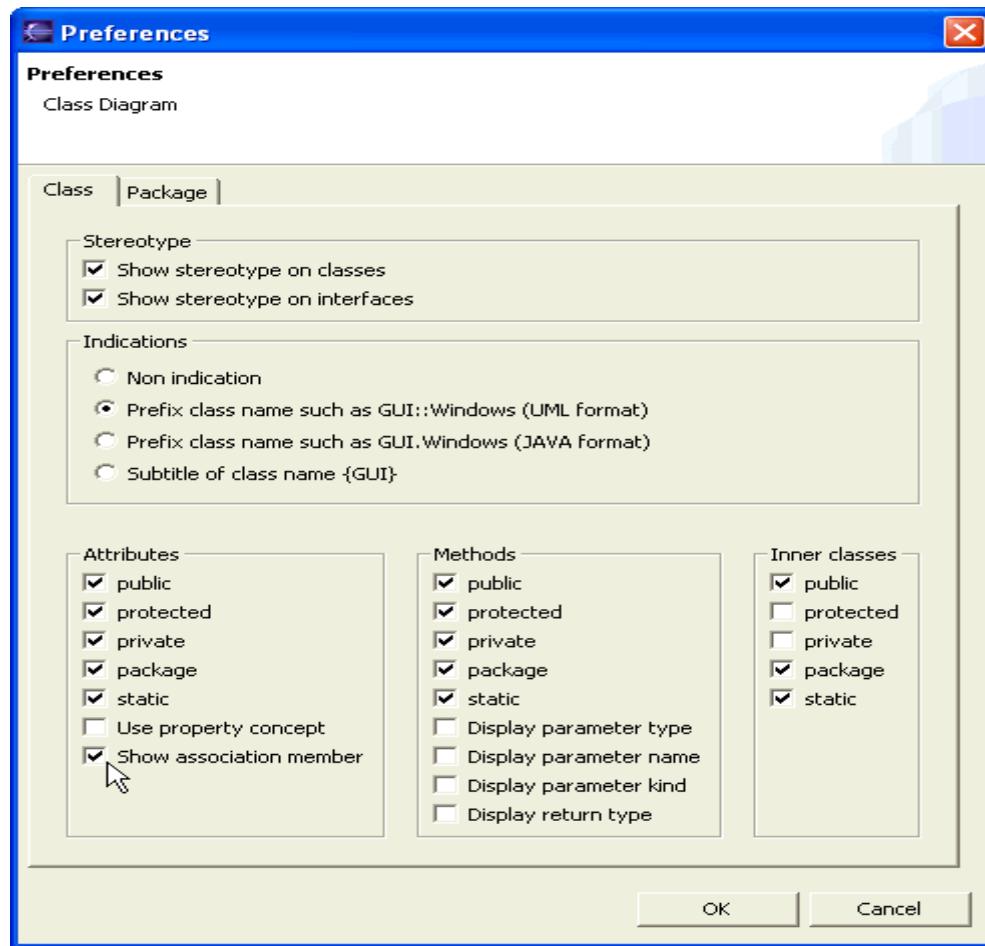
1st Association End: Name: leader, Dimension 0, Multiplicity 0..1

2nd Association End: Name Project, Dimension 0, Multiplicity 0..1

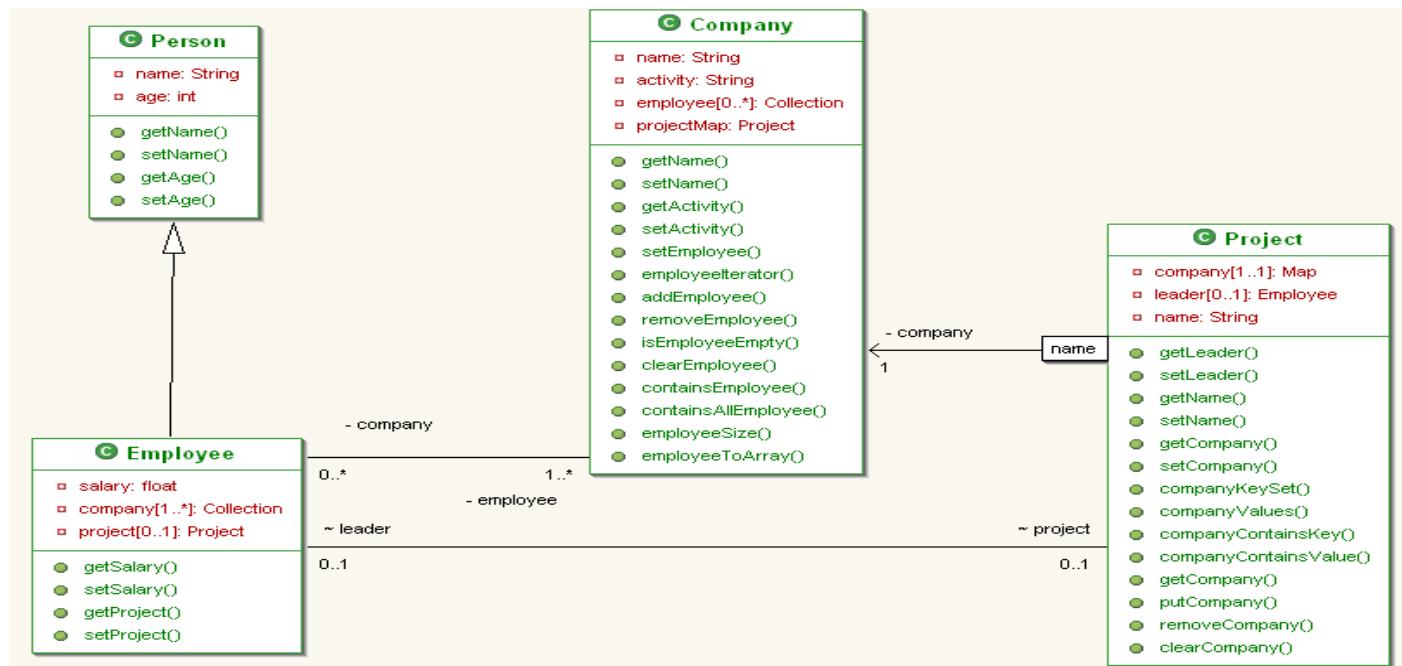


We also need to show association members.

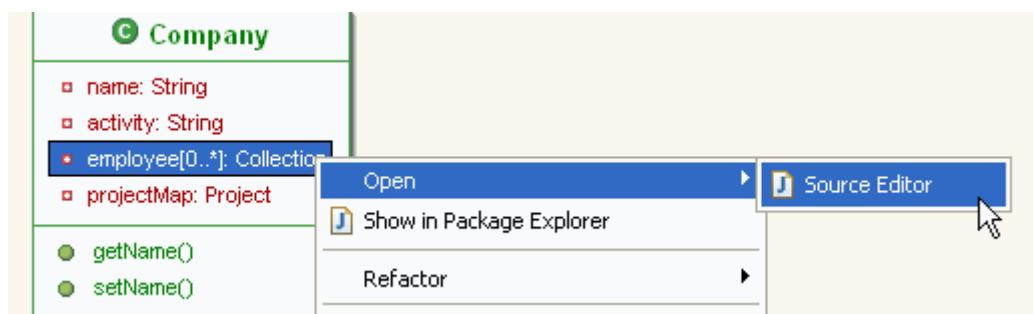
Right click inside the diagram editor, making sure that you don't select one of the elements - otherwise you are going to open the element popup menu and not the diagram popup menu. Select the Show association member checkbox and click on the OK button.



All association attributes are now displayed in the class diagram.



Select an element and right click to open **the element popup menu > View Selector**
Use view selector (if needed) to display employee, activity and project map attributes etc...
Show code. Click on **Employee attribute>Open>Source Editor**



Type new code:

```
private Collection employees = new ArrayList();
```

```
private Map projectMap = new Hashtable();
```

Copy Java code inside company class.

The marker shows unresolved type.

The screenshot shows the Eclipse Java editor with the file 'Company.java' open. The code is as follows:

```
    }
    if (employees != null)
    {
        Iterator iterator = projectMap.values().iterator();
        while (iterator.hasNext())
        {
            Project element = (Project) iterator.next();
            buffer.append(" Project: " + element.getName() + "\n");
        }
    }

    return buffer.toString();
}
```

A yellow warning marker is present on the line 'Iterator iterator = projectMap.values().iterator();'. The word 'Iterator' is underlined with a red squiggly line, indicating it is an unresolved type. The cursor is positioned at the end of the word 'Iterator'.

Correct the compilation error -> Resolve the type.

Place the mouse cursor at the end of "Iterator".

Control + space and select Iterator - java.util

The screenshot shows a Java code editor window in Eclipse. The code is as follows:

```

        }
    }
    if (employees != null)
    {
        Iterator iterator = projectMap.values().iterator();
        while (i < employees.size())
        {
            Proj
            buff
        }
    }
    return buff;
}

```

A tooltip for the word 'Iterator' is displayed, showing two entries:

- I Iterator - java.util
- C IteratorPool - org.apache.xpath.axes

The tooltip contains the following text:

An iterator over a collection. Iterator takes the place of Enumeration in the Java collections framework. Iterators differ from enumerations in two ways:

- Iterators allow the caller to remove elements from the underlying collection during the iteration with well-defined semantics.
- Method names have been improved.

See Also:

- Collection
- ListIterator
- Enumeration
- @author

Now you can go back to your class diagram and enjoy modeling!

The class diagram extension is *.ucd.

Sequence Diagram

In this section you will learn how to use the EclipseUML Sequence diagram.
The following areas are covered:

- [Concept](#)
- [ToolBar](#)
- [Message Creation](#)
- [Drag'n'Drop](#)
- [Message information](#)
- [Reverse Engineering](#)
- [Reverse Engineering extension](#)
- [Reverse Engineering filtering](#)
- [Sequence Diagram Example](#)
- [Tips and tricks](#)

Concept

A sequence diagram is a model describing how groups interact over time. It is an interaction diagram that details how operations are carried out: what messages are sent and when.

You use the sequence diagram model to describe the aspects of your system that change over time. These are events that mark changes, sequences of events and so forth. The time sequence interaction is shown in the sequence diagram and consists of two dimensions. The vertical dimension (time) and the

horizontal dimension (different objects).

UML sequence diagrams can be used to:

- explore your design
- feel which classes are going to be complex
- validate the logic and completeness of a usage scenario
- detect bottlenecks within an object-oriented design

The elements of the sequence diagrams are available in the tool bar:

 New Object can be created by selecting the icon in the toolbar. Allows you to add a new instance within the diagram.

 New Actor can be created by selecting the icon in the toolbar

 Message can be created by selecting the icon in the toolbar. Each message represents a method call.

 Self Message can be created by selecting the icon in the toolbar. A reflexive message is a message sent by an instance to itself.

Toolbar items

This section describes the tools provided in order to build a Sequence Diagram. For further information please refer to the UML specification available on the [OMG](#) web site.

Actor

Add a new actor on the diagram.

Instance

Add a new instance on the diagram.

Message

This tool creates an interaction between an actor/instance and an instance. Each message represents a method call.

Self Message

This tool allows you to create reflexive messages. A reflexive message is a message sent by an instance to itself

Message creation

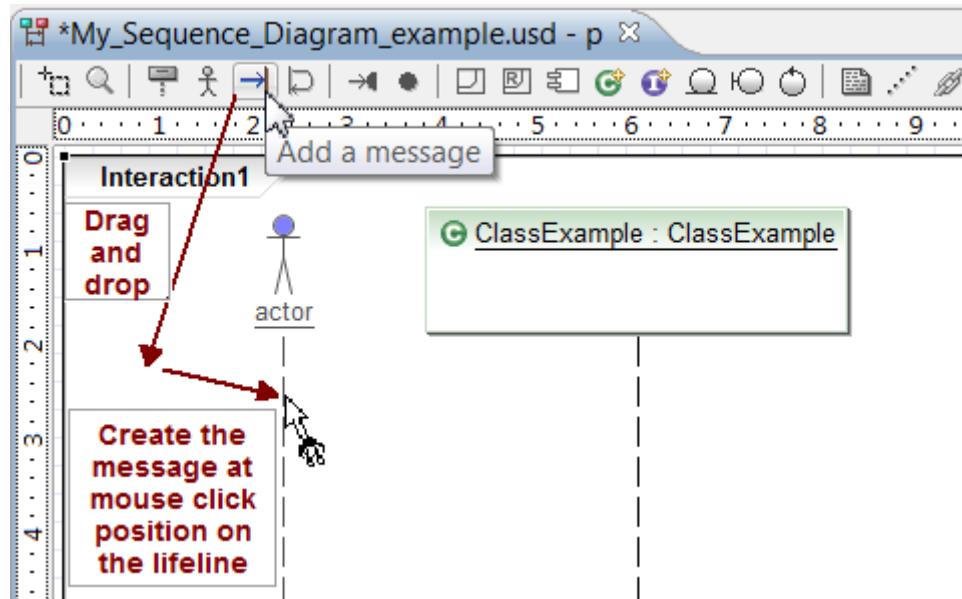
In a sequence diagram you can create messages and add advanced properties

1. [Create a message](#)
2. [Create a return message](#)

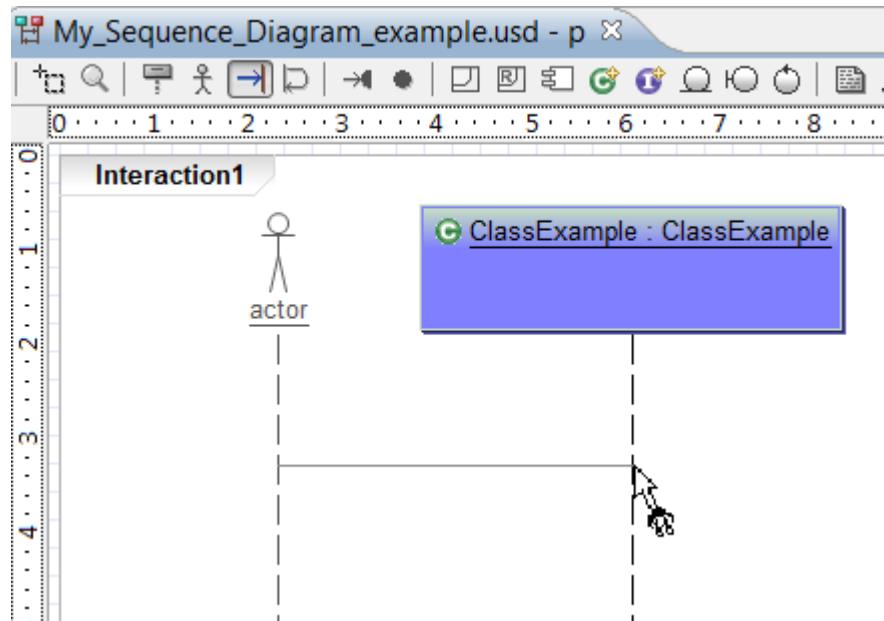
3. [Create a Creation Message](#)
4. [Create a Destruction Message](#)
5. [Add Return Arrow](#)

1. Create a message

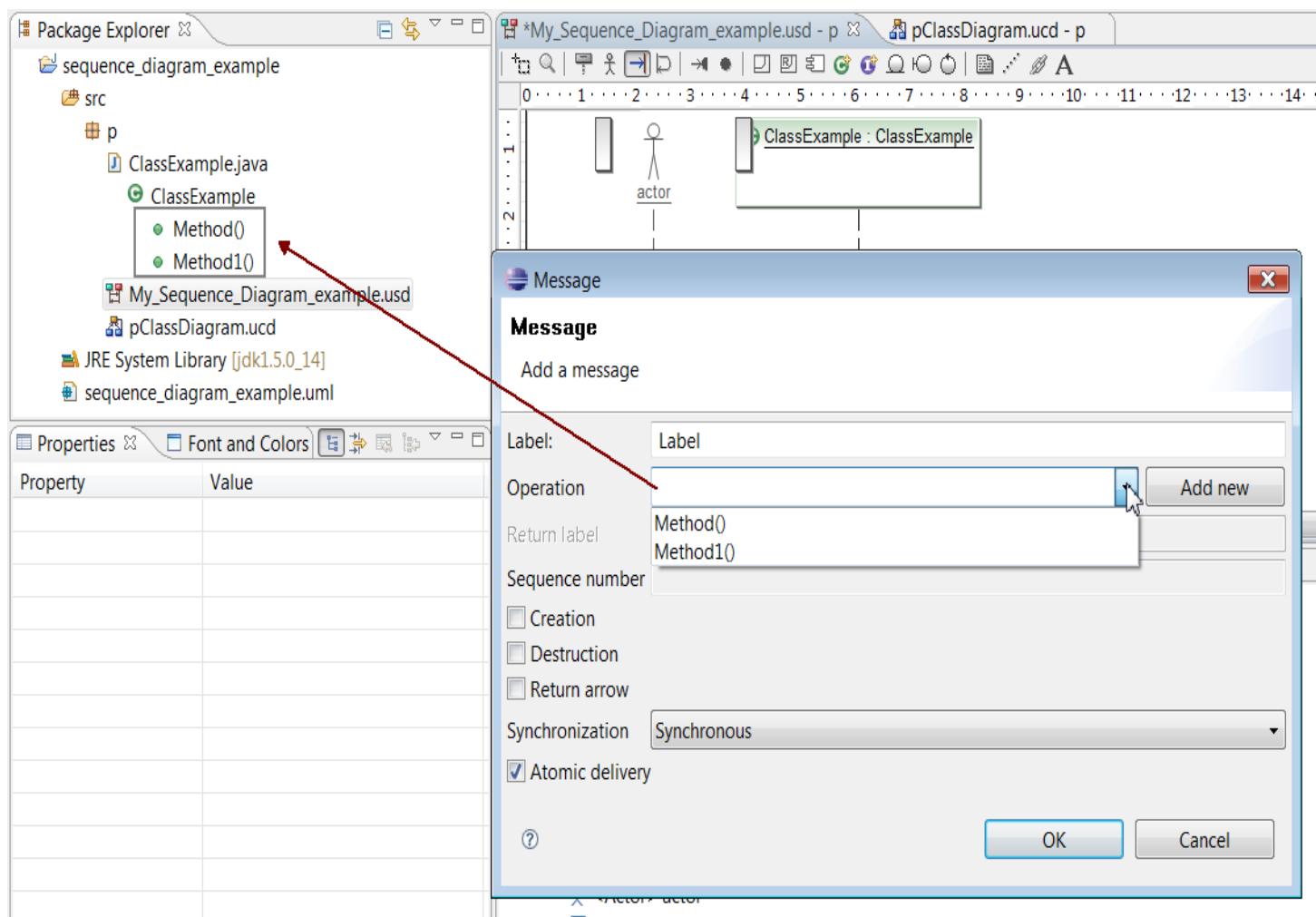
Select the **Add a message icon** in the toolbar and drag drop the message to the Lifeline. Note that your message will be created at the mouse click drop position on the lifeline.



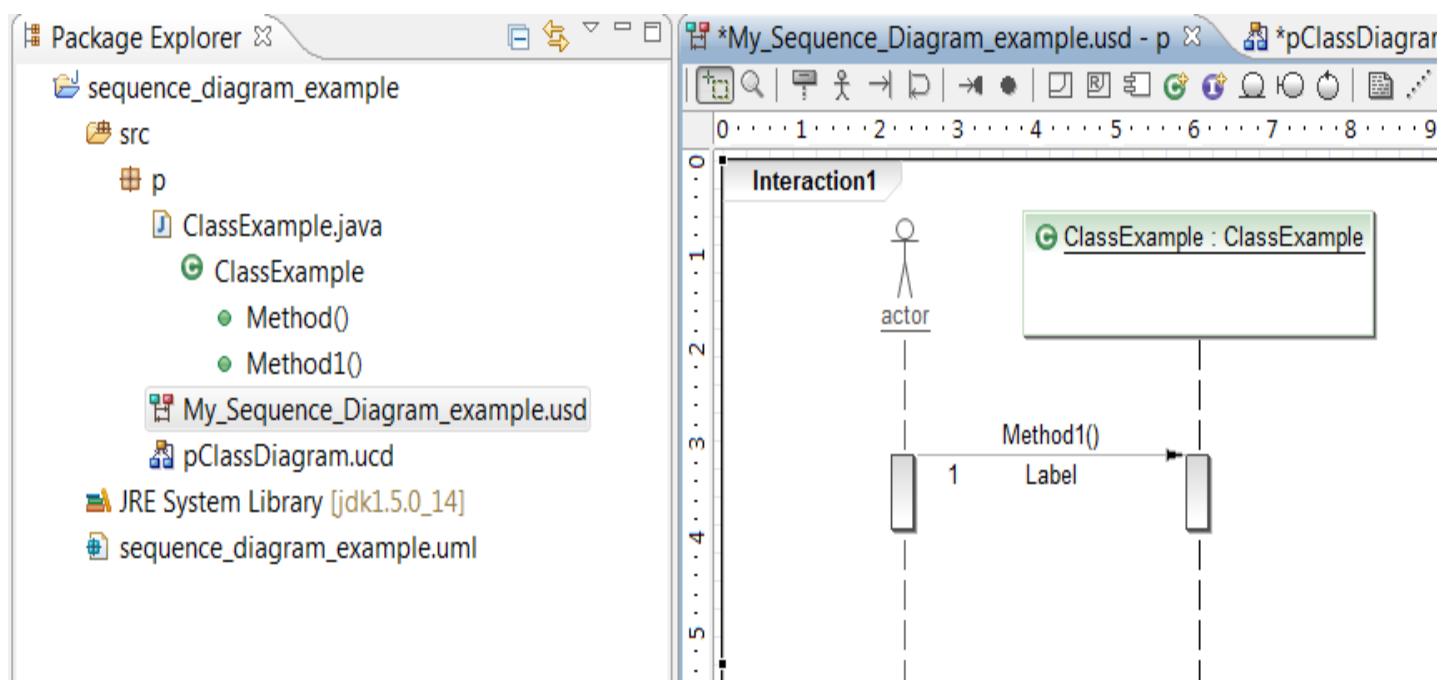
Once the first click has been made then drops the message to another property which will become blue if selected.



You can select an existing method of a Java class ([java class drop feature](#)) and add it to the diagram if you click on the top/down arrow of the Operation field.



The message has been created in the Sequence Diagram



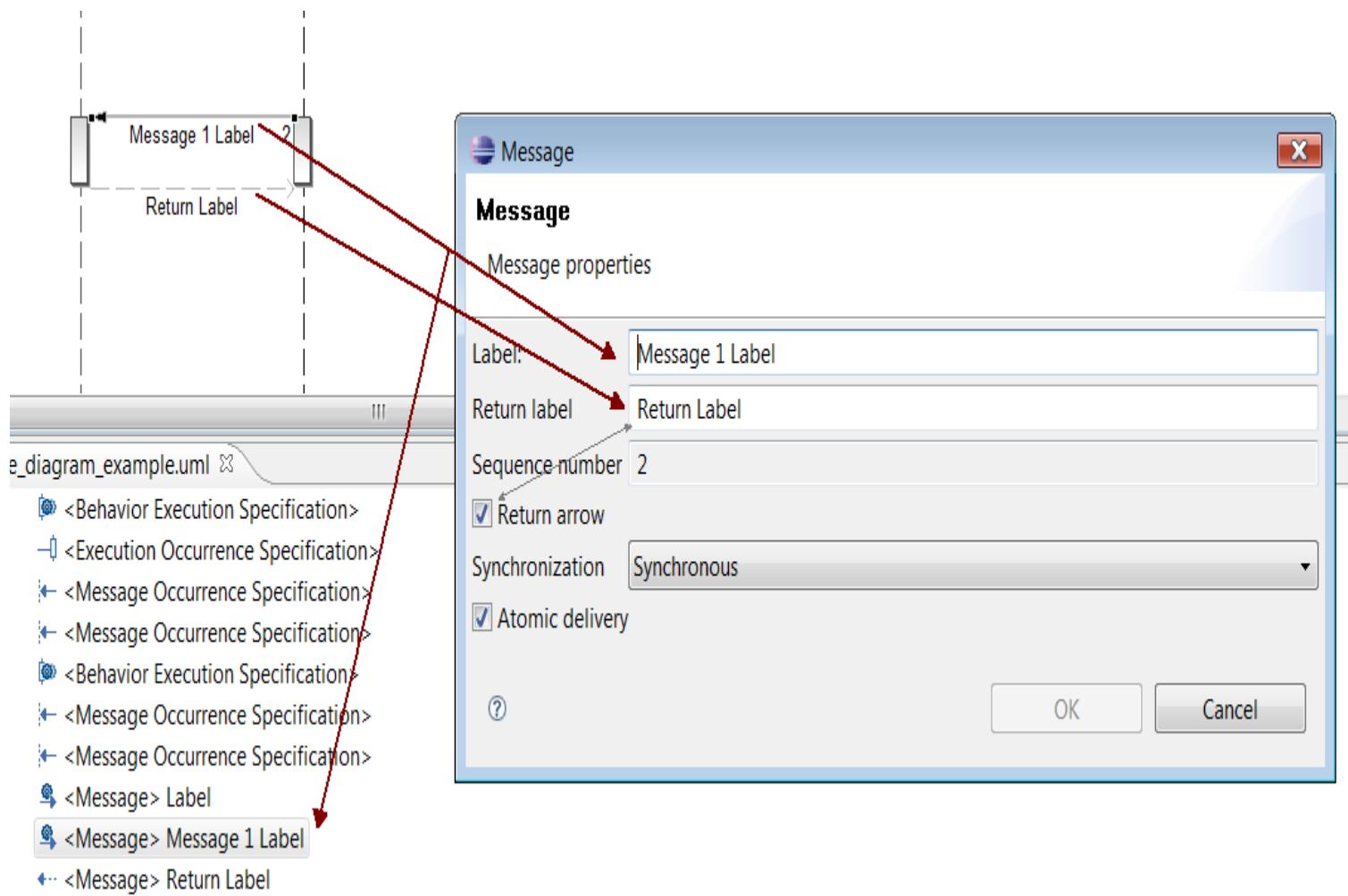
2. Create a return message

You can also make a message going backward by selecting a property on the right and drop it on the left of the diagram.

The Message property menu is different if going from right to left and not from left to right like in the previous example above.

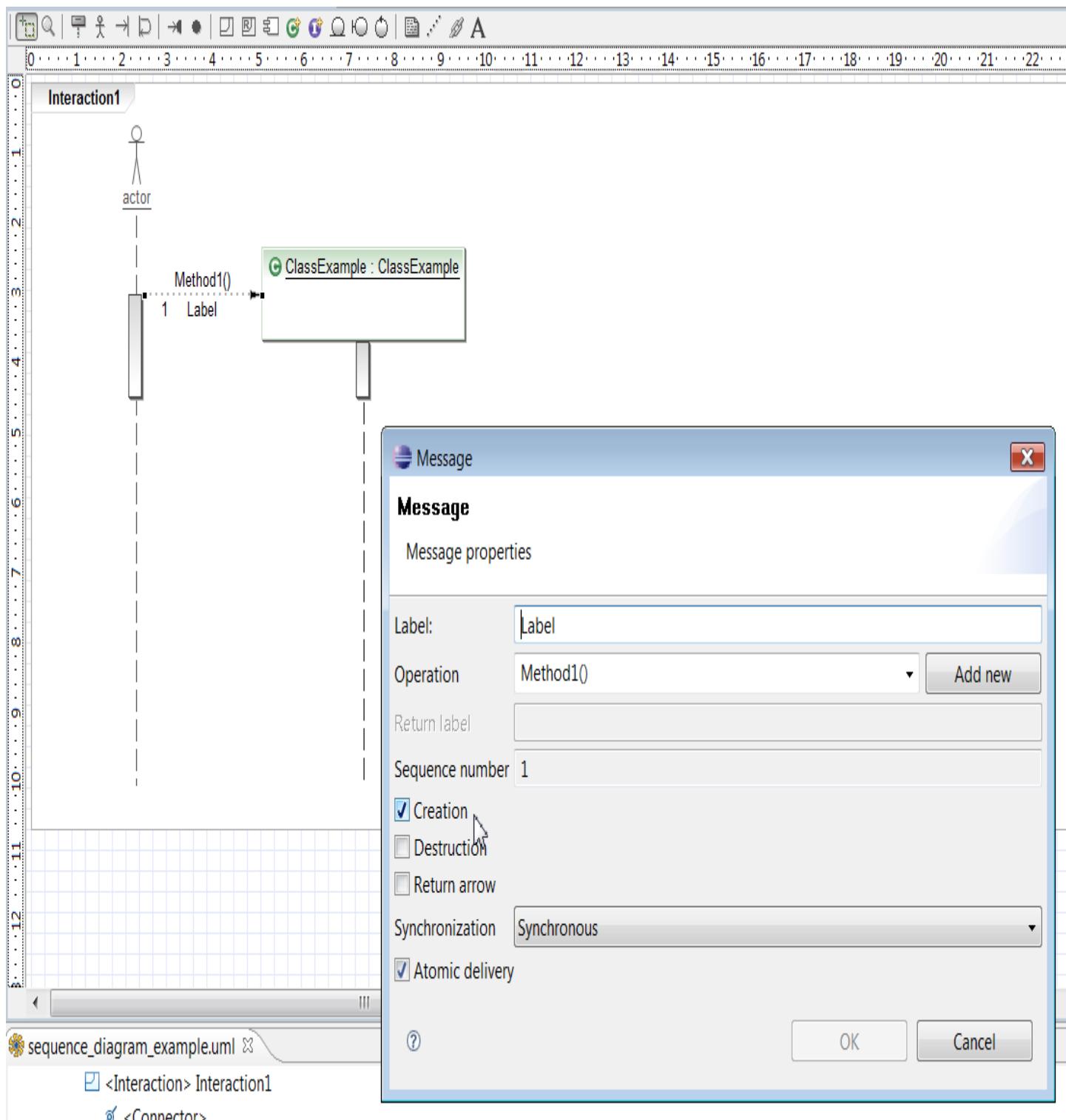
It allows you to add a return label by selecting the Return arrow option.

Below are the created message and its return label which is displayed in the Sequence Diagram and in the UML Superstructure of the model



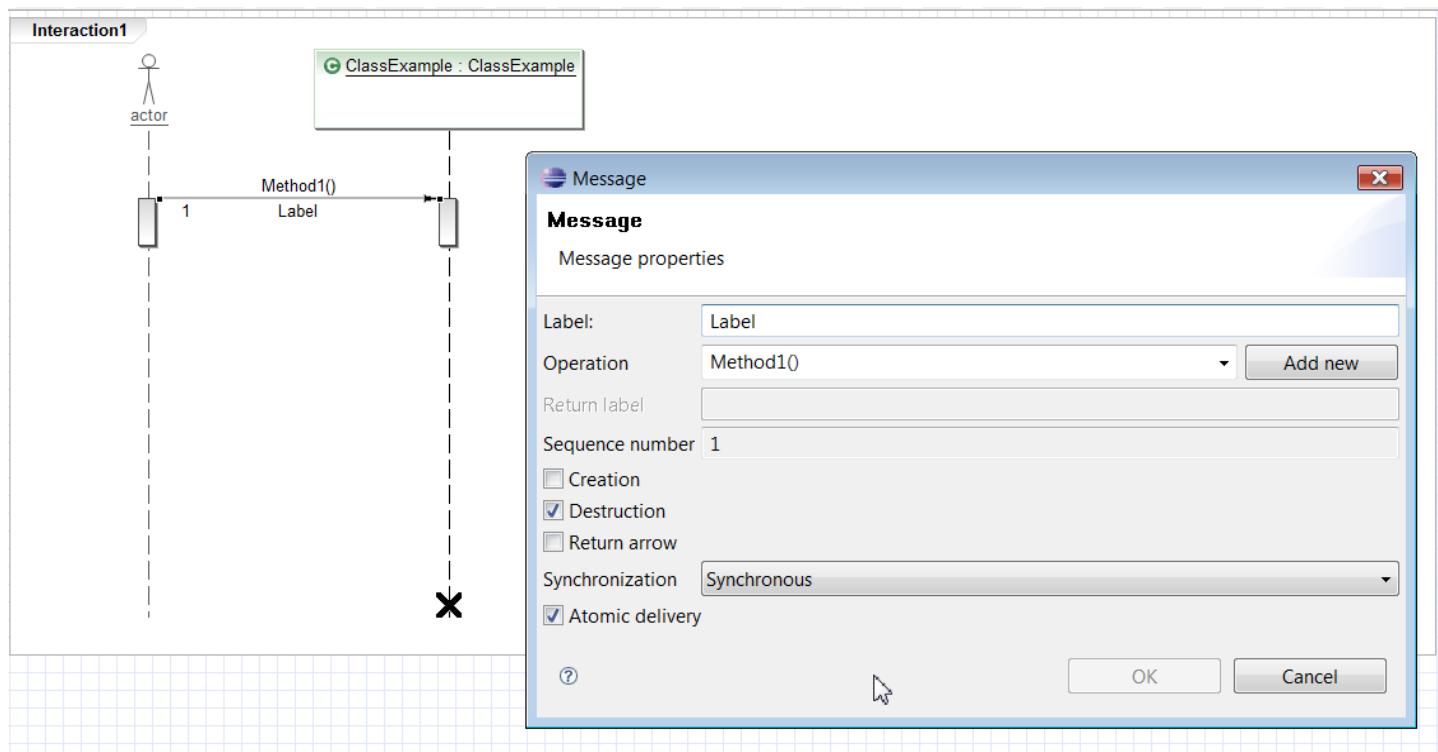
3. Create a Creation Message

You can add a **creation message** to your diagram by selecting the Creation option in the message property.



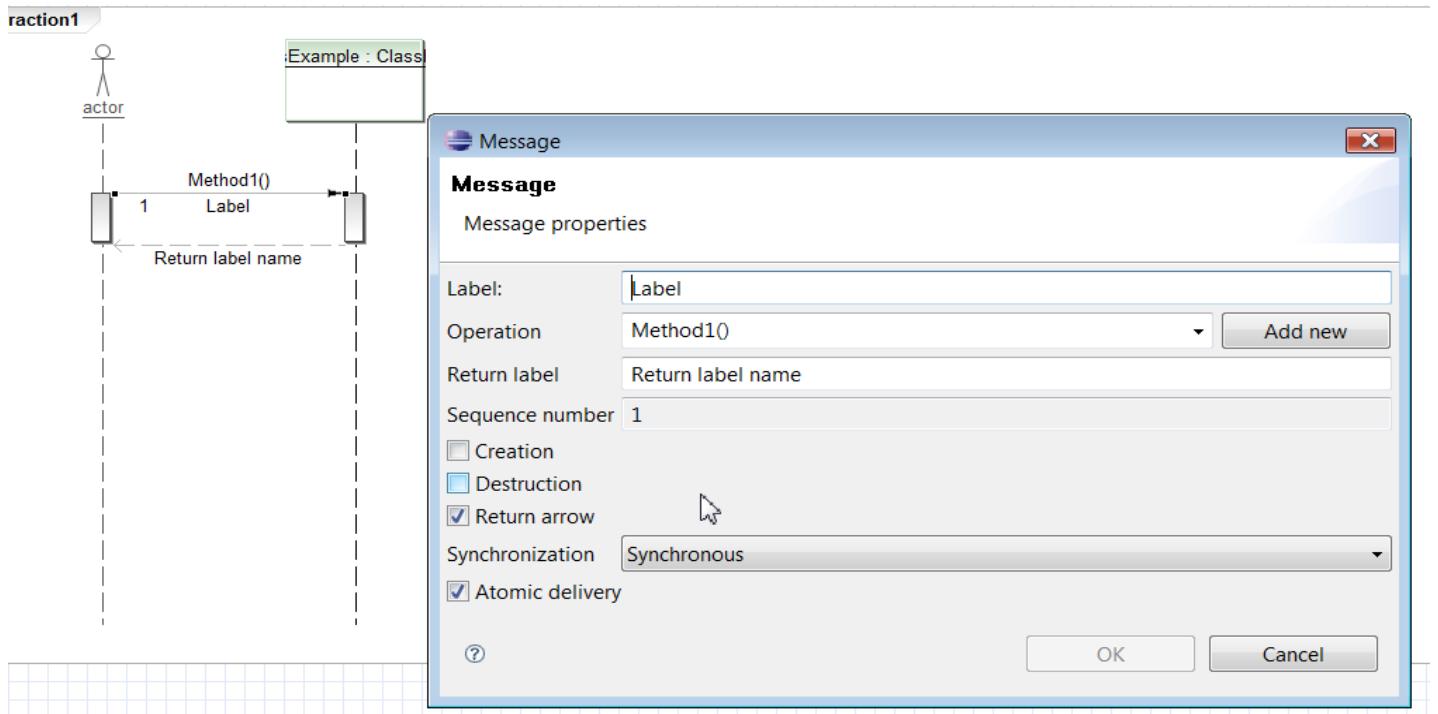
4. Create a Destruction Message

You can add a **destruction message** to your diagram by selecting the destruction option in the message property.



5. Add Return Arrow

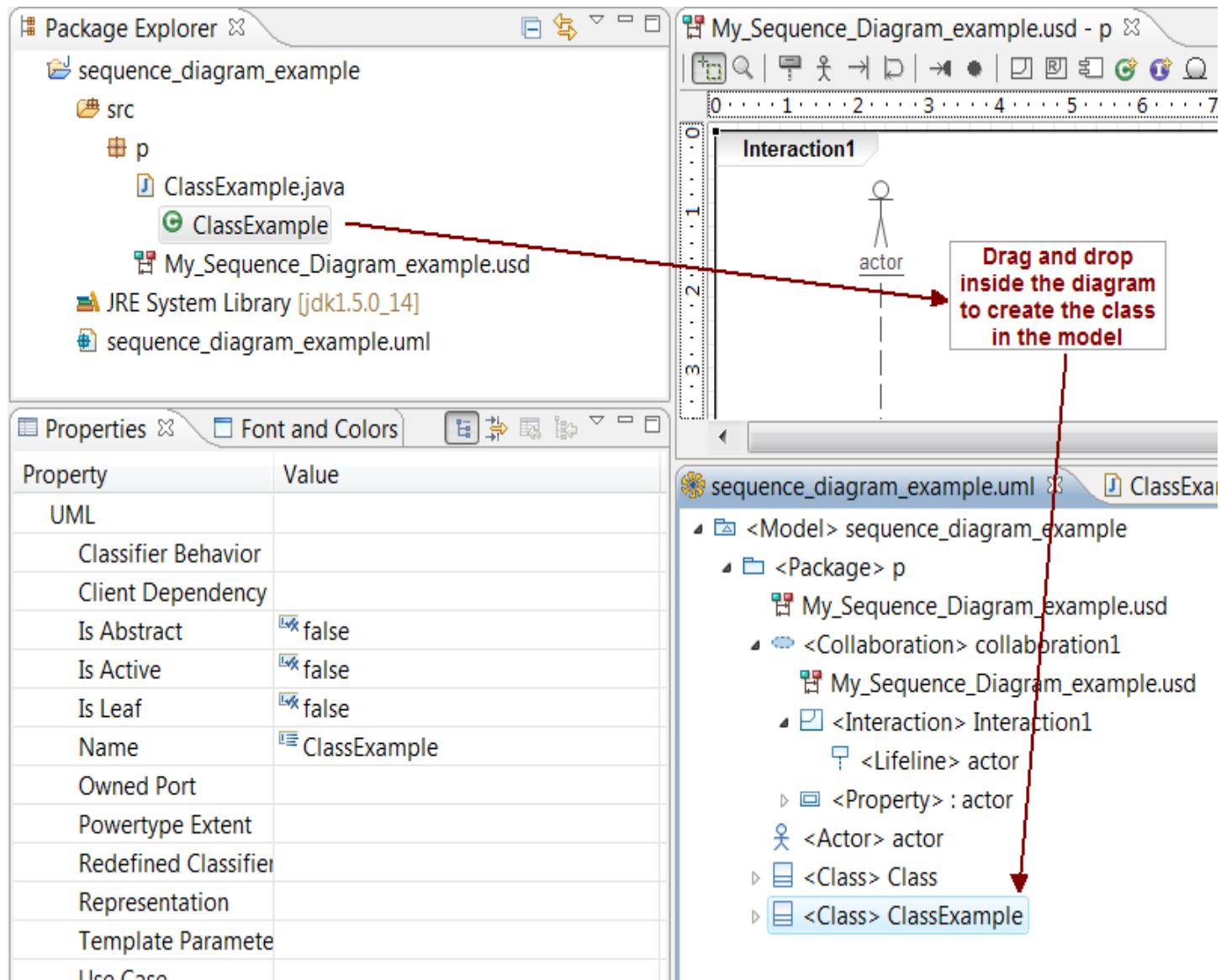
You can add a **Return label message** to your diagram by selecting the Return arrow option in the message property.



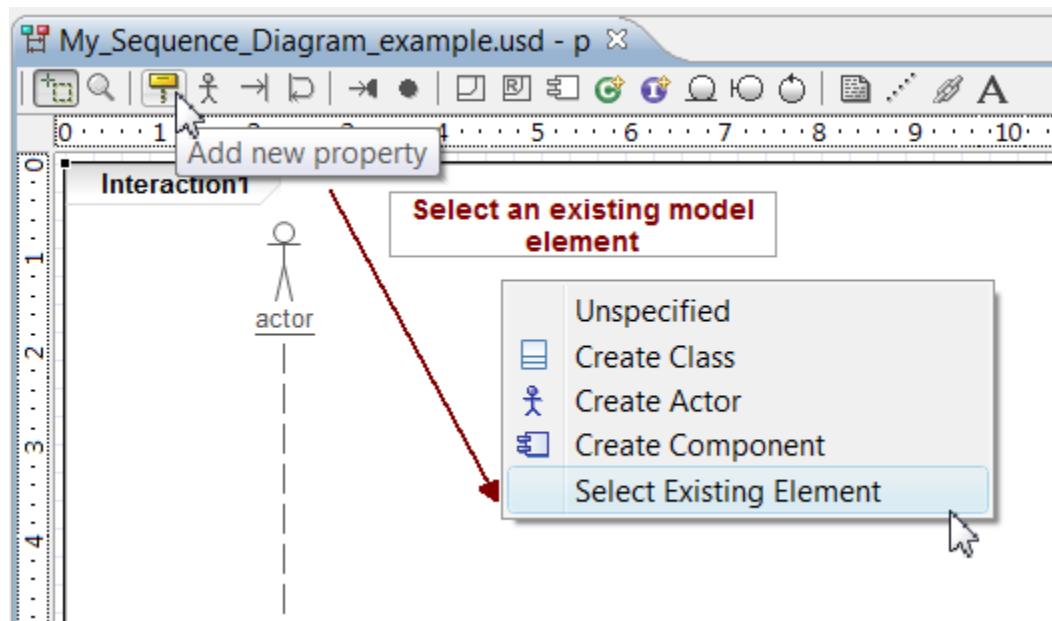
Drag and Drop

In a sequence diagram, you can drag and drop a class or an interface from the Package Explorer to the Sequence Diagram Editor and get a new instance.

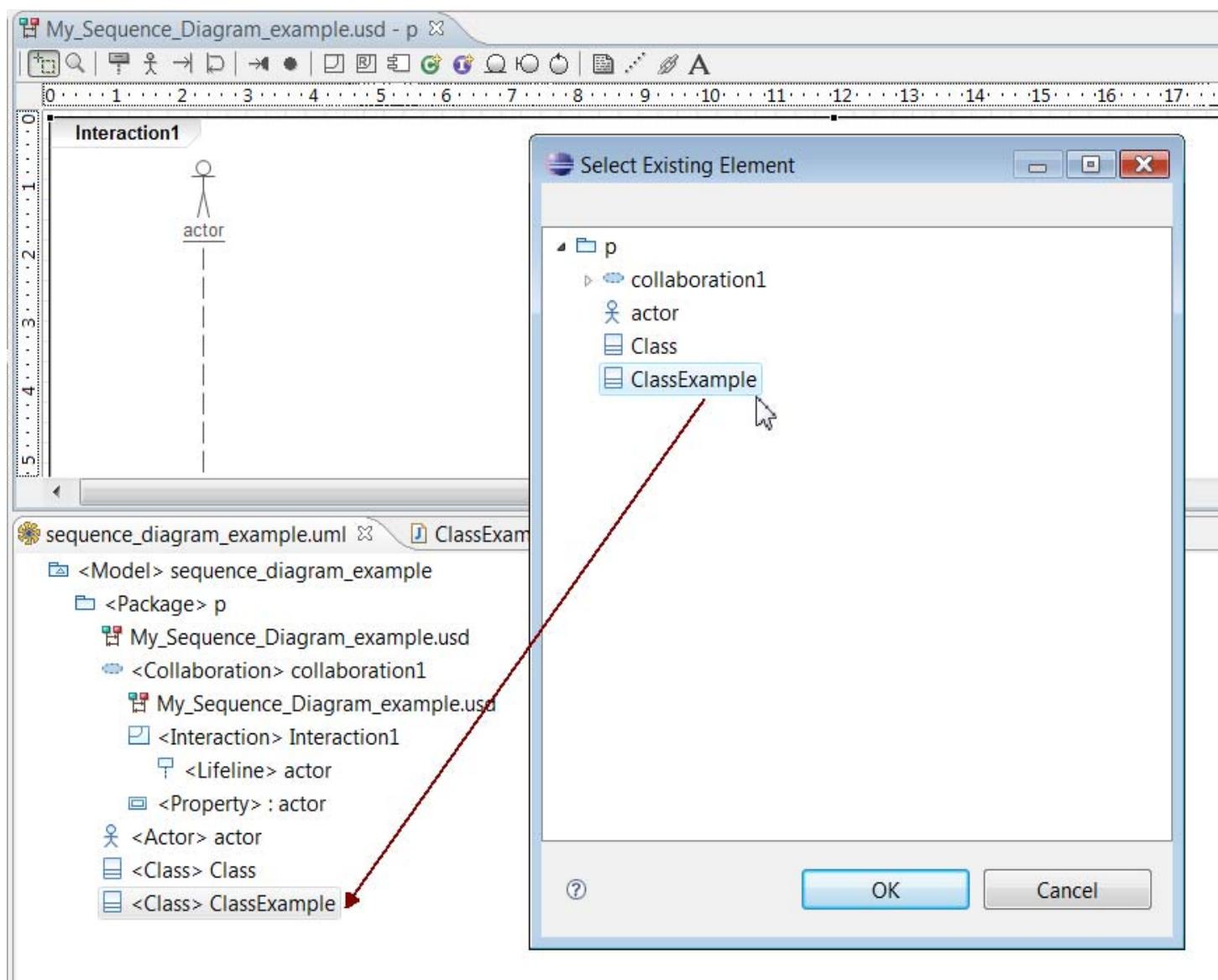
You need first to **create the classifier in the model** if it doesn't already exist by **a drag and drop from the Package Explorer to the Sequence Diagram**



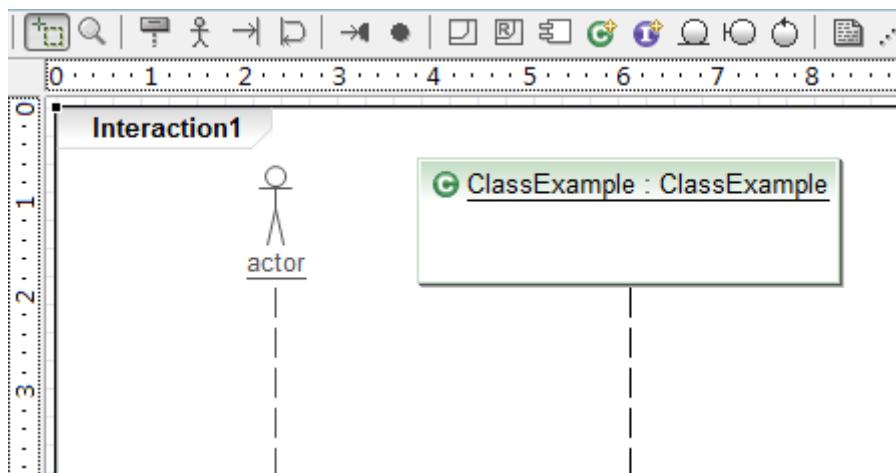
Click on the **Add new property icon** and Select the **Select Existing Element menu**



Select the newly created class in the model explorer and click on the Ok button



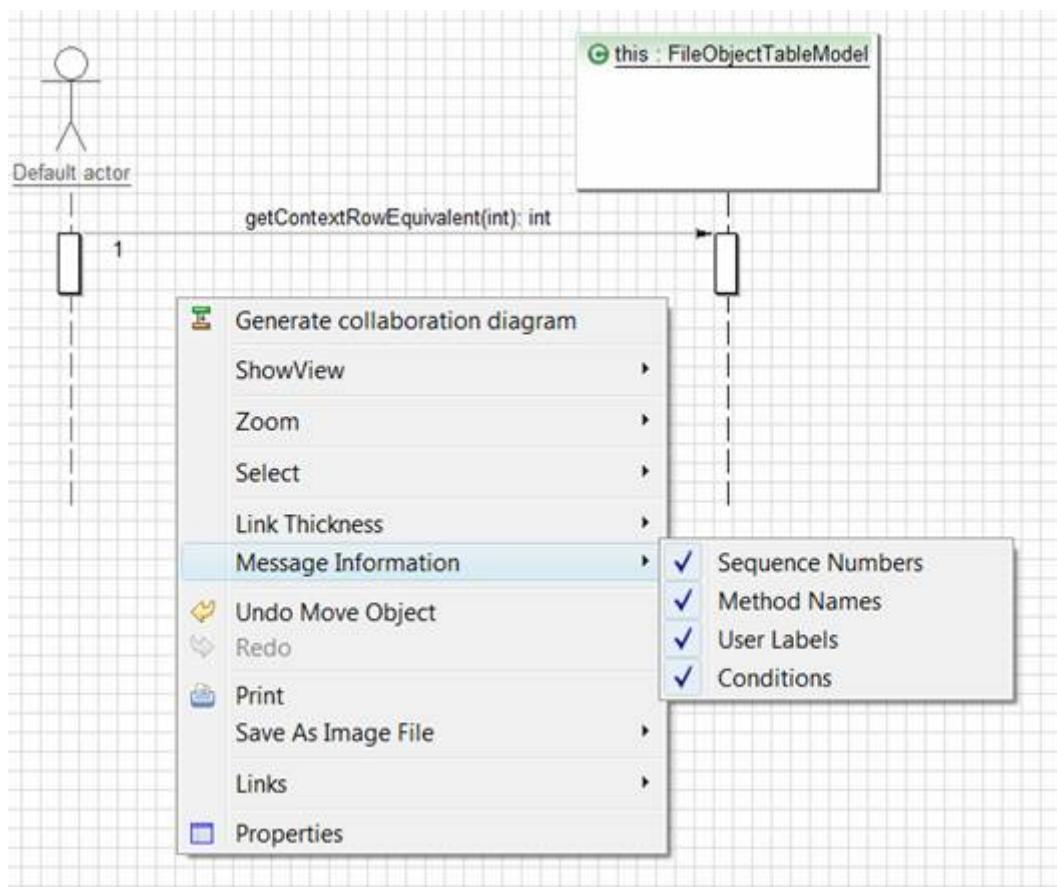
The Java Class will be added to the sequence diagram



Message information

The following example is the reverse of a simple "com.example" package focusing on a virtual reservation website.

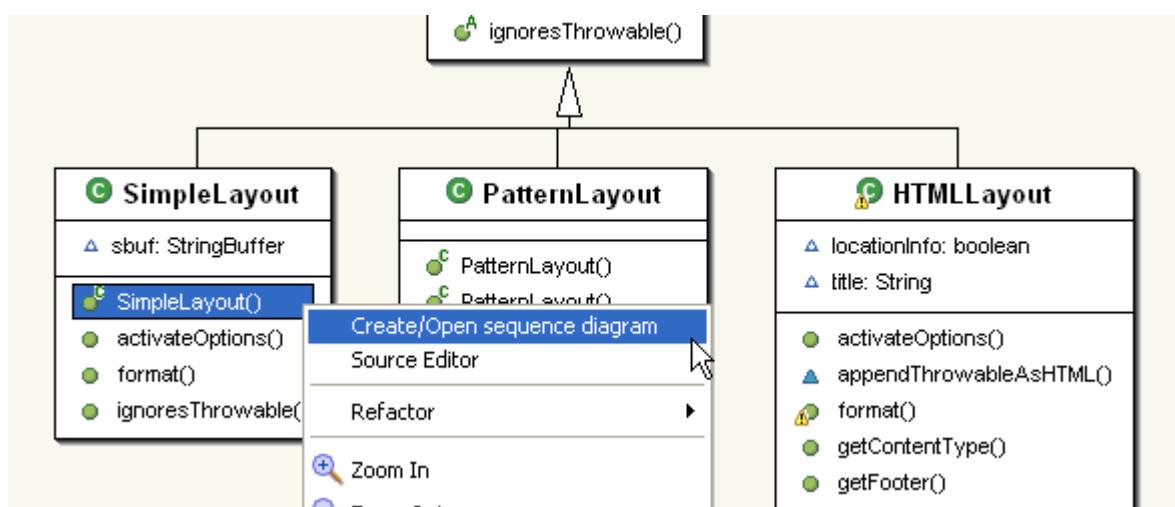
You can customize the message information by using the sequence diagram contextual menu > **Message Information** > ...



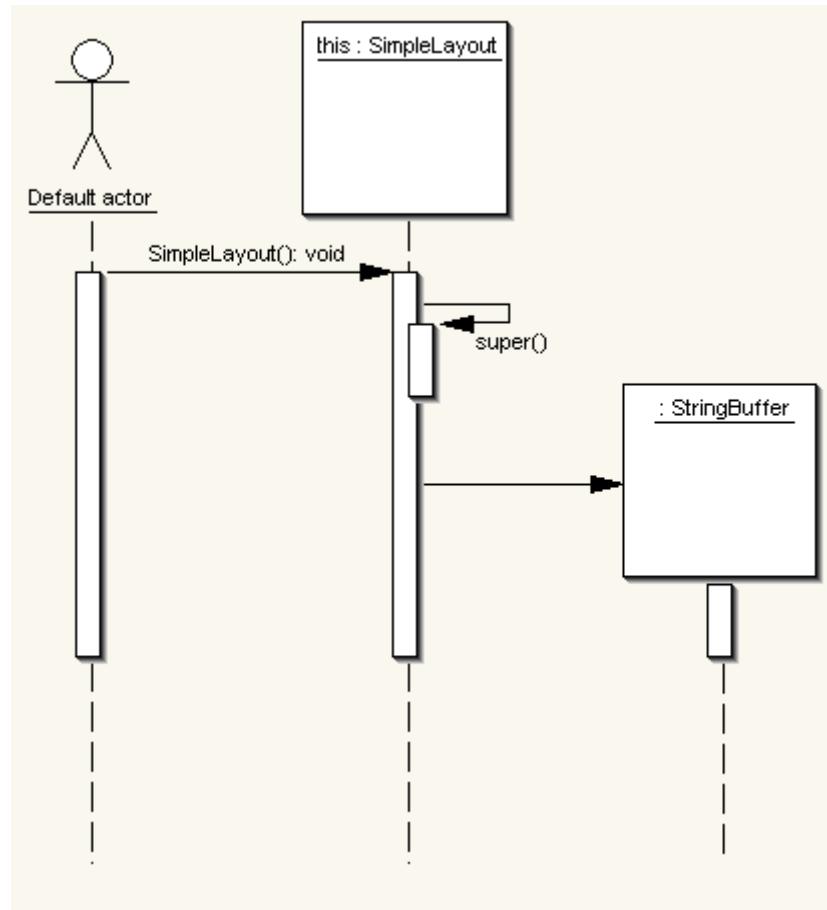
Sequence diagram reverse engineering

Double click on a method inside the class diagram editor or select a method in the package explorer **open the class diagram popup menu > Create/open sequence diagram**.

The following example is the reverse of the SimpleLayout() method in the SimpleLayout class from the org.apache.log4j.or package.



This result of the reverse of the SimpleLayout() method.

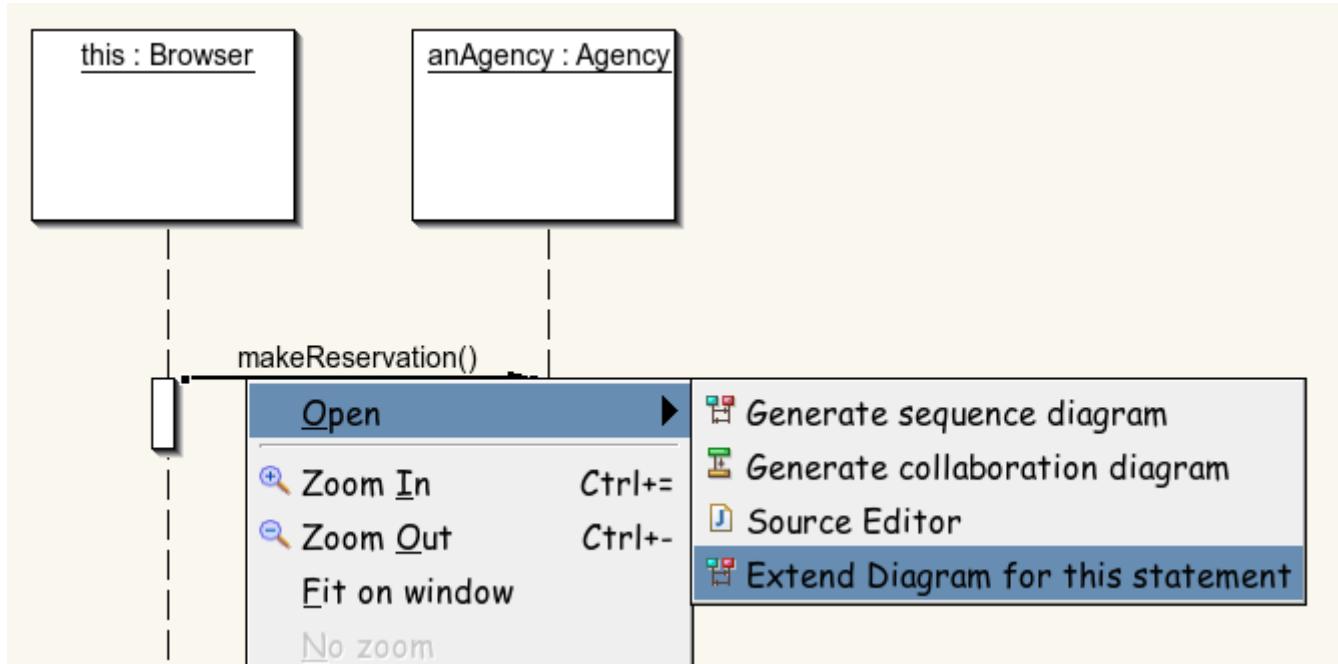


Sequence diagram extension

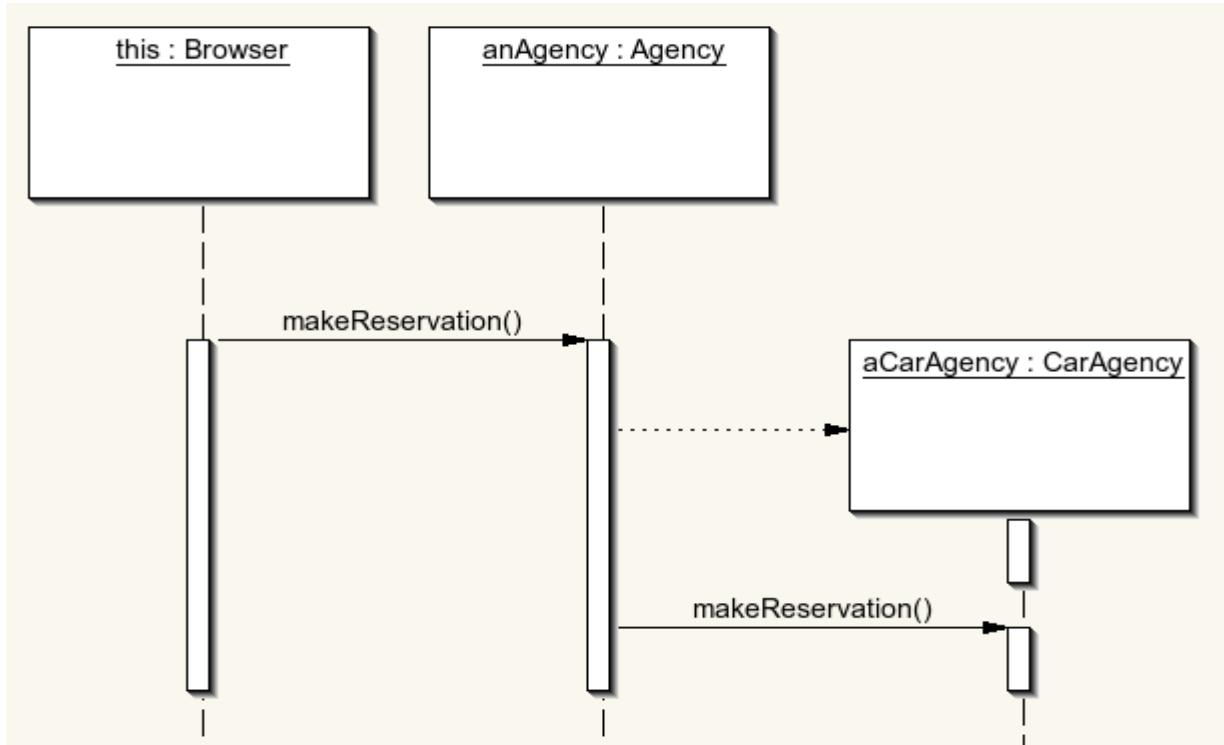
The following example is the reverse of a simple "com.example" package focusing on a virtual reservation website.

If a reverse message is bold, it means that you can reverse it again.

Select a message, right click to open the pop-up menu and select Open > Extend Diagram for this statement in order to **extend the diagram**.

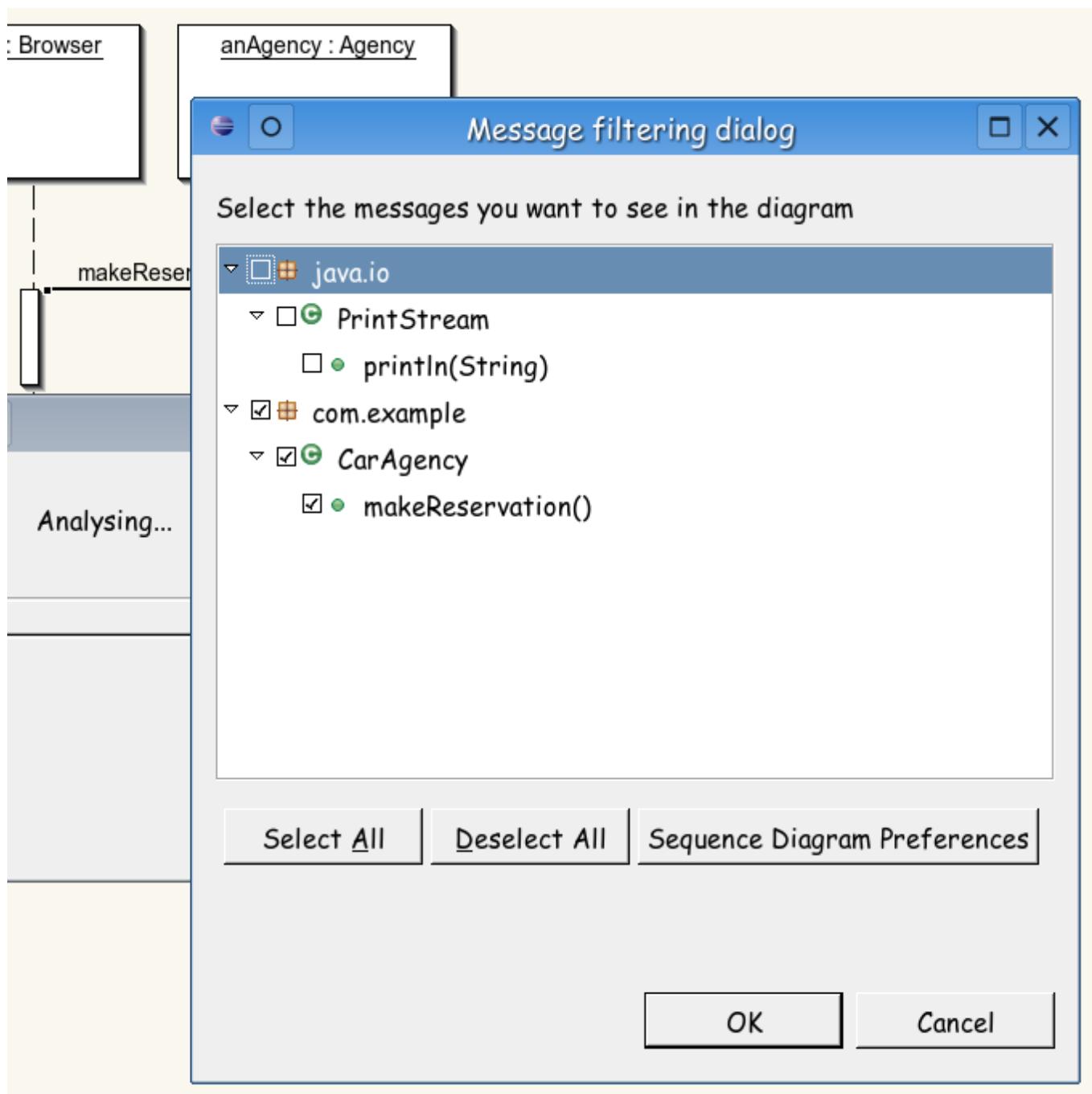


The result of the **reverse of the makeReservation() method** is:

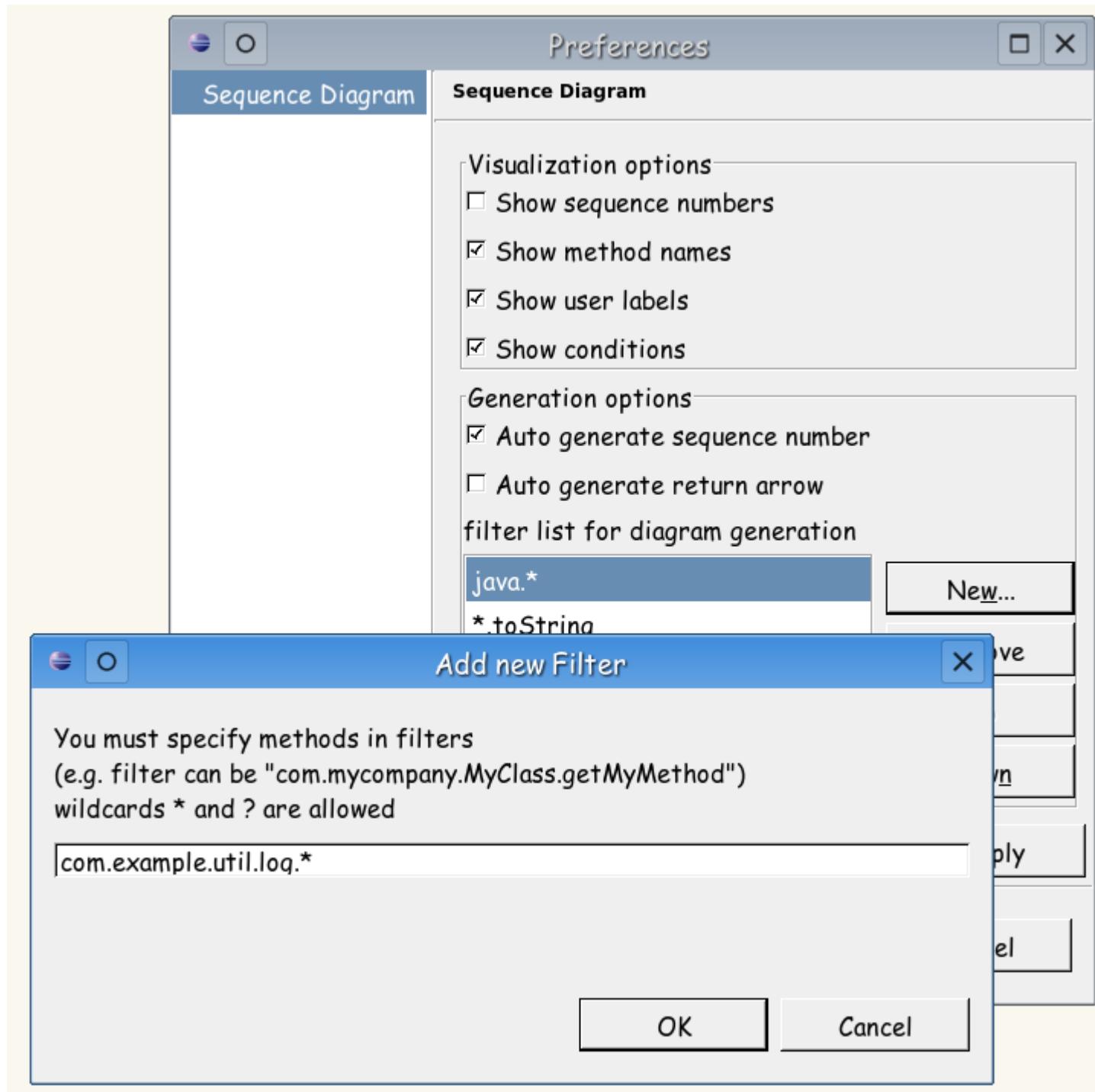


Sequence diagram filtering

When you **reverse a method**, a dialog box will allow you to **filter messages**. Selecting **Cancel** will import all the messages without filtering.



You can add **your own filters** by selecting the **Sequence diagram preferences button**.



Sequence Diagram example

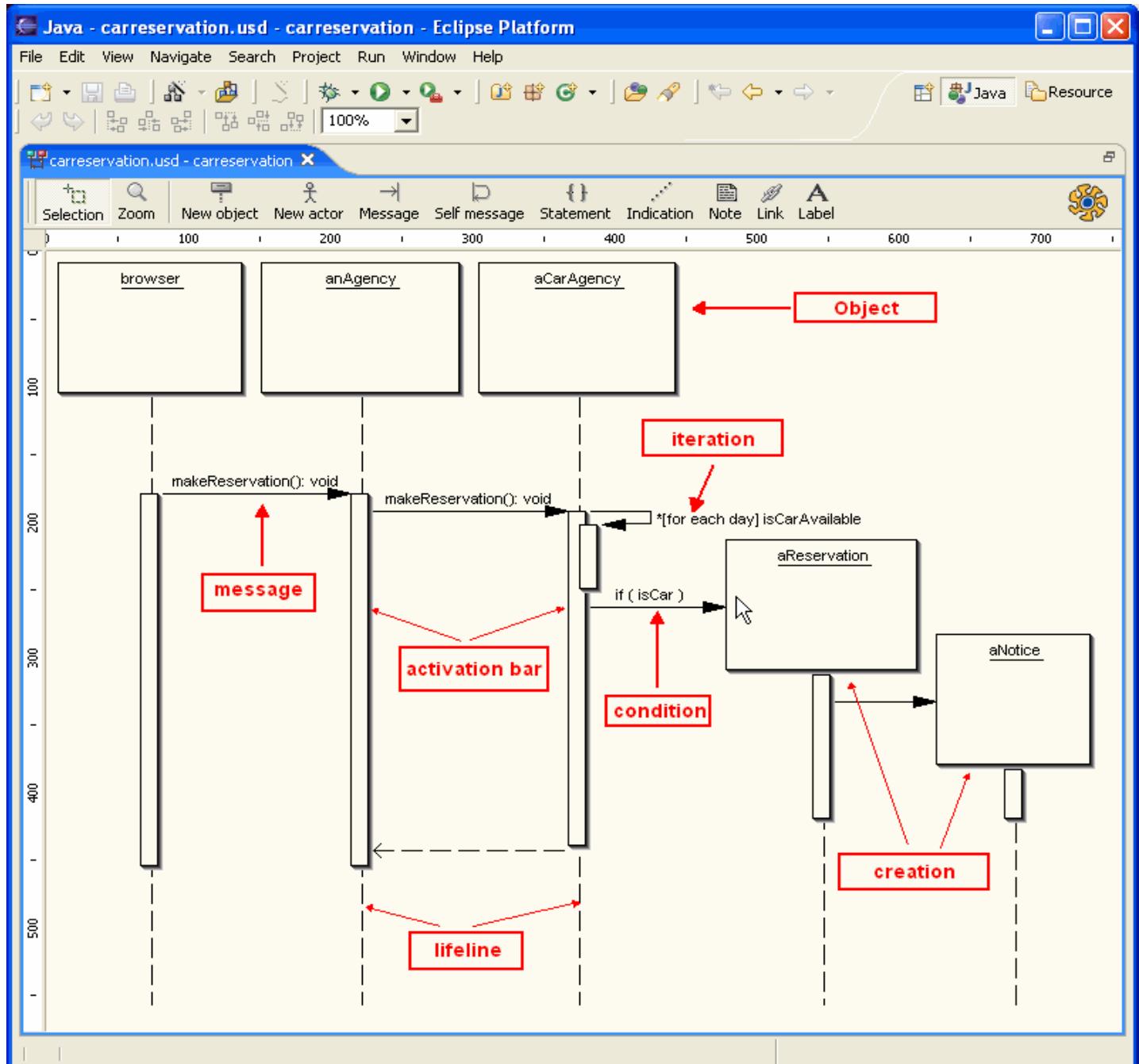
Below is a sequence diagram for making a car reservation using a browser.

The object initiating the sequence of message is a reservation internet browser.

The internet browser sends a makeReservation() message to a travel agency, which sends a makeReservation() message to a car rental reservation. If the car is available, then it makes a Reservation and a Confirmation.

Each vertical line represents the time that an object exists. Each arrow is a message call. An arrow goes from the sender of the activation bar of the message on the receiver's lifeline. The activation bar represents the duration of execution of the message.

The car rental reservation issues a self call to determine if a car is available. If so, the car rental reservation makes a Reservation and a Confirmation.



The file deployment diagram extension is *.usd. Notes can be included.

Tips and tricks

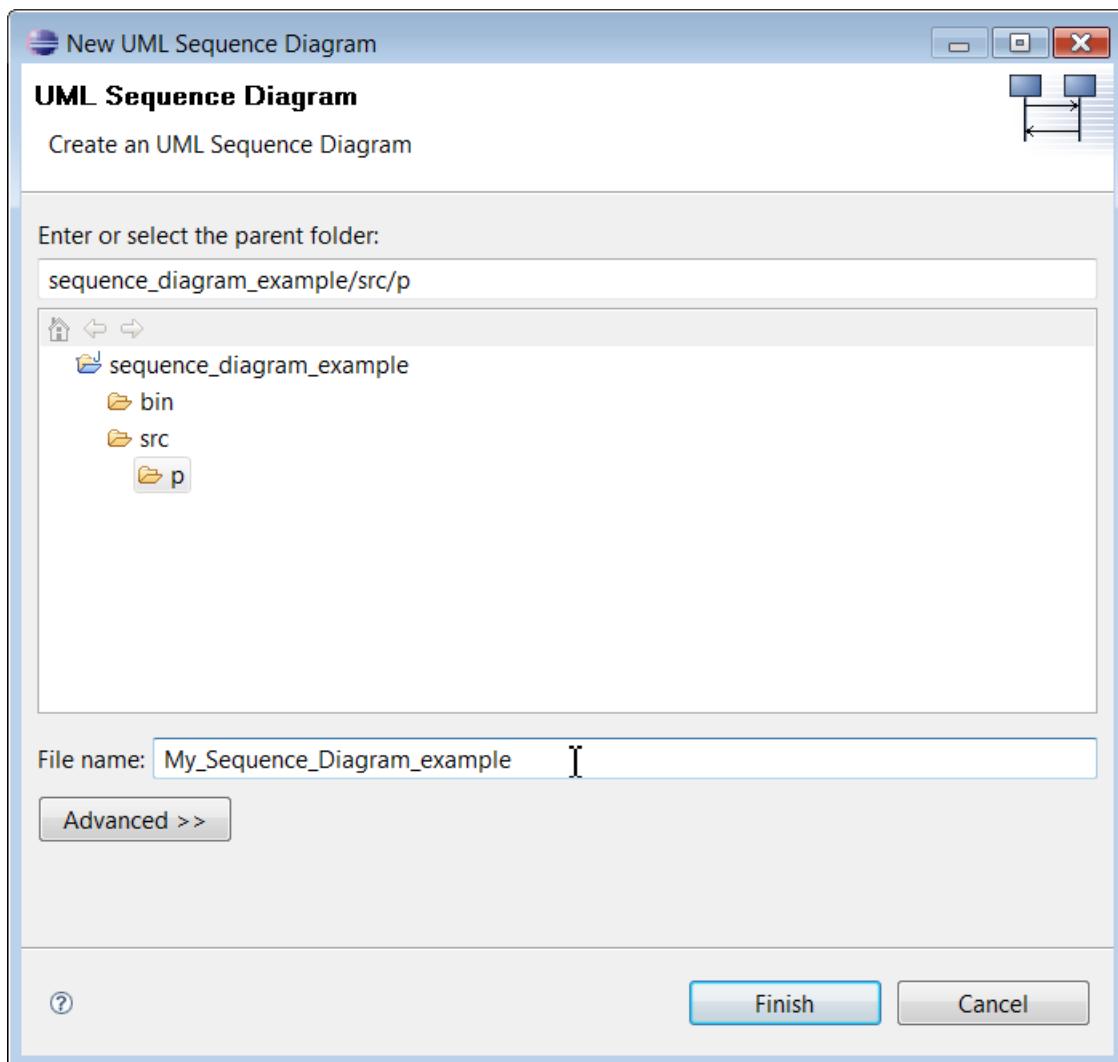
In a sequence diagram you have many hidden tips and tricks which have been implemented and are not in the documentation.

You can for example use the following tips:

1. [Give a name to your diagram](#)
2. [Change Property name](#)
3. [Have the same Lifeline height](#)
4. [Change the property size](#)
5. [Move a message](#)
6. [Move a frame](#)
7. [Resize an activation bar](#)
8. [Move a creation message up and down](#)
9. [Message straight line](#)
10. [Frame compartments size update](#)
11. [Change the name of the frame](#)
12. [Ok button of the frame is not activated](#)

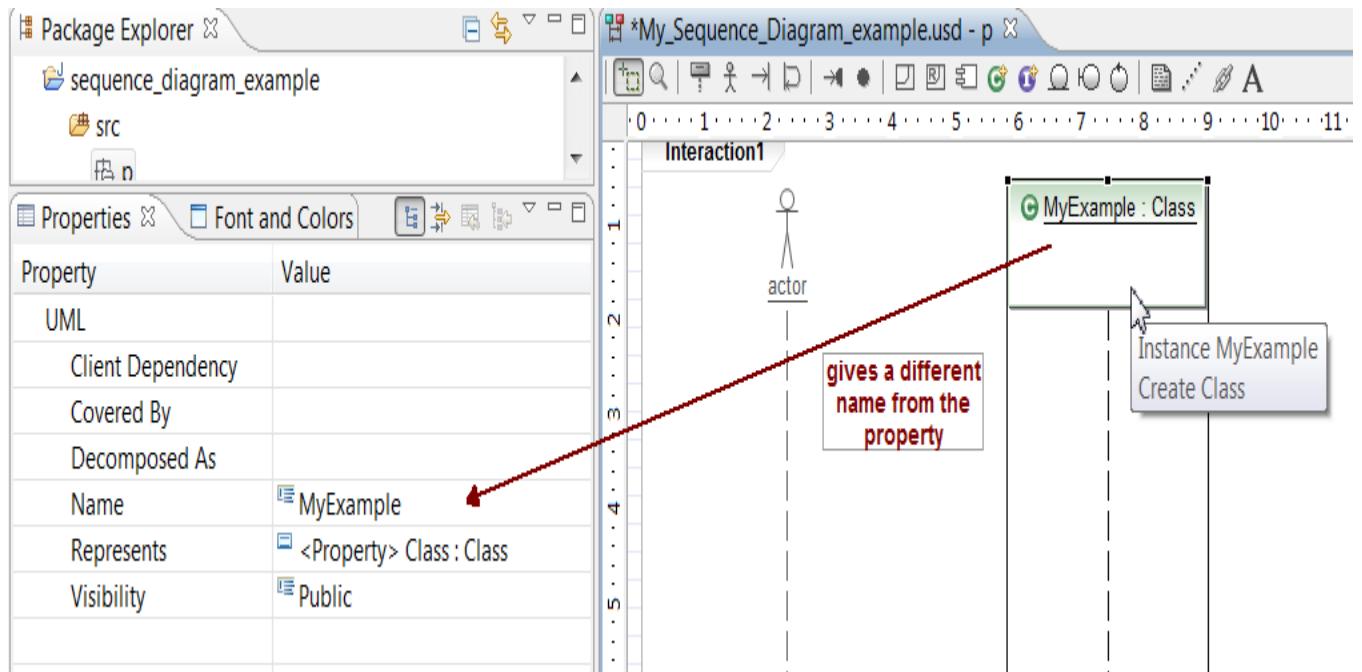
1. Give a name to your diagram

You can change the name of the sequence diagram by typing a new Sequence Diagram name in the File name field.



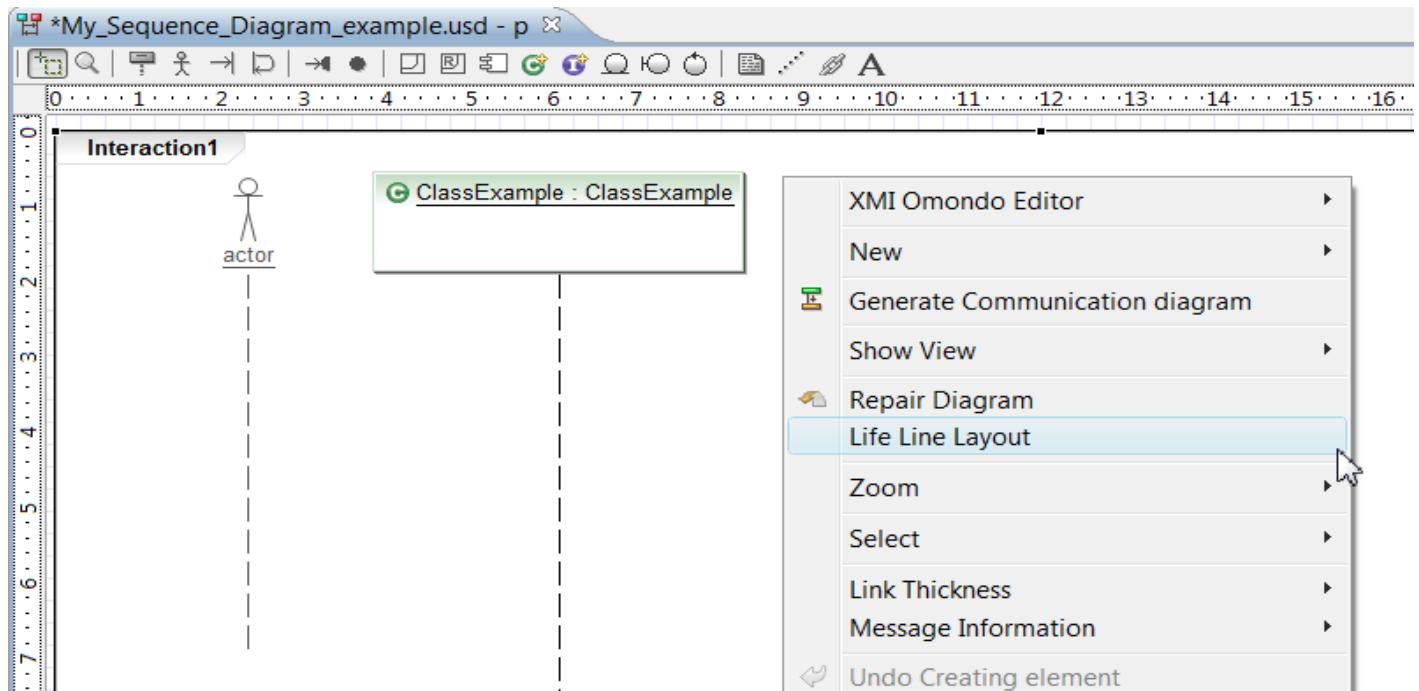
2. Change Property name

It is possible to change the name of the property in the UML Superstructure Model by using the properties View and changing the name manually in the Name field. *Please note that the Properties view could be shared by few other plugins. EclipseUML can't therefore customize this graphical presentation in order to keep plugin compatibility. It means that you have to select the Name Value and enter directly the new value in the View even if a property wizard would certainly have been more adapted. The respect of plugins compatibility requires sometimes not being too intrusive.*



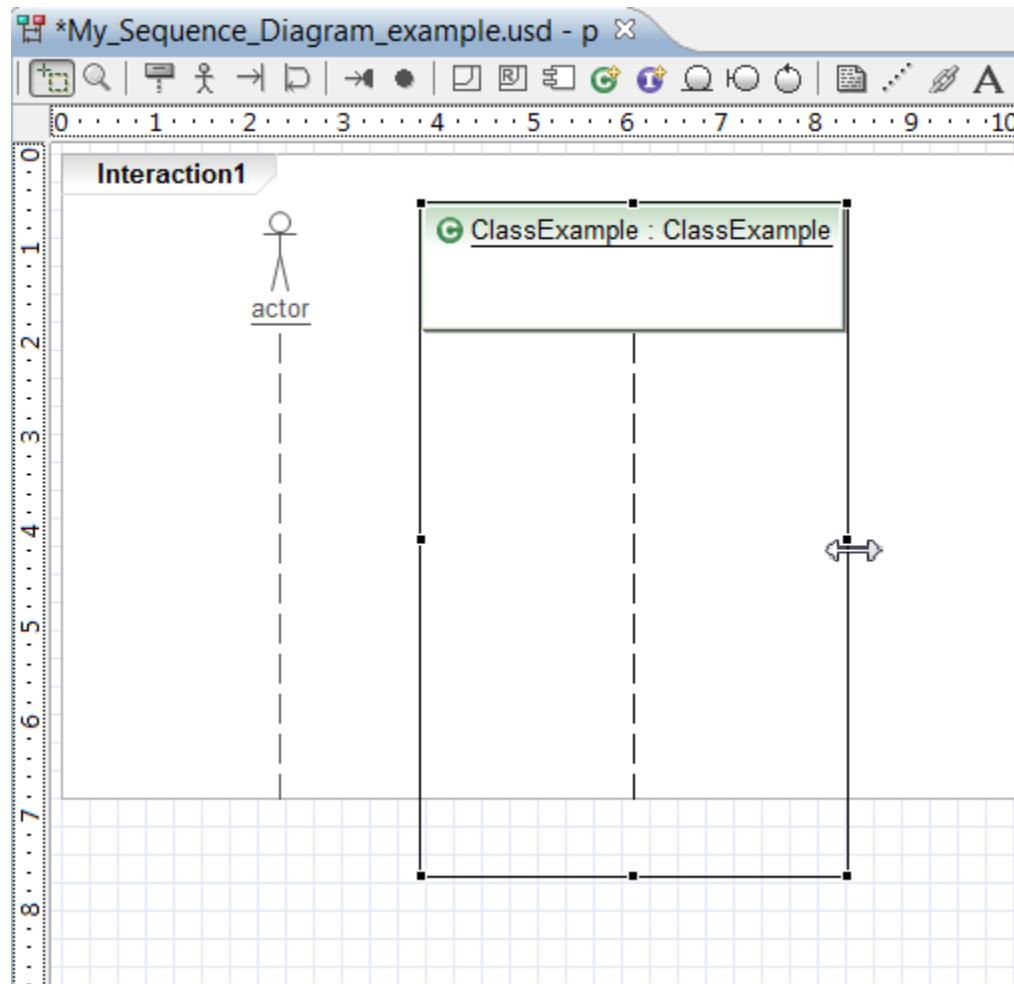
3. Have the same Lifeline height

EclipseUML will not launch any refresh of the Sequence Diagram in order to keep users graphical presentation. It is therefore sometimes important to manually change the height of each Lifeline using the **Sequence Diagram contextual menu > Life Lane Layout**.



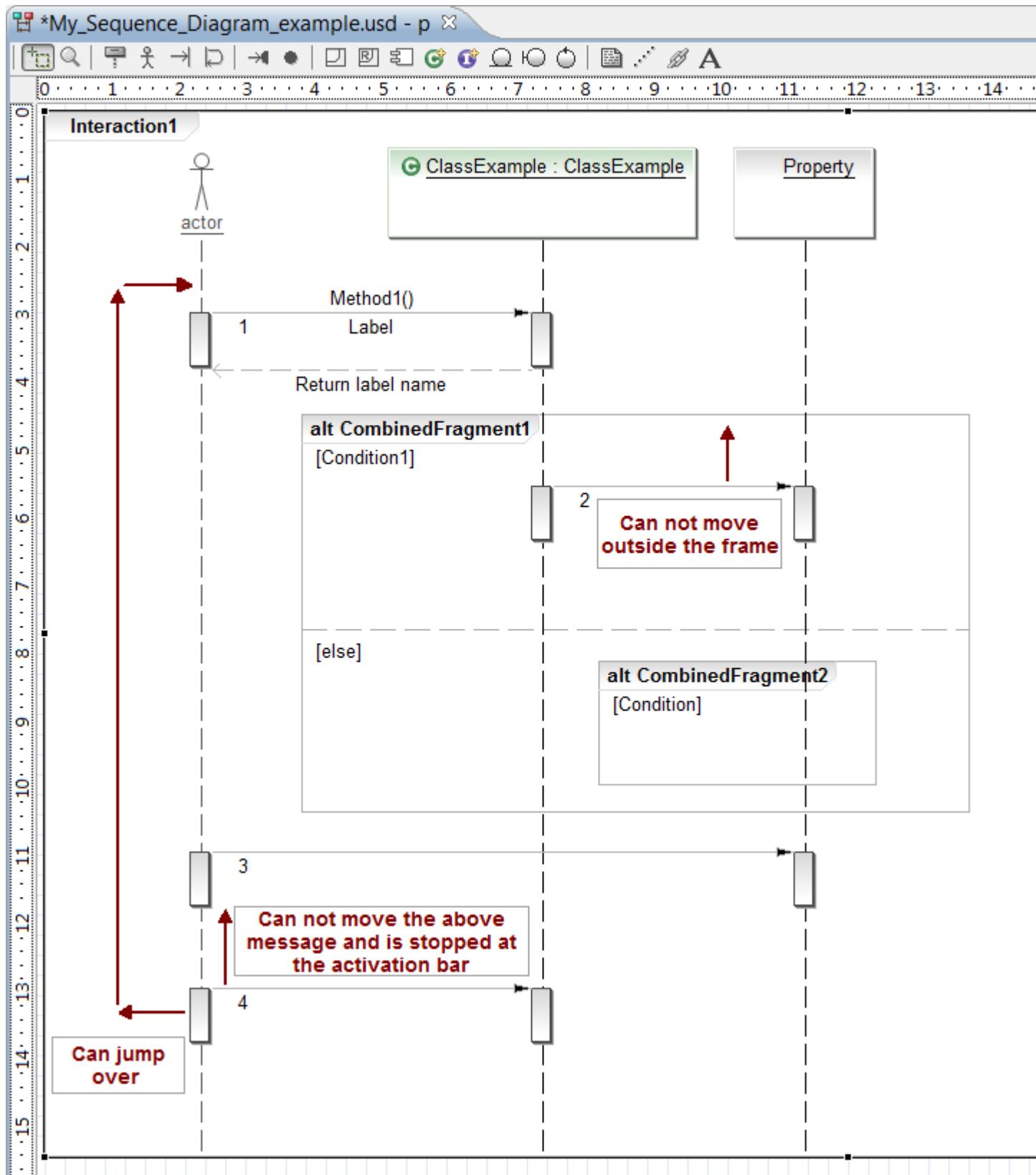
4. Change the property size

It is possible to manually resize the height and the width of each property by using the mouse. Click on the property and select one of the black points to resize the shape with the mouse. You can notice that the Lifeline is sometimes longer than the frame. You can use the [Life Line layout](#) to be sure that the Life line is inside the frame.



5. Move a message

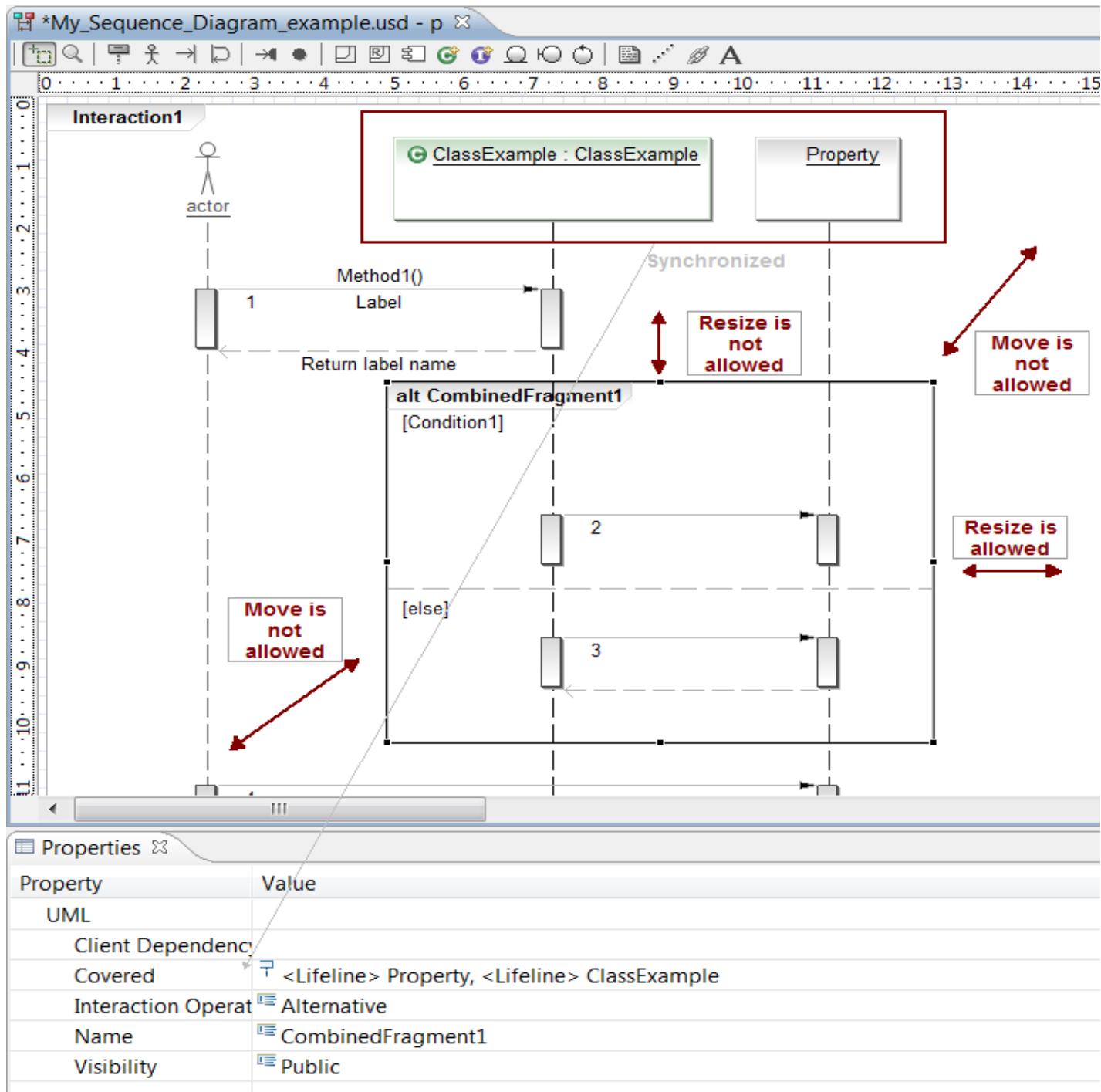
You can move a message up and down at any time if the message is not in a frame. If the message is in a frame then it can't go outside the frame anymore.



6. Move a frame

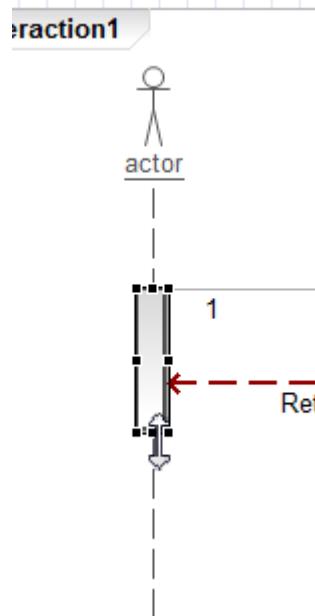
You can graphically move a frame but don't forget that **messages are created inside the frame at the frame or message creation**. It means that if you move a frame then the message will remain in the frame in the UML Superstructure of your model but not graphically. We therefore refuse to block the frame move but don't recommend using this graphical feature. **The best is to erase the frame and create a new frame as soon as you need to move the frame.**

You can resize the frame on the left and the right because the graphical and UML Superstructure model is updated but **shouldn't resize it up and down** because of the message order. You can also not move inner frames outside the frame of creation. Moving frame outside will layout the diagram and the inner frame will disappear from the frame and be placed in the upper right corner.



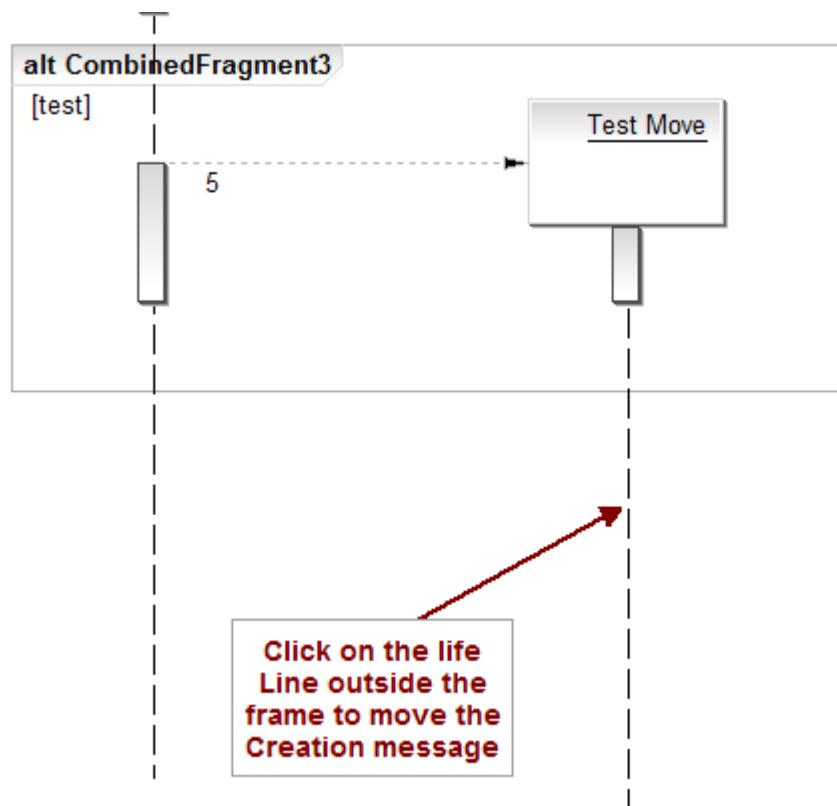
7. Resize an activation bar

You can resize activation bar using the mouse. Please note that the activation bar is only a graphical element and is not anymore in UML 2 a UML Superstructure element Model.



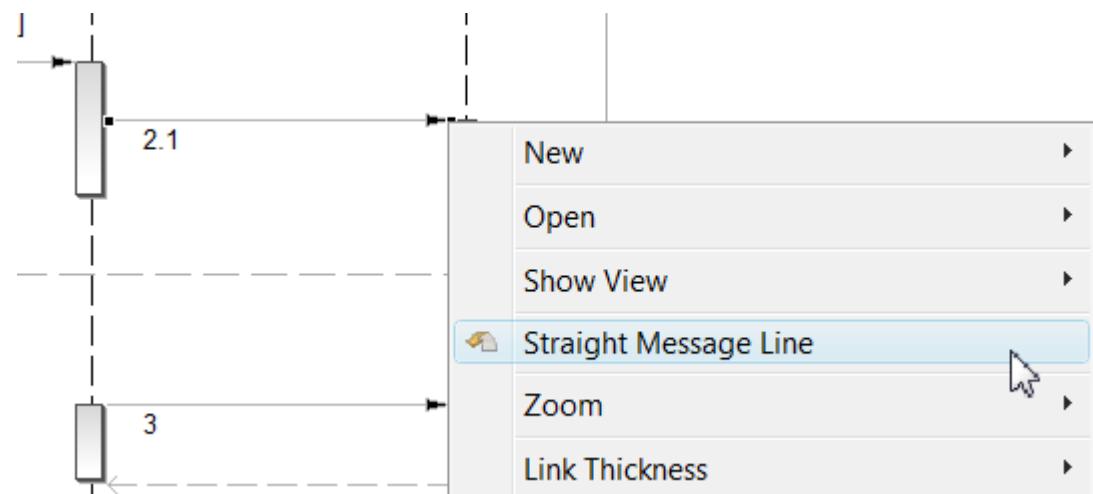
8. Move a creation message

You can move a creation message if you select the life line with the mouse.



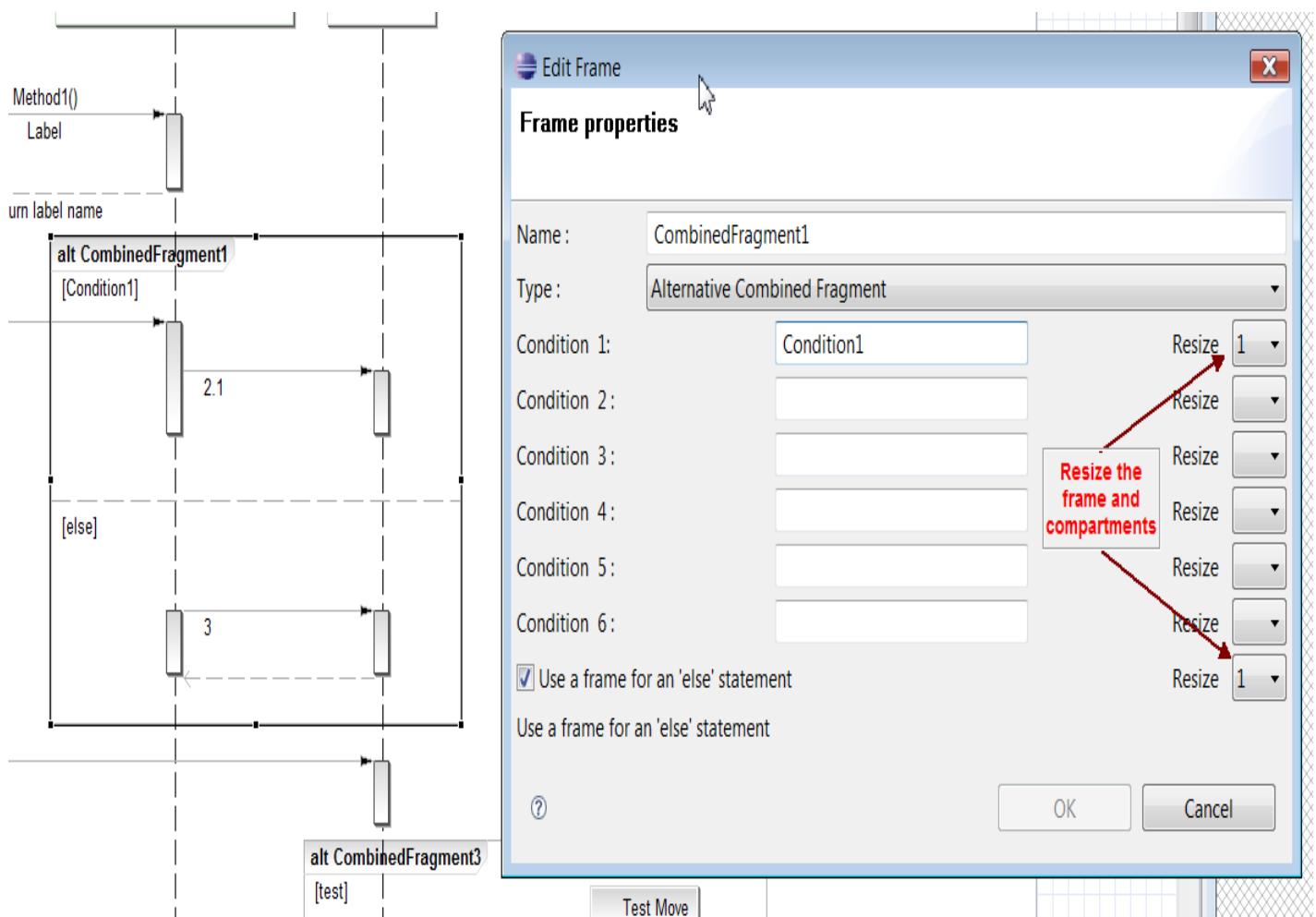
9. Message straight line

If a message is not straight anymore you can refresh the target connector.
Click on the **Message > Straight Message Line**



10. Frame compartments size update

if after adding, deleting, moving or changing properties of a message the frame is not anymore as it was at the creation then you can refresh the size of each inner compartment. Click on the frame to open the Frame Properties. *You can notice that the Ok button is grey but if you erase and add the same information then the Ok button is now activated.* If you click on the Ok button then the size of inner compartments will be refreshed immediately.



11. Change the name of the frame

It is possible to change the name of the frame using the Properties View.
Select the **Name Value Field** and type the new name.

The screenshot shows the UML Properties View on the left and a sequence diagram frame on the right. A red arrow points from the 'Name' row in the Properties View to the frame's title bar. The Properties View table is as follows:

Property	Value
UML	
Classifier Behavior	
Client Dependency	
Covered	
Is Abstract	false
Is Active	false
Is Leaf	false
Is Reentrant	false
Name	MySequenceDiagramFrame
Owned Port	
Powertype Extent	
Redefined Behavior	
Redefined Classifier	

The sequence diagram frame on the right has a title bar labeled "MySequenceDiagramFrame".

12. Ok button of the frame is not activated

The Ok button of the frame is not activated if:

- you need to enter at least one condition
- the [frame listener doesn't see any change](#) with the previous frame property.

UseCase Diagram

In this section you will learn how to use the EclipseUML UseCase diagram.
The following areas are covered:

- [Concept](#)
- [Toolbar](#)
- [Extend UML 2](#)
- [Show Hide](#)
- [Drag and drop](#)
- [UseCase diagram example](#)

Concept

The Use Case Diagram describes the behavior of a system from a user's standpoint: functional description and its major processes.

It provides a graphic description of who will use a system and what kind of interactions to expect within that system. Use Case Diagrams are useful when describing requirements for a system in the analysis, design, implementation and documentation stages.

A Use Case Diagram is a collection of actors, Use Cases, and their communications.

The elements of the Use Case diagrams are available in the tool bar:

- ❶ New Actor can be created by selecting the icon in the toolbar.
- ❷ New Use Case can be created by selecting the icon in the toolbar.
- ❸ New system can be created.
- ❹ Create inheritance relationships between Use Cases or actors.
- ❺ The Includes Relationship is used to indicate that the source Use Case includes the behavior of the target Use Case.
- ❻ The Extends Relationship indicates that the source Use Case adds its behavior to the target Use Case.
- ❼ Association is linking an actor to its Use Cases.

Toolbar item

This section describes the provided tools in order to build a Usecase Diagram. For further information please refer to the UML specification available on the [OMG](#) web site.

❶ Actor

Allow to add a new actor on the diagram.

❷ Usecase

Allow to add a new usecase on the diagram.

□ System

Allow to add a new system on the diagram.

↗ Generalization

Allow to create inheritance relationships between usecases or actors.

↖ Includes

The includes relationship is used to indicate that the source usecase includes the behavior of the target usecase.

↖ Extends

The extends relationship indicates that the source usecase adds its behavior to the target usecase.

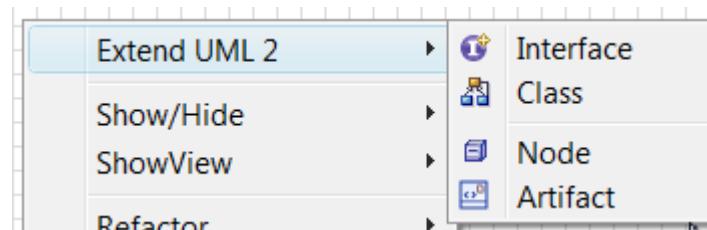
↙ Association

This tool aims is linking an actor to its usecases.

Extend UML2

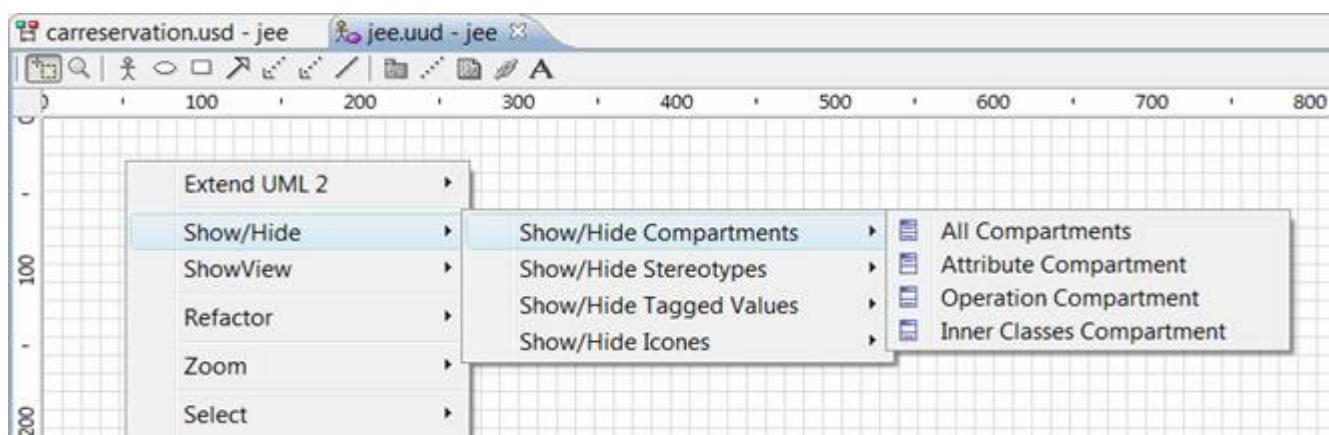
The following option is an UML 2 extension, in order to add new elements inside the usecase diagram. You can add new UML 2 elements in your usecase diagram

- Interface
- Class
- Node
- Artifact



Show Hide

The following option is an UML 2 extension, in order to add new elements inside the usecase diagram. The Show/ Hide usecase diagram contextual menu allows you to hide attribute and method compartment from a class or an interface.

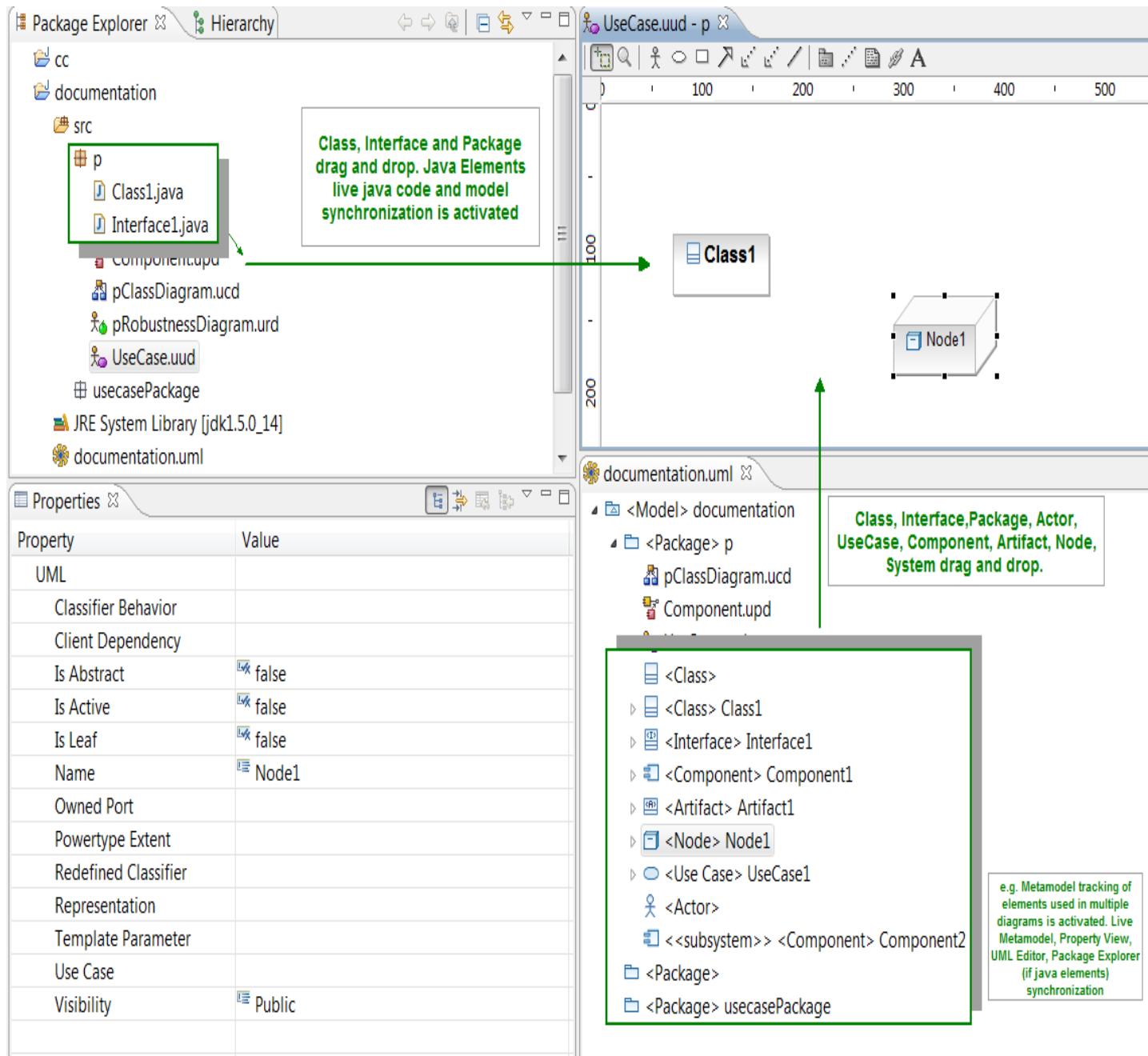


Drag and Drop

The drag and drop concept is important in order to track UML elements used in more than one diagram. It could be important to use the same actor in two different usecase diagrams and keep the metamodel information updated. If you click on the element in the XMI Editor, you will see the diagram in which this element is used. If you move an actor or any other element into another diagram, then the new UML Editor diagram element will include all updated metamodel information which has been created in multiple diagrams. *The drag and drop concept and the use of the same metamodel + UML Editor + Java instance allows what was not previously possible with traditional UML tools !!*

In a usecase diagram, you can drag and drop both Java and UML elements from:

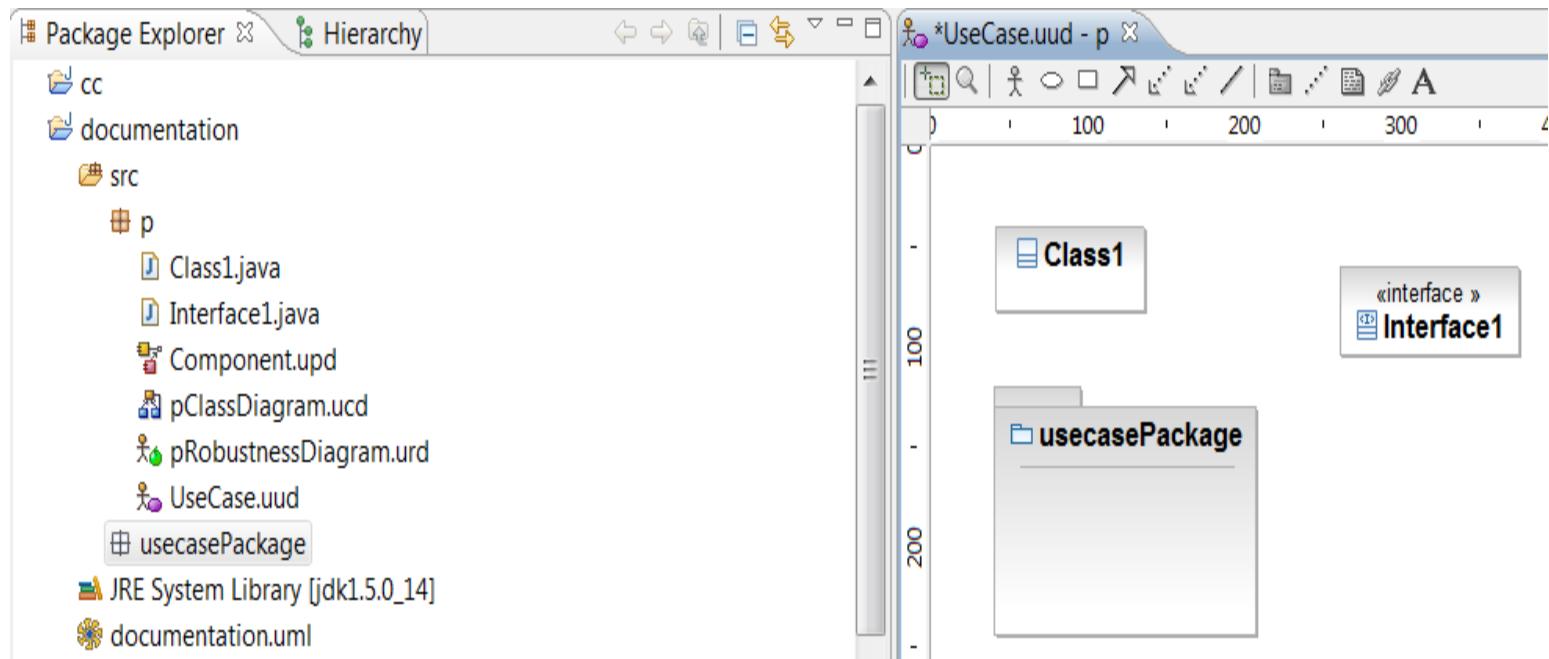
1. [Package Explorer](#)
2. [Model Editor](#)



1. From the Package Explorer you can drag and drop the following Java elements:

- Package
- Class
- Interface

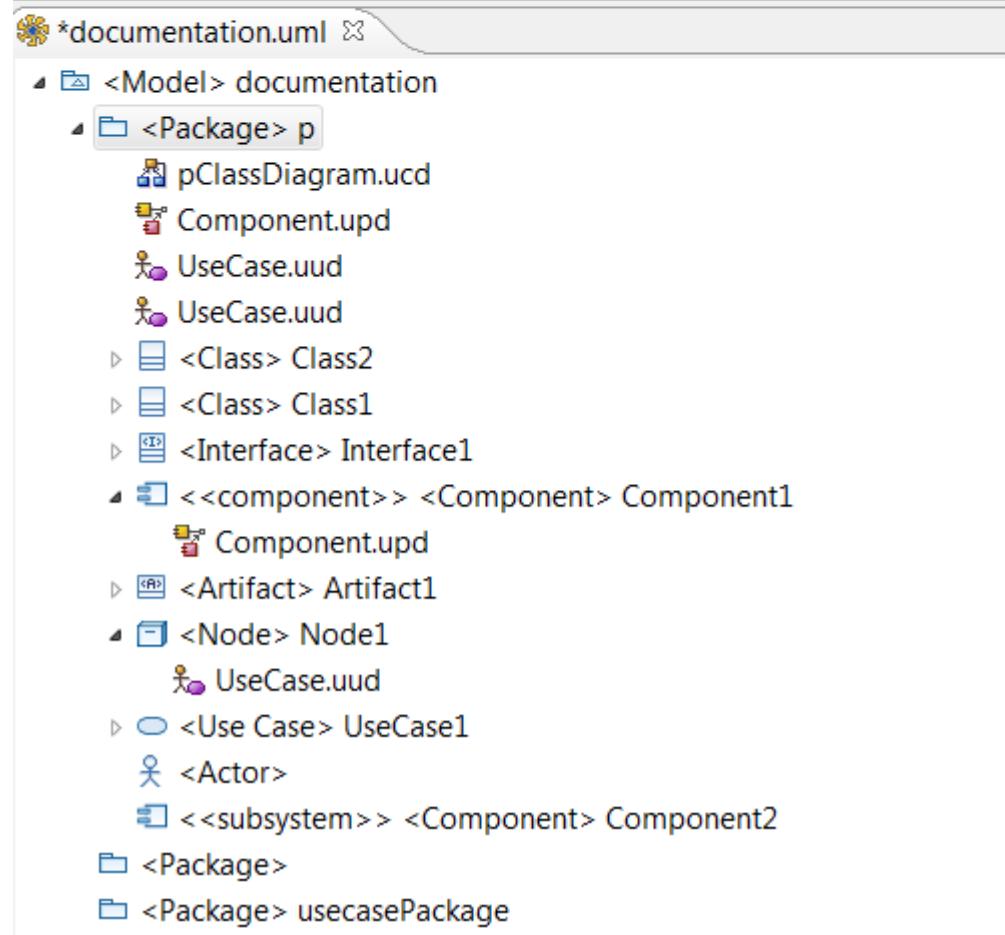
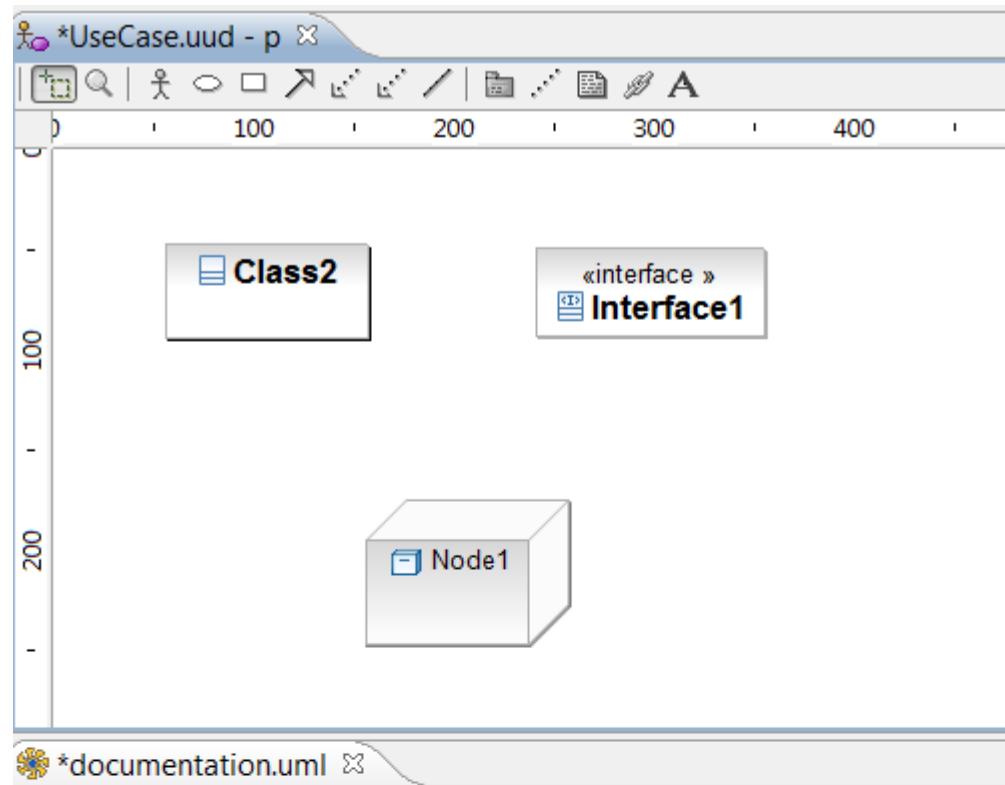
Select one element in your Package Explorer and drag it to the UseCase Diagram. Release the mouse button to drop it into your diagram editor (e.g. *Class1*, *Interface1* & *usecasePackage*).



2. From the Model Editor you can drag and drop the following Java or UML (*just a metamodel element not related to Java*) elements:

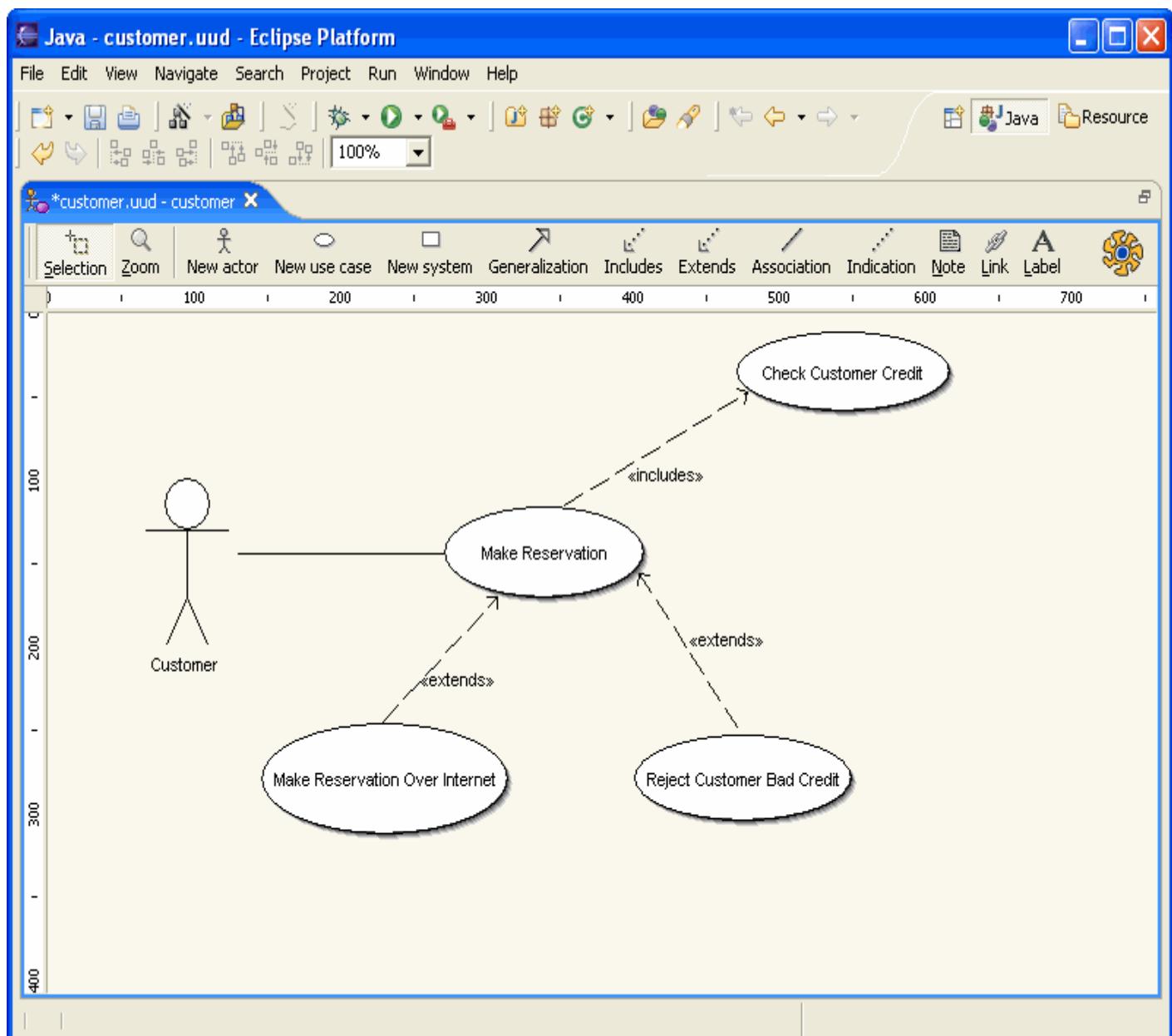
- Package
- Class
- Interface
- Actor
- UseCase
- System
- Component
- Artifact
- Node

Select one element in your Model Editor and drag it to the Usecase Diagram. Release the mouse button to drop it into your diagram editor (e.g. *Node1*)



Use Case Diagram Example

Below is an example of the Use Case as part of a diagram with one actor and four Use Cases. In this example the Customer is an actor who Makes a Reservation.
[See also the Robustness Diagram example.](#)



The file deployment diagram extension is *.uud.

State Diagram

In this section you will learn how to use the EclipseUML State diagram.
The following areas are covered:

- [Concept](#)
- [Toolbar](#)
- [Drag and Drop](#)
- [State diagram example](#)

Concept

A state diagram shows the sequences of states that an object or an interaction goes through during its life in response to received stimuli, together with its responses and actions. The state of an object depends on its current activity or condition. A state diagram shows the possible states of the object and the transition that cause a change in state.

State chart diagrams are often used to

- explore the complex behavior of a class, actor, subsystem or component etc...
- model real-time systems

The elements of the State diagrams are available in the tool bar:

- An initial state item represents the first active state within a diagram.
- A final state item represents the end of the active state
- State items indicate the states of the elements modeled by the diagram
- This tool creates a history item which is used to access the previous state of a state item.
A history state item can only be created into a state item.
- ⇨ A join/fork item is used to synchronize (ie join) or fork transitions.
- ⟳ Transitions represent a change in state of the elements modeled by the diagram.

Toolbar items

This section describes the tools provided in order to build a State Diagram. For further information please refer to the UML specification available on the [OMG](#) web site.

● Initial state

An initial state item represents the first active state within a diagram.

● Final state

A final state item represents the end of the active state

□ State

State items indicate the states of the elements modeled by the diagram.

🕒 History state

This tool creates a history item which is used to access the previous state of a state item.
A history state item can only be created into a state item.

‡ Join/Fork

A join/fork item is used to synchronize (ie join) or fork transitions.

○ Transition

Transitions represent a change in state of the elements modeled by the diagram.

Drag and Drop

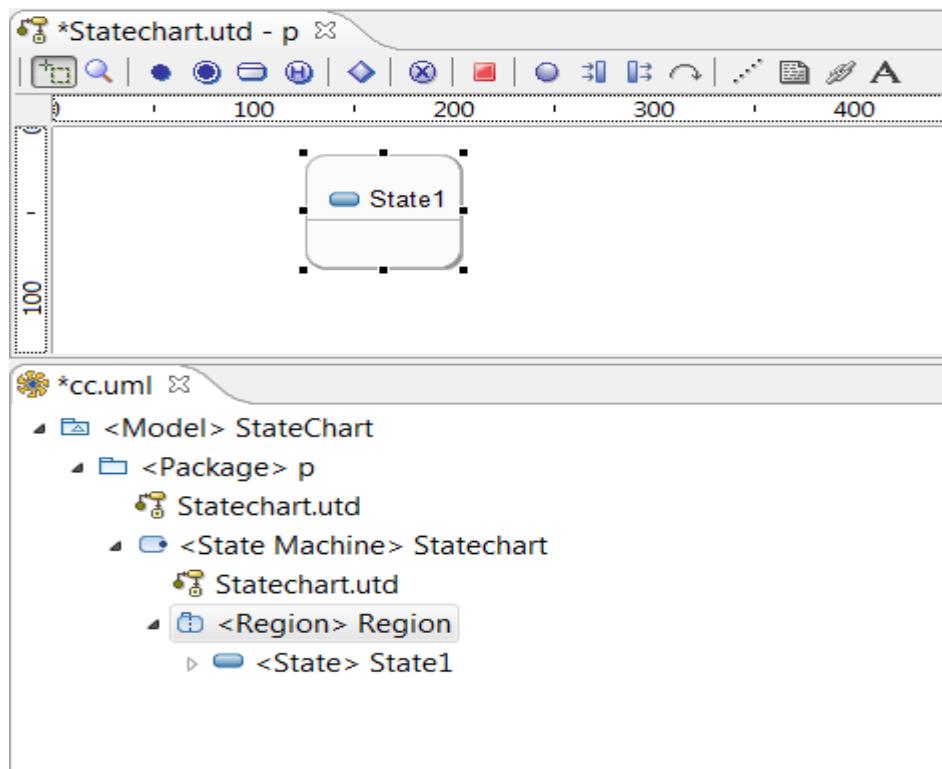
The drag and drop concept could be important in order to track the state elements used in more than one diagram.

It is not recommended to use the same element in different diagrams, but it could be important for DSL modeling to have this option.

In a state diagram, you can drag and drop from **the XMI 2.1 Editor** the following element:

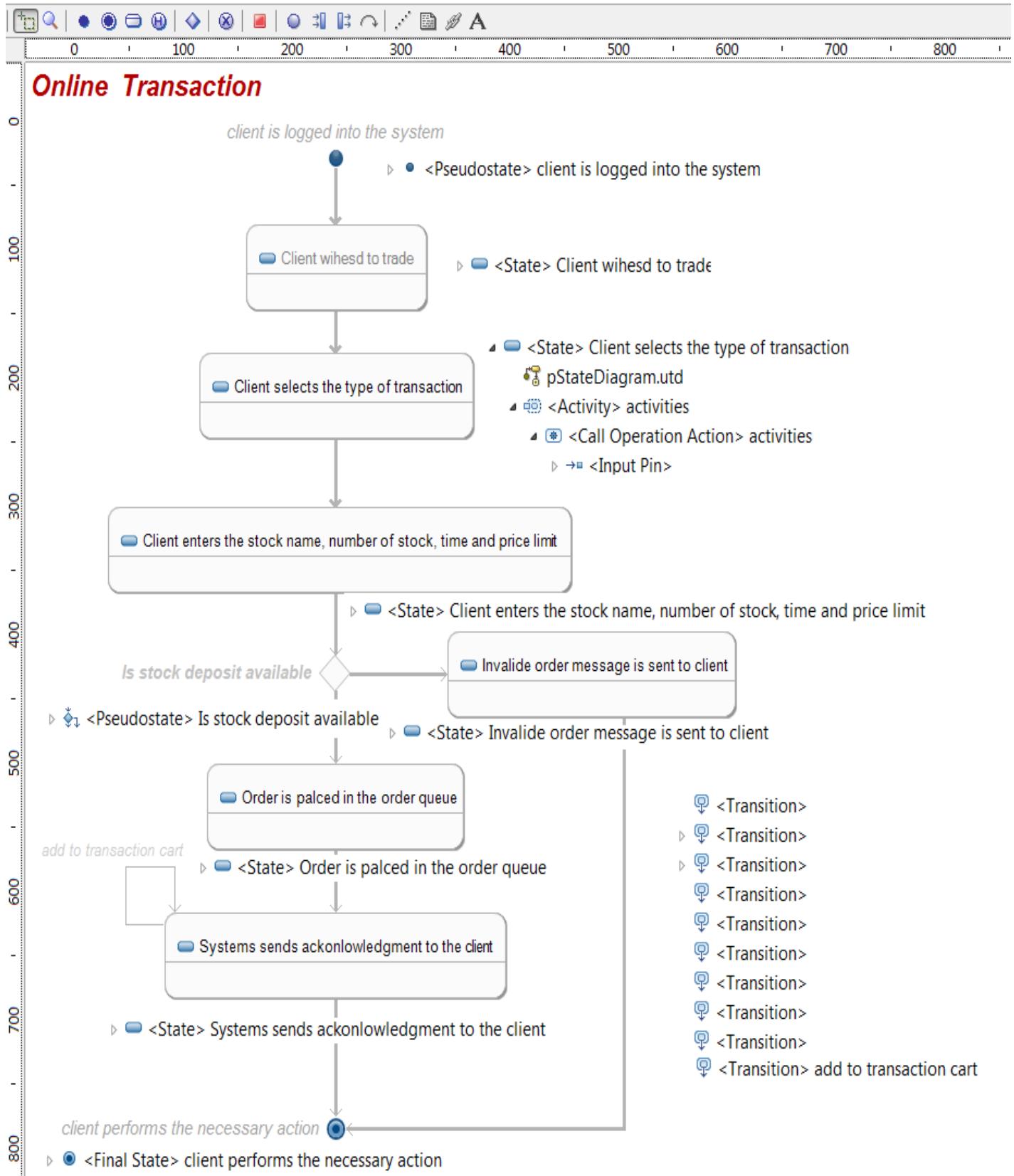
- State

Select a state element in your XMI 2.1 Editor and drag it to the Statechart Diagram.
Release the mouse button to drop it into your diagram editor (*e.g. State1*)



State Diagram Example

The diagram that follows models the Online Transaction. It shows the UML 2.1 graphical Editor and the XMI Editor information (*manually pasted in the diagram picture*).



Activity Diagram

Concept

Activity diagram

UML activity diagrams are the object-oriented equivalent of flow charts and data-flow from structured development. They describe activities and flows of data or decisions between activities. Activity diagrams and state diagrams are related. While a state diagram focuses on an object undergoing a process, an activity diagram focuses on the flow of activities involved in a single process.

Activity diagrams provide a very broad view of business processes. It can be used to break down the activities that occur within a use case. An Activity Diagram could be used for describing work flows across many use cases, analyzing a use case or dealing with multi-threaded applications. Activity diagrams can be used to explore the logic of:

- a complex operation
- a single use case
- several use cases
- a business process
- complex business rules
- software processes

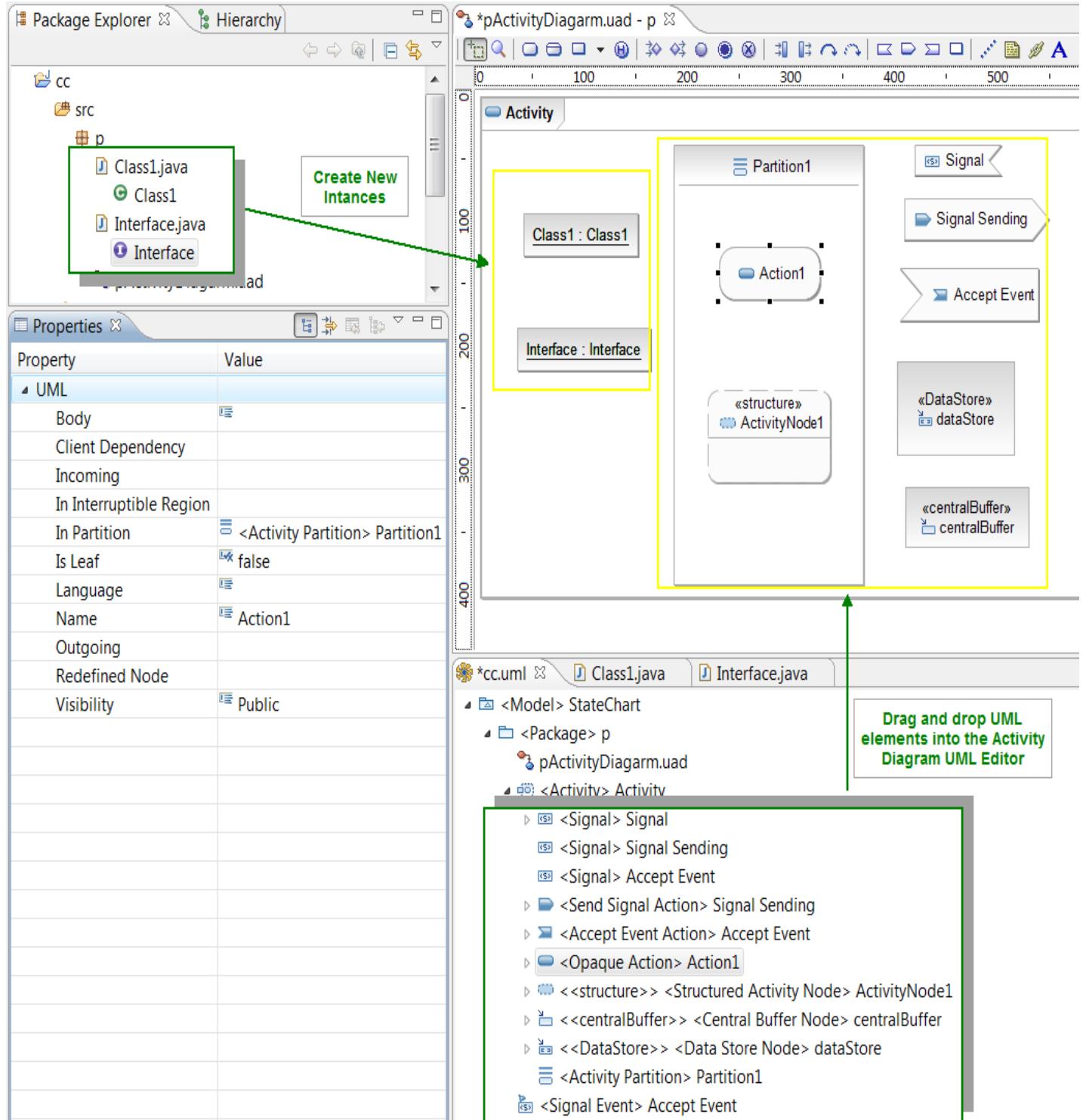
The elements of the Activity diagrams are available in the tool bar:

- New Activity can be created by selecting the icon in the toolbar.
- ◊ New Decision can be created by selecting the icon in the toolbar.
- New Start can be created by selecting the icon in the toolbar.
- New End can be created by selecting the icon in the toolbar.
- New State can be created by selecting the icon in the toolbar.
- New History can be created by selecting the icon in the toolbar.
- ‡ New join/fork can be created by selecting the icon in the toolbar.
- A transition can be created by selecting the icon in the toolbar.
- New Object can be created by selecting the icon in the toolbar.
- Object flow can be created by selecting the icon in the toolbar.
- New signal receipt can be created by selecting the icon in the toolbar.
- New signal sending can be created by selecting the icon in the toolbar.
- New Partition can be created by selecting the icon in the toolbar.

Drag and Drop

The drag and drop concept could be important in order to track the activity diagram elements used in more than one diagram. It is not recommended to use the same activity diagram element in different diagrams, but it could be important for DSL modeling to have this option. In an Activity diagram, you can drag and drop both Java and UML elements from:

1. [Package Explorer](#)
2. [Model Editor](#)



1. From the Package Explorer you can drag and drop the following Java elements:

- Class
- Interface

Select one element in your Package Explorer and drag it to the Activity Diagram.

Release the mouse button to drop it into your diagram editor (*e.g. Class1, Interface*). The classes or interfaces from the Package Explorer to the Diagram Editor will display a new instance in your Activity Diagram.

2. From the Model Editor you can drag and drop the following Java or UML (*just a metamodel element not related to Java*) elements:

- Action
- Structured Activity Node
- Signal
- Signal Sending
- Accept Event
- CentralBuffer
- DataStore
- Partition

Select one element in your XMI 2.1 Editor and drag it to the Activity Diagram.

Release the mouse button to drop it into your diagram editor (*e.g. Action1*)

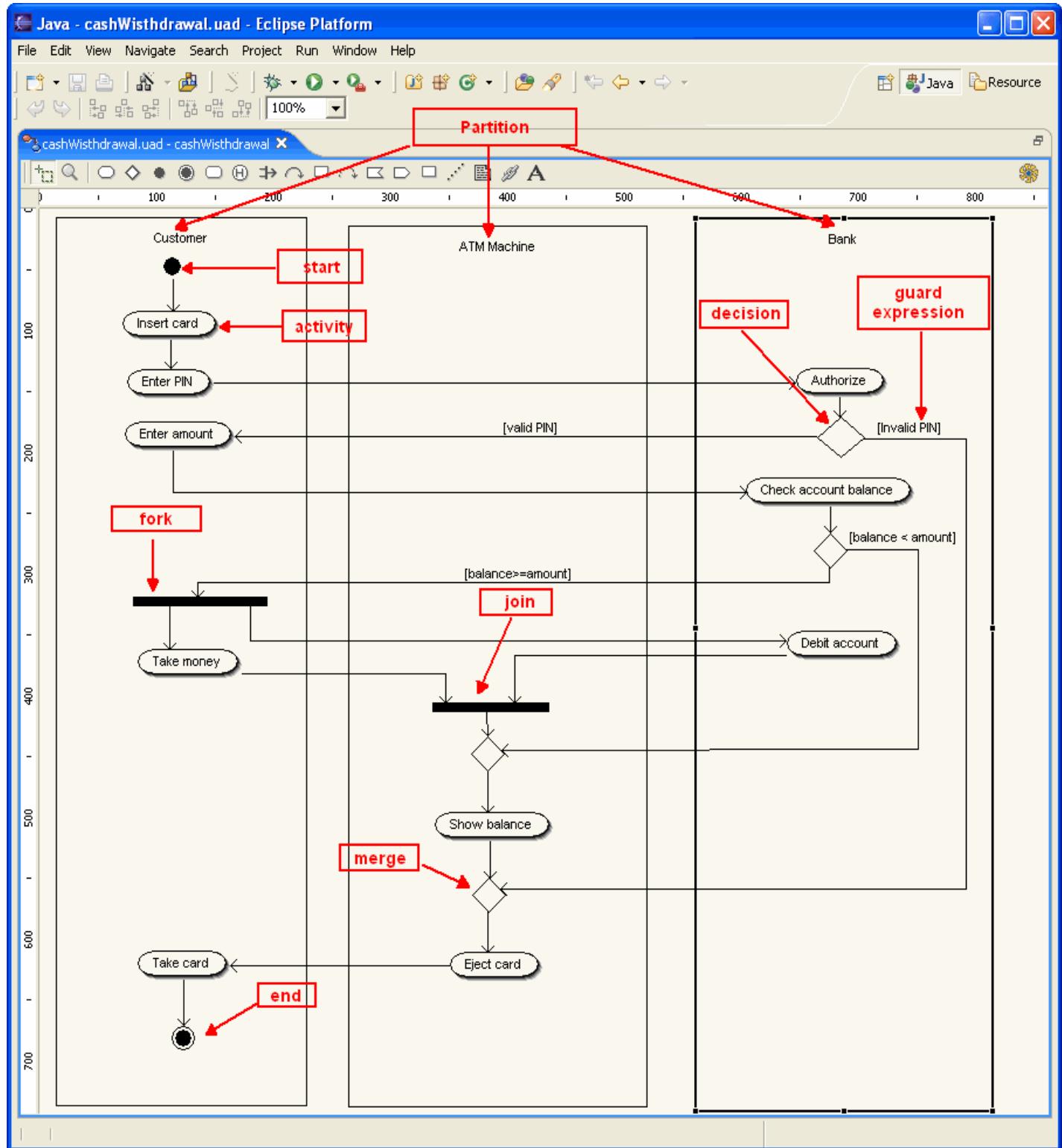
Toolbar items

This section describes the tools provided in order to build an Activity Diagram. For further information please refer to the UML specification available on the [OMG](#) web site.

- New Activity can be created by selecting the icon in the toolbar.
- ◊ New Decision can be created by selecting the icon in the toolbar.
- New Start can be created by selecting the icon in the toolbar.
- New End can be created by selecting the icon in the toolbar.
- New State can be created by selecting the icon in the toolbar.
- New History can be created by selecting the icon in the toolbar.
- ‡ New join/fork can be created by selecting the icon in the toolbar.
- A transition can be created by selecting the icon in the toolbar.
- New Object can be created by selecting the icon in the toolbar.
- Object flow can be created by selecting the icon in the toolbar.
- New signal receipt can be created by selecting the icon in the toolbar.
- New signal sending can be created by selecting the icon in the toolbar.
- New Partition can be created by selecting the icon in the toolbar.

Activity Diagram Example

This diagram is useful in showing work flow connections and describing behavior that has a lot of parallel processing. When you use an activity diagram you can choose the order in which to do things. The following example shows the withdrawal of money from an ATM Machine. The three classes involved are Customer, ATM Machine and Bank... Activates are shown as rounded rectangles.



The file deployment diagram extension is *.uad.

Communication Diagram

In this section you will learn how to use the EclipseUML Communication diagram.
The following areas are covered:

- [Concept](#)
- [Drag'n'Drop](#)
- [Toolbar](#)
- [Communication diagram Example](#)

Concept

A Communication diagram is the cross between a symbol and a sequence diagram. It describes a specific scenario. Numbered arrows show the movement of messages during the course of a scenario. Communication diagrams show the same information as sequence diagrams, but focus on object roles instead of the times that messages are sent.

UML Communication diagrams, like UML sequence diagrams, are used to explore the dynamic nature of your software.

Communications diagrams are often used to:

- provide an overview of a collection of collaborating objects
- allocate functionality to classes
- model the logic of implementation
- examine the roles that objects take within a system

In a sequence diagram, object roles are the verticals and messages the connecting links. In a Communication diagram, the object-role rectangles are labeled with either a class or object name.

The elements of the Communication diagram are available in the tool bar:

- ✖ New Actor can be created by selecting the icon in the toolbar.
- ◻ New Object can be created by selecting the icon in the toolbar.
- ➕ Add new message

Drag and Drop

In a Communication diagram, you can drag and drop a class or an interface from the Package Explorer to the Communication Diagram Editor and get an Instance.

Toolbar items

This section describes the tools provided in order to build a Communication Diagram. For further information please refer to the UML specification available on the [OMG](#) web site.

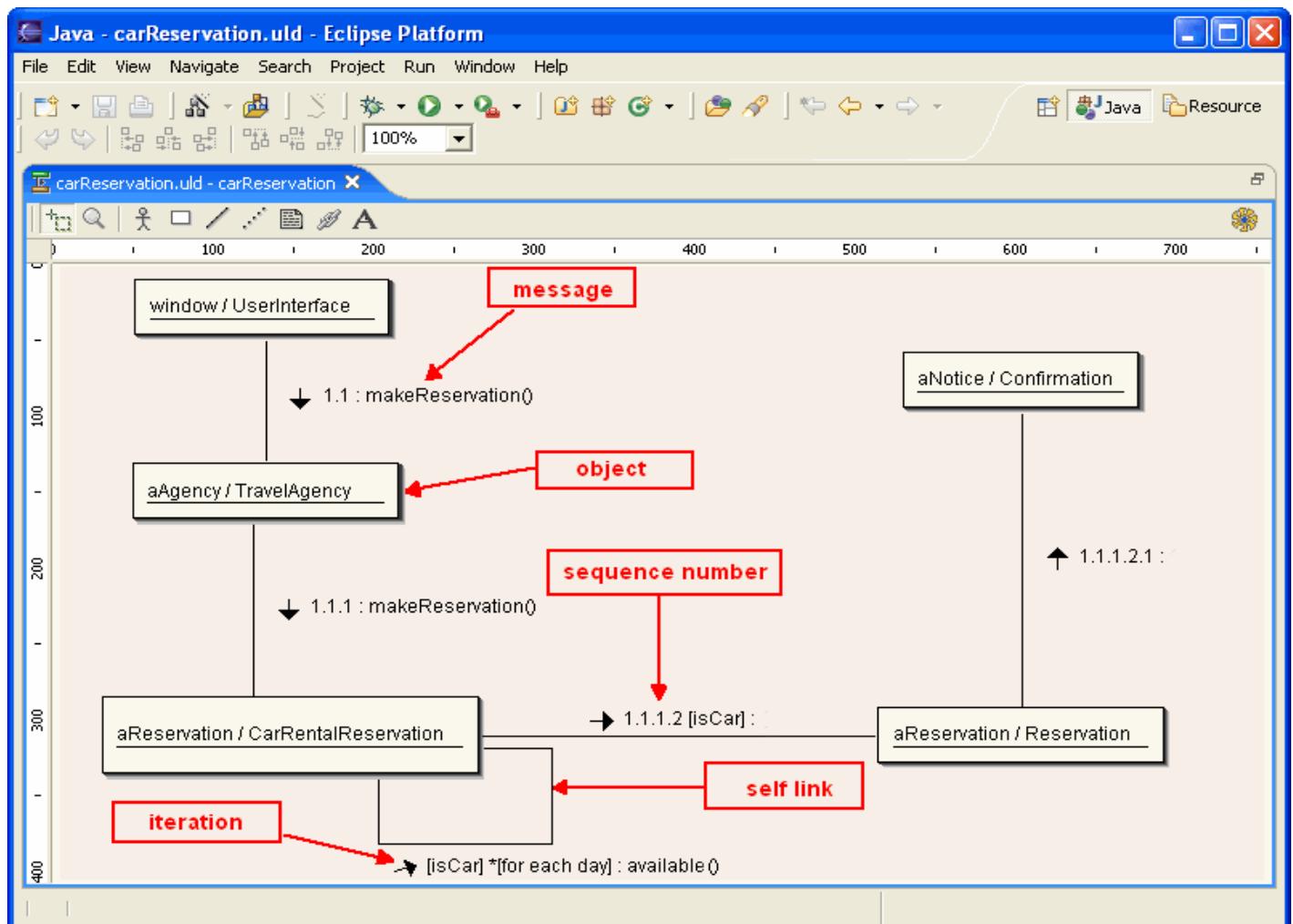
New Actor

Add a new actor within the diagram.

- New Object: Add a new object
- Message : Add a new message

Communication Diagram Example

A Communication diagram shows an interaction organized around the objects in the interaction and their links to each other. The following example is the same one used in the sequence diagram. The internet browser sends a makeReservation() message to a travel agency, which sends a makeReservation() message to a car rental reservation. If the car is available, then it makes a Reservation and a confirmation. The car rental reservation issues a self call to determine if a car is available. If so, the car rental reservation makes a Reservation and a Confirmation.



Notes can be included.

Object Diagram

In this section you will learn how to use the EclipseUML Object diagram.
The following areas are covered:

- [Concept](#)
- [ToolBar](#)

Concept

An Object diagram shows the existence of objects, their relationships in the logical view of the system and how they execute a particular scenario or use case.

Object Diagram, also called instance diagram shows an overview of a system. Using the Object diagram model, you describe the static structure of the symbols in your system.

The elements of the Object diagram are available in the tool bar:

- New Instance can be created by selecting the icon in the toolbar.
- ✓ New association between instances can be created by selecting the icon in the toolbar.
- ↗ New flow indicates how an instance is created (stereotype copy) or shows a change in state (stereotype become).

The file deployment diagram extension is *.uod.

Toolbar items

This section describes the tools provided in order to build an Object Diagram. For further information please refer to the UML specification available on the [OMG](#) web site.

□ New instance

Add a new instance on the diagram.

✓ New association

Create an association between instances.

↗ New flow

Indicates how an instance is created (stereotype copy) or shows a change in state (stereotype become).

Component Diagram

In this section you will learn how to use the EclipseUML Component diagram.
The following areas are covered:

- [Concept](#)
- [Drag'n'Drop](#)
- [Extend UML 2](#)
- [Show View](#)
- [Toolbar](#)
- [Component Diagram Example](#)

Concept

UML Component diagrams show the dependencies among software components, including the classifiers that specify them, such as implementation classes; and the artifacts that implement them, such as source-code files, binary-code files.

It is a simple, high-level diagram, which refers to physical components in a design. It is used to show how code is actually divided into modules. While package diagrams show logical or conceptual division, component diagrams are used to show physical division used for implementation.

A Component diagram shows the dependencies among software components, including source code, binary code and executable components. Some components exist at compile time, some exist at link time, and some exist at run time.

Components are physical units including:

- External libraries, COM components
- Enterprise JavaBeans and jar files
- Operating systems and virtual machines

The elements of the Component diagram are available in the tool bar:

- Create component items. Note that components can be nested.
- ↗ Create inheritance relationships between components.
- Add interface to the diagram.
- Create artifact within the diagram.
- ↙ Create an implementation relationship between a component and its interfaces.
- ↖ Indicate dependency relationships among the diagram items.

Drag and Drop

The drag and drop concept is important in order to track UML elements used in more than one diagram.

It could be important to use the same component in two different usecase and component diagrams while keeping the metamodel information updated.

If you click on the element in the XMI Editor, you will see the diagram in which this element is used.

If you move a component or any other element into another diagram, then the new UML Editor diagram element will include all updated metamodel information which has been created in multiple diagrams.

The drag and drop concept and the use of the same metamodel + UML Editor + Java instance allows what was not previously possible with traditional UML tools !!

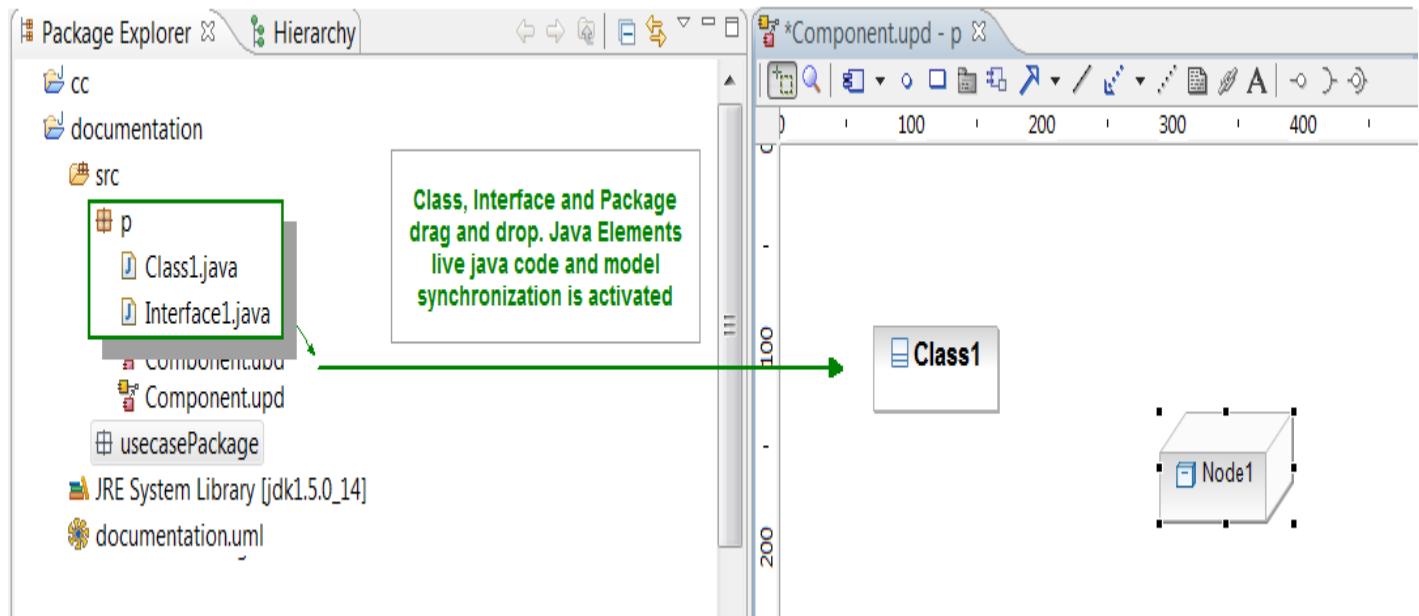
In a Component diagram, you can drag and drop both Java and UML elements from:

1. [Package Explorer](#)
2. [Model Editor](#)

1. **From the Package Explorer** you can drag and drop the following Java elements:

- Package
- Class
- Interface

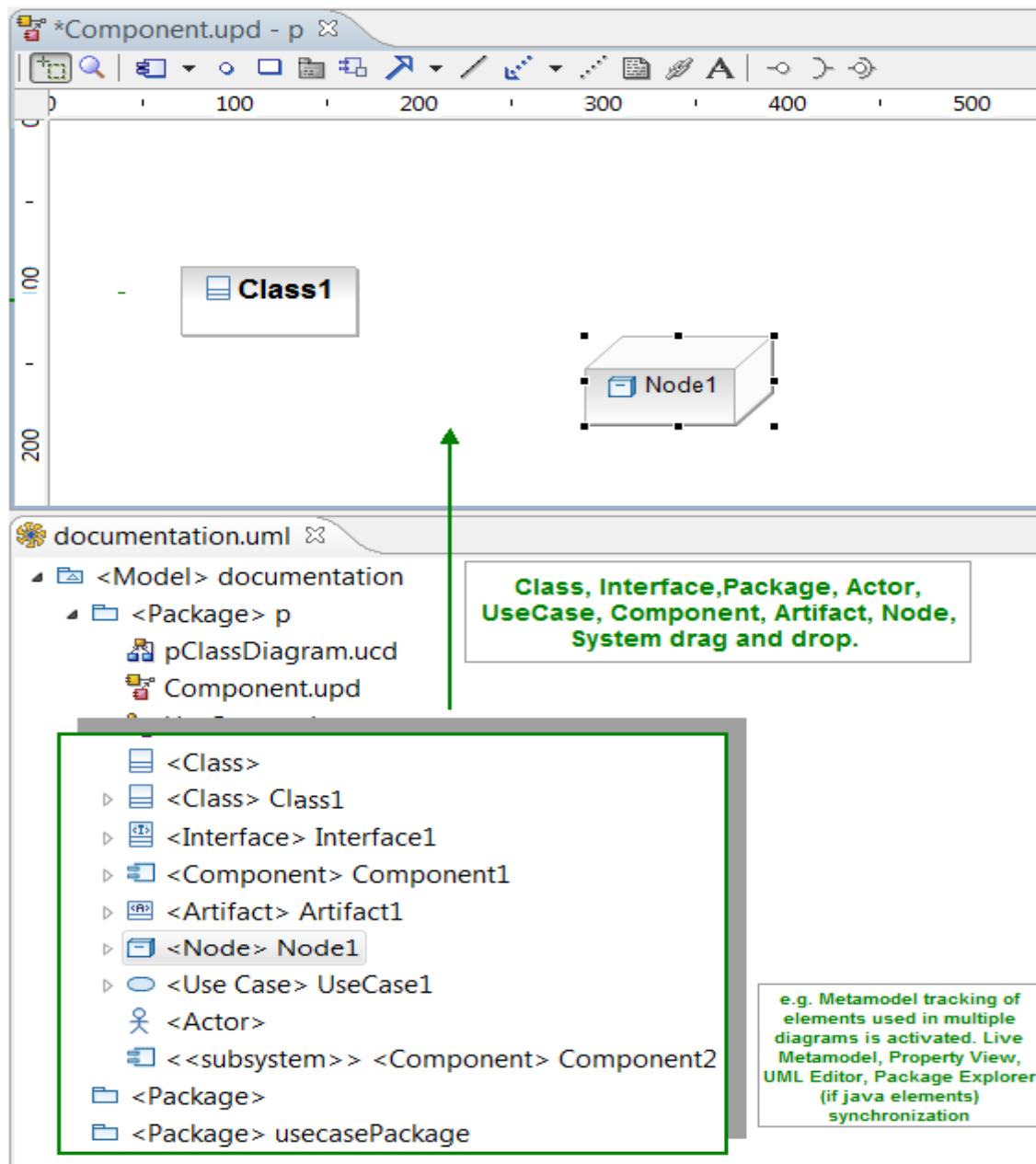
Select one element in your Package Explorer and drag it to the Component Diagram. Release the mouse button to drop it into your diagram editor (e.g. *Class1*).



2. From the Model Editor you can drag and drop the following Java or UML (just a metamodel element not related to Java) elements:

- Package
- Class
- Interface
- Actor
- UseCase
- System
- Component
- Artifact
- Node

Select one element in your Model Editor and drag it to the Component Diagram.
Release the mouse button to drop it into your diagram editor (e.g. Node1)

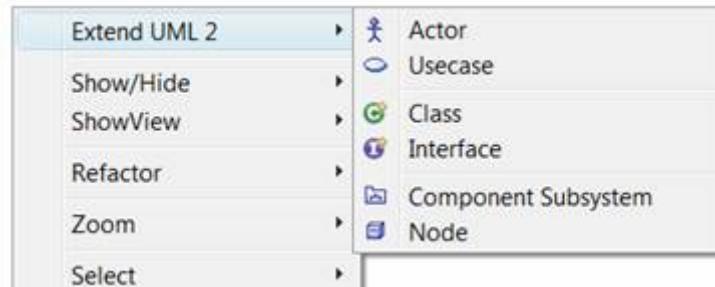


Extend UML2

The following option is an UML 2 extension, in order to add new elements inside the component diagram.

You can add new UML 2 elements in your component diagram

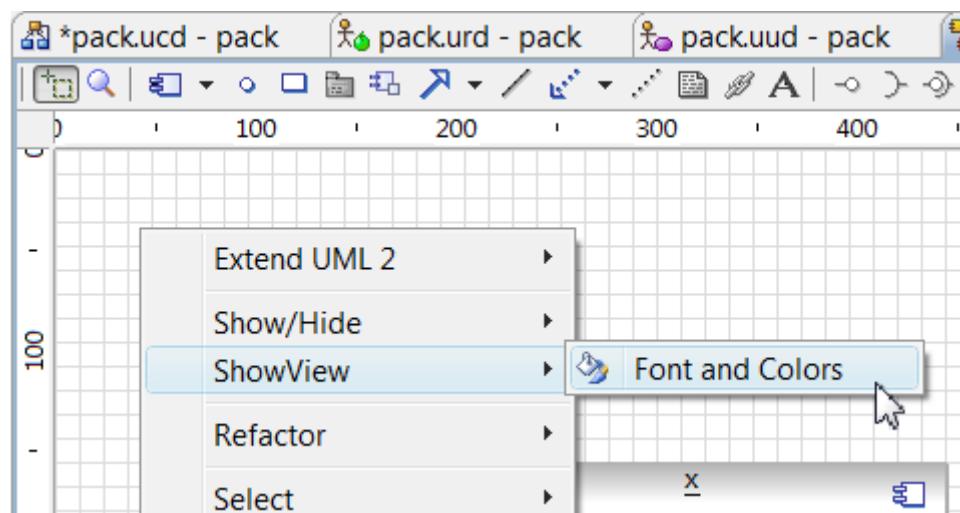
- Interface
- Class
- Node
- Component System
- Actor
- Usecase



Show View

The following option allows adding foreground and background colors.

You can change the background and foreground colours by selecting the component diagram contextual menu > **ShowView** > **Font and Colors**.



Toolbar items

This section describes the tools provided in order to build a Component Diagram. For further information please refer to the UML specification available on the [OMG](#) web site.

 **Component:** Create component items. Note that components can be nested.

 **Generalization:** Create inheritance relationships between components.

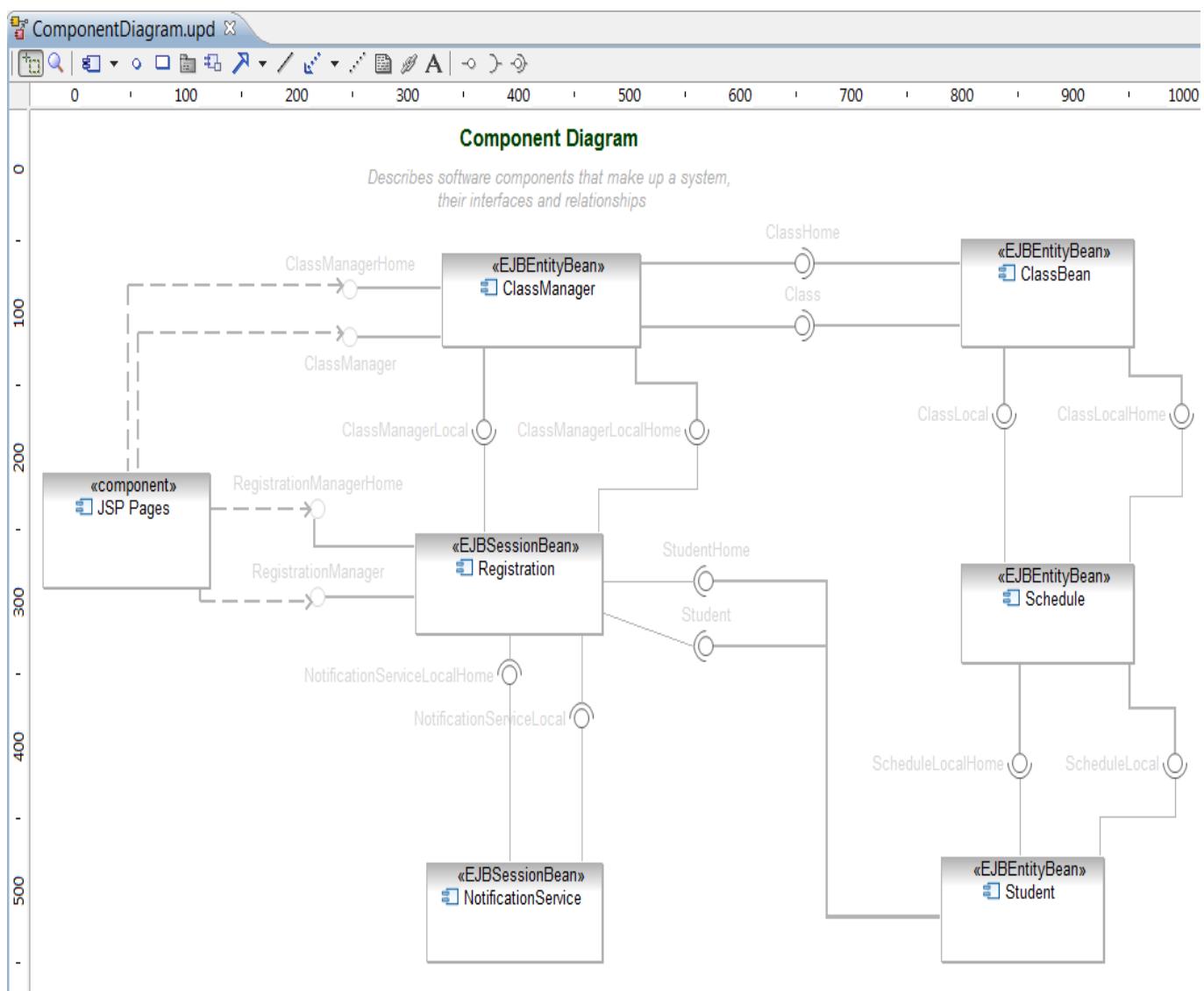
 **Interface:** Add an interface to the diagram.

 **Artifact:** Create an artifact within the diagram.

 **Realize:** Create the implementation relationship between a component and its interfaces.

 **Dependency:** Indicate dependency relationships among the diagram items.

Component Diagrams Example



The file deployment diagram extension is *.upd.

Deployment Diagram

In this section you will learn how to use the EclipseUML deployment diagram.
The following areas are covered:

- [Concept](#)
- [Toolbar](#)
- [Drag and drop](#)
- [Deployment Diagram example](#)

Concept

A UML Deployment diagram depicts a static view of the runtime configuration of hardware nodes and the software components that run on those nodes. It shows the configuration of run-time processing elements and the software components, processes, and objects. Software component instances represent run-time code units. Components that do not exist as run-time entities do not appear.

Developers can model the physical platforms and network connections to be used in their application. Nodes usually represent hardware platforms. Component diagrams can be placed on top of deployment diagrams to indicate which modules of code will go on which hardware platform.

Toolbar items

This section describes the tools provided in order to build a Deployment Diagram. For further information please refer to the UML specification available on the [OMG](#) web site.

Node

Create node items. A node is usually used to represent a physical device.

Communication association

Modelize communication links between nodes.

Component

Create component items. Note that components can be nested.

Generalization

Create inheritance relationships between components.

Interface

Add an interface to the diagram.

Realize

Create the implementation relationship between a component and its interfaces.

Dependency

Indicate dependency relationships among the diagram items.

Drag and Drop

The drag and drop concept could be important in order to track the Deployment diagram elements used in more than one diagram.

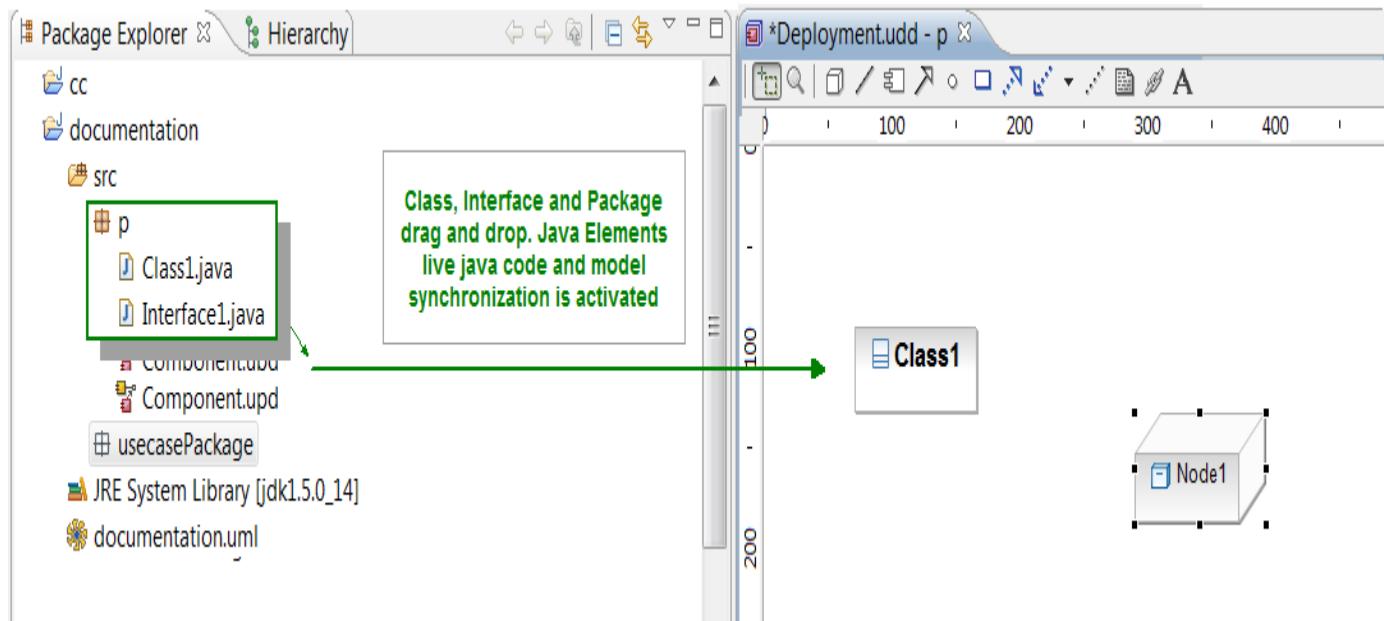
In a Deployment diagram, you can drag and drop both Java and UML elements from:

1. [Package Explorer](#)
2. [Model Editor](#)

1. **From the Package Explorer** you can drag and drop the following Java elements:

- Class
- Interface
- Package

Select one element in your Package Explorer and drag it to the Deployment Diagram. Release the mouse button to drop it into your diagram editor (e.g. *Class1*).

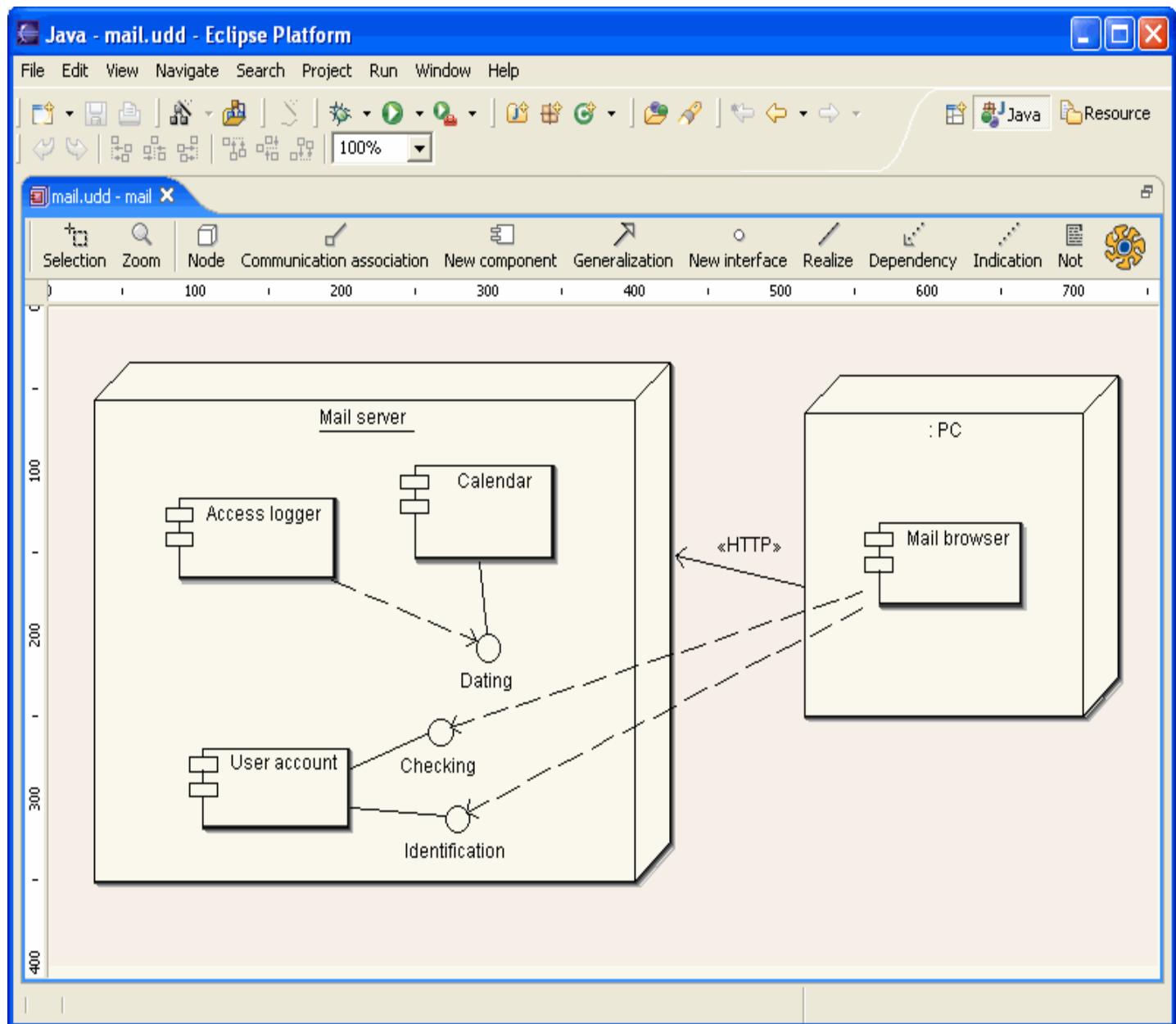


2. **From the Model Editor** you can drag and drop the following Java or UML (*just a metamodel element not related to Java*) elements:

- Package
- Class
- Interfaces (*both Java and UML lollipop*)
- Actor
- UseCase
- System
- Component
- Artifact
- Node

Deployment Diagram Example

The following example is a mail checking deployment diagram.

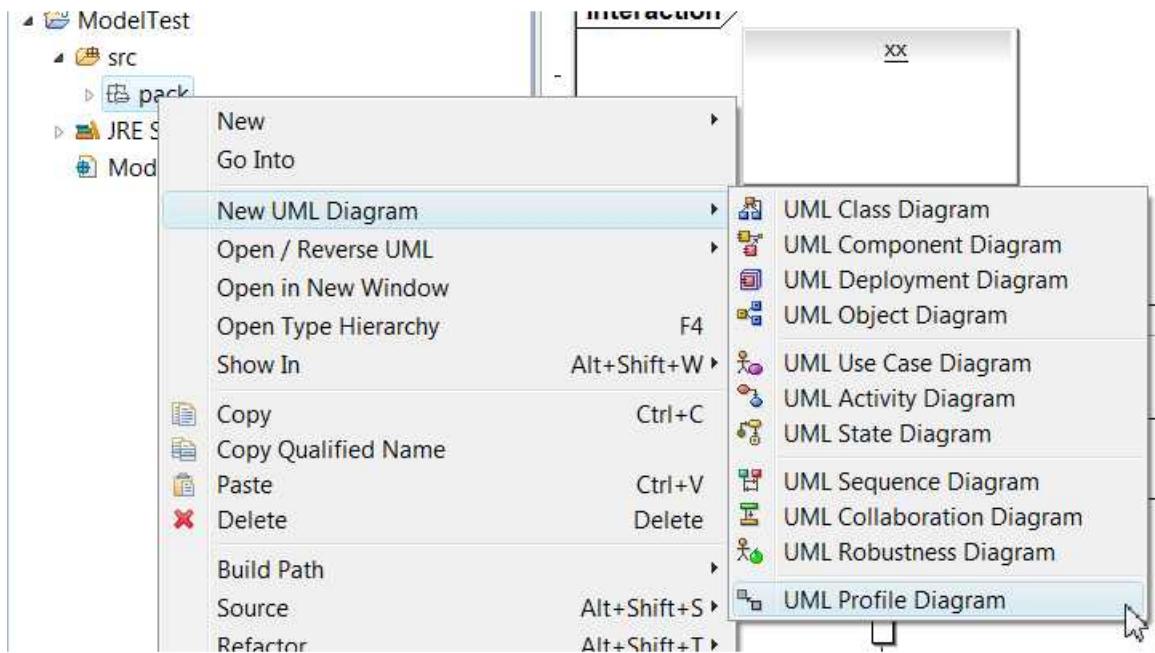


The file deployment diagram extension is *.udd.

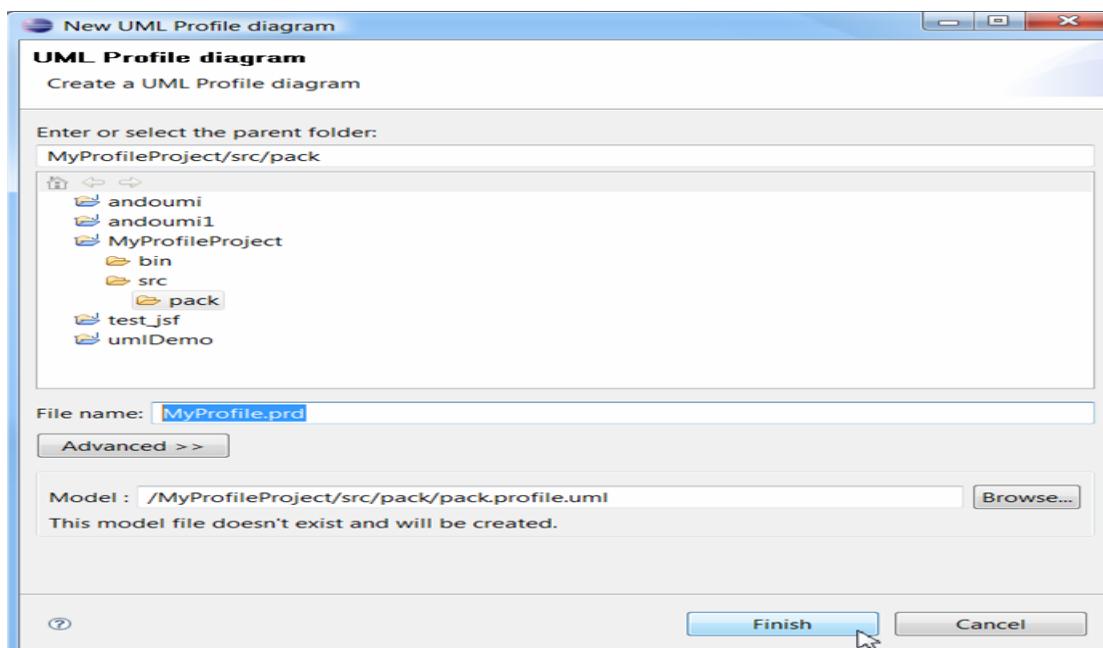
Profile Diagram

Profile Diagram Example

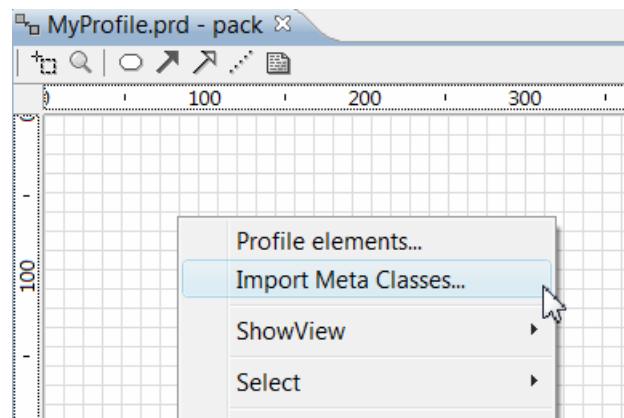
The following diagram is the extension of the UML 2 metamodel by adding a new UML 2 profile in the UML 2 metamodel. The profile will then be added to a project Create a new project named MyProfileProject and add a src folder and a package named pack in your Package explorer. Click on the package **pack** > **New UML Diagram** > **UML Profile Diagram**



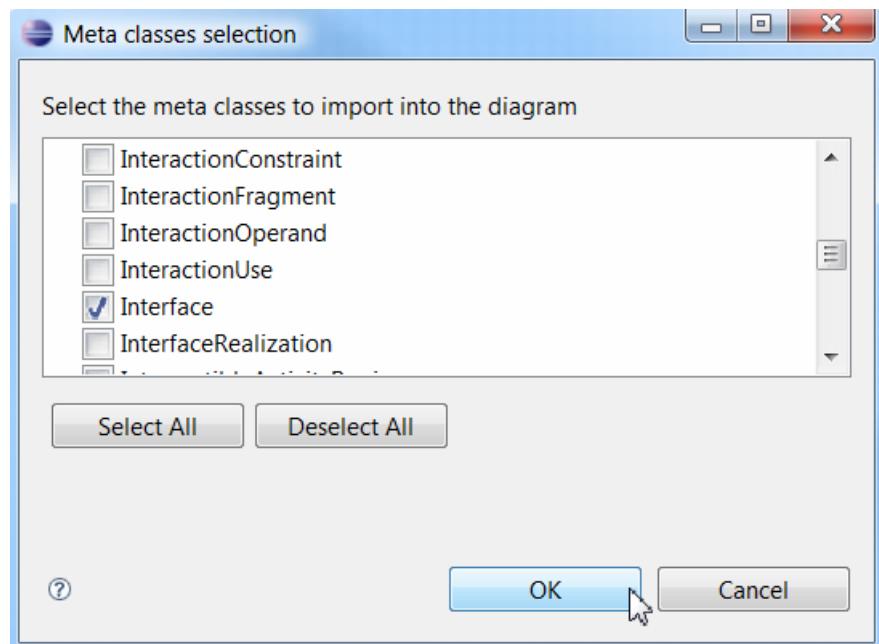
Change the Profile diagram name, and type MyProfile.prd



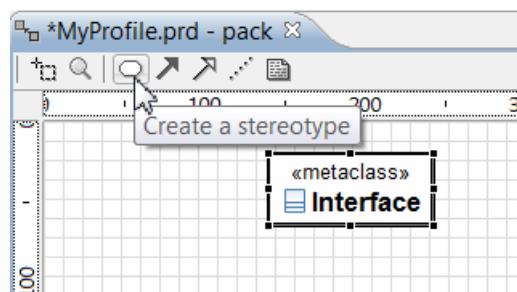
Open the Profile diagram contextual menu > **Import Meta Classes**



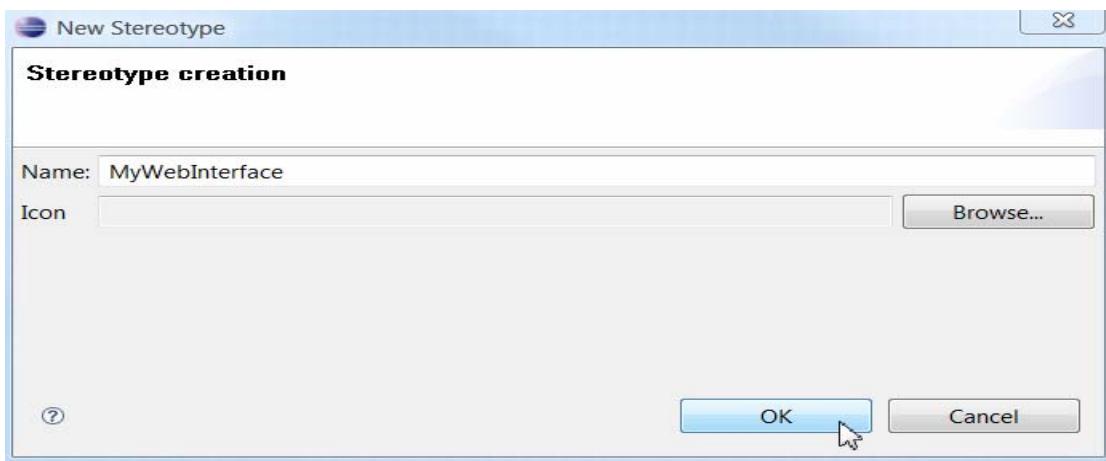
Select the UML 2 meta classe to be extended.



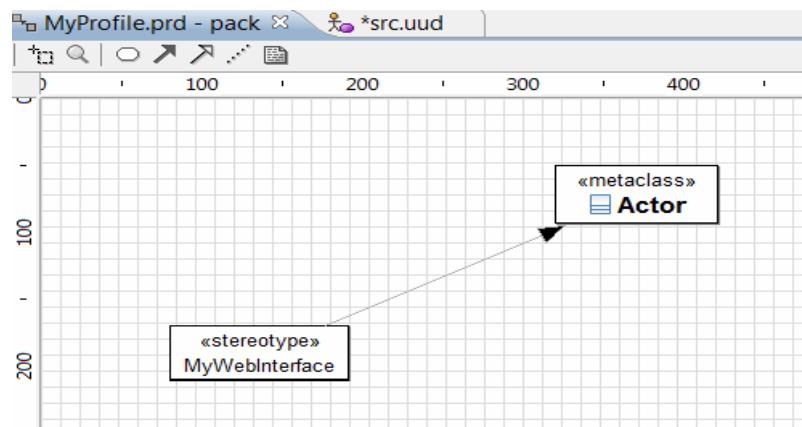
Drag and drop one stereotype from the toolbar



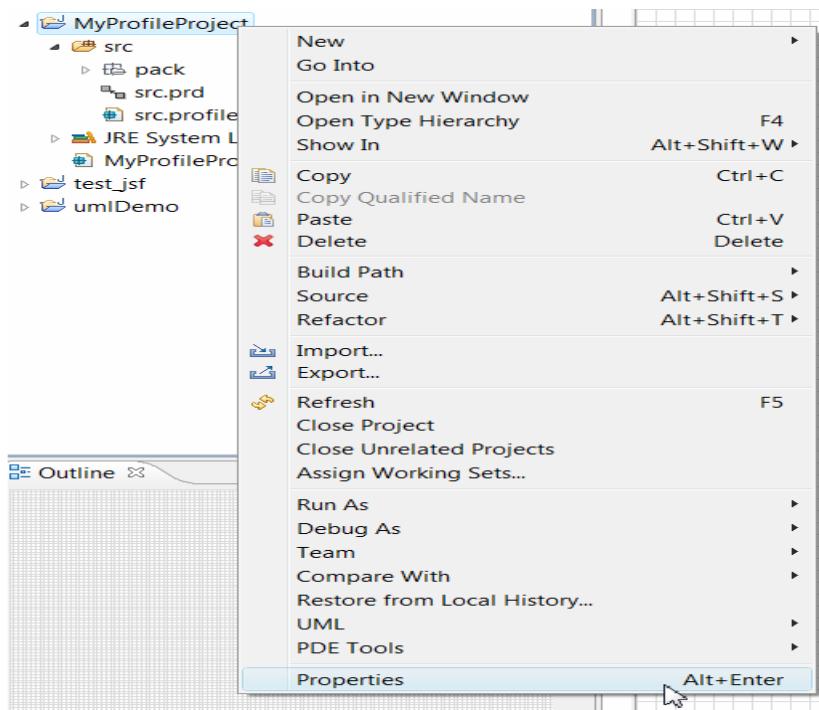
Give a name to your new stereotype and extend it to the metaclass.



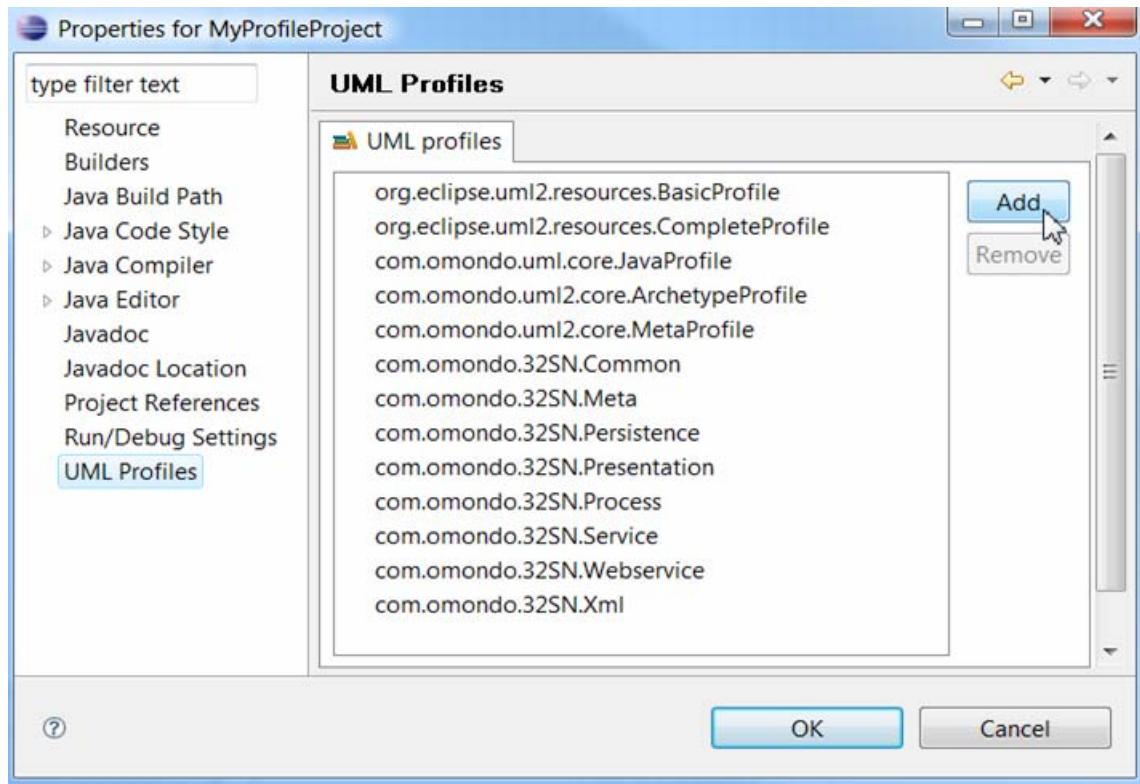
Note that you can add your own icon using the browse menu.



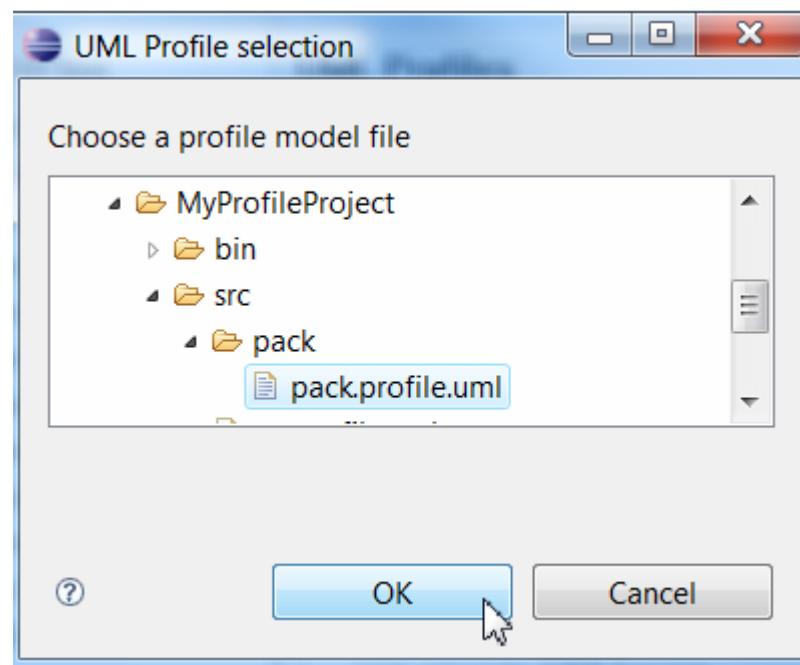
Save your profile diagram and then add the profile property on a project.



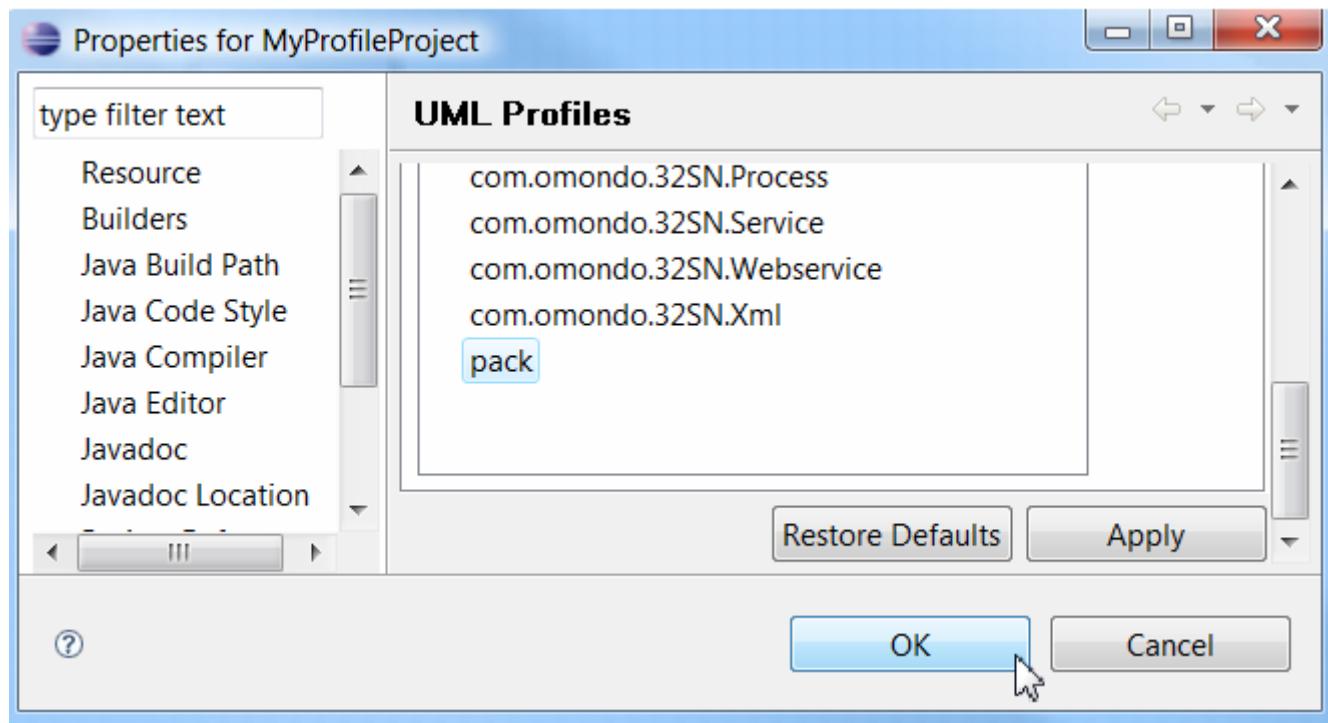
Select UML Profiles and click on the Add button



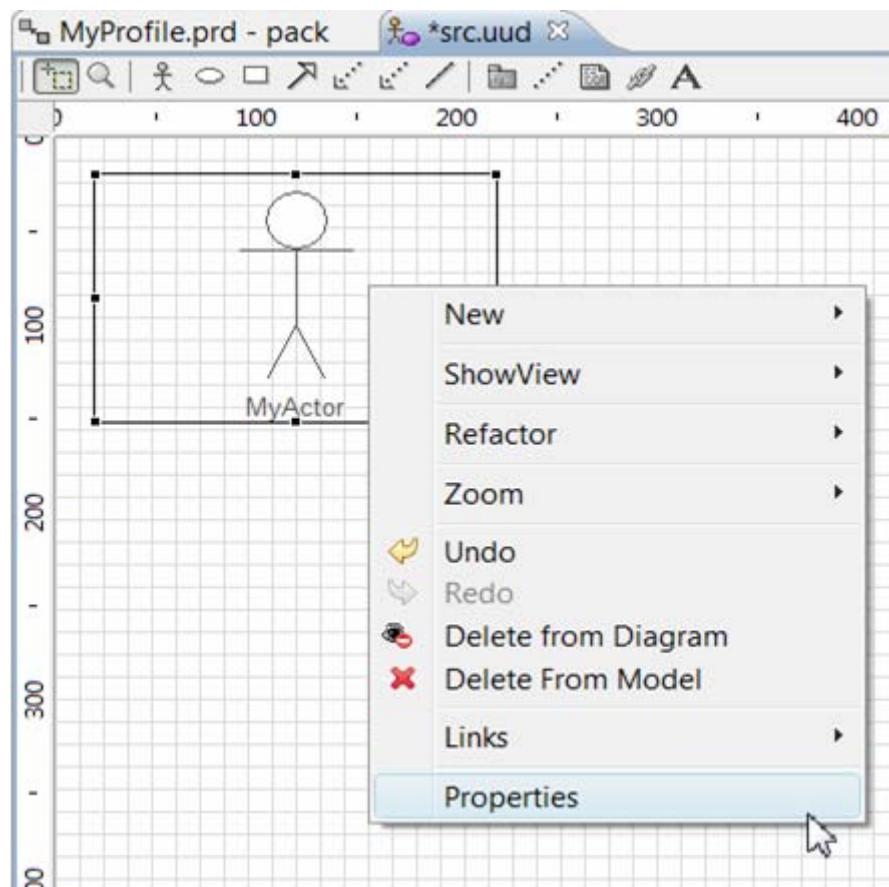
Select your profile diagram project model extension



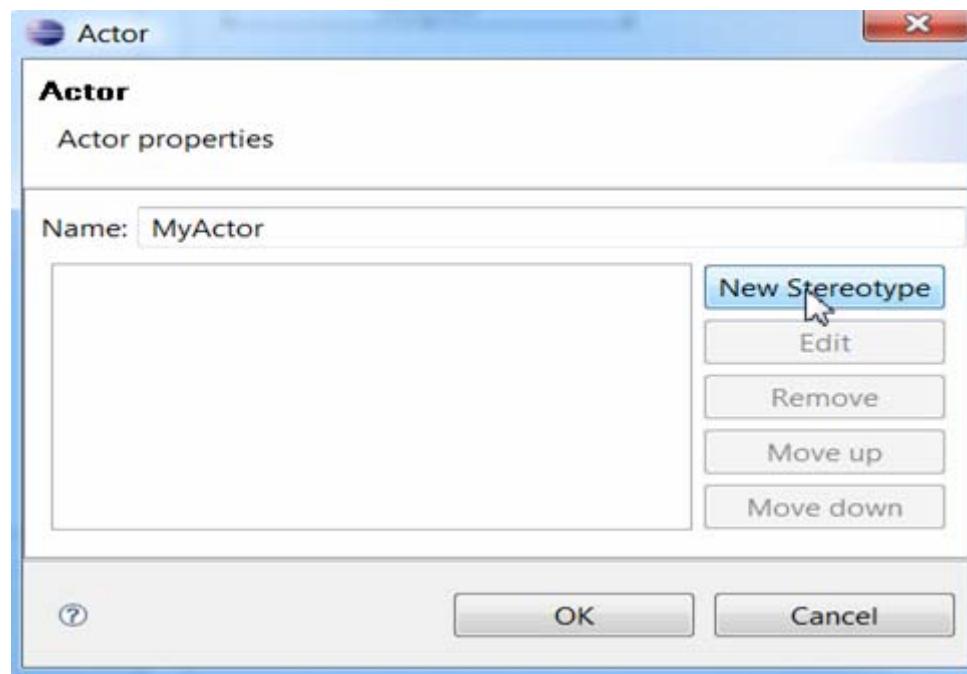
Your profile will appear in your Property wizard in the UML tab.



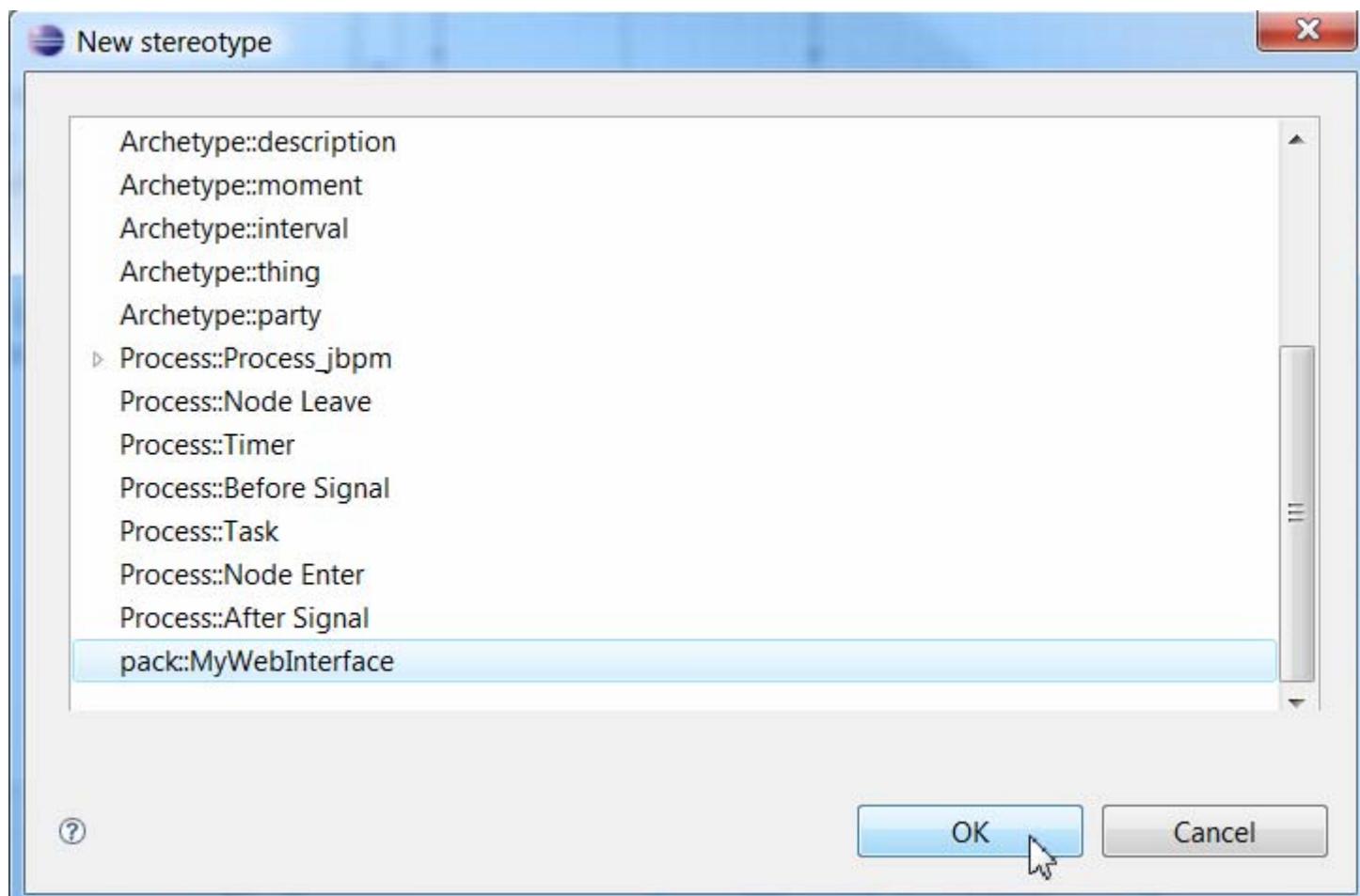
Create now a usecase diagram and an actor



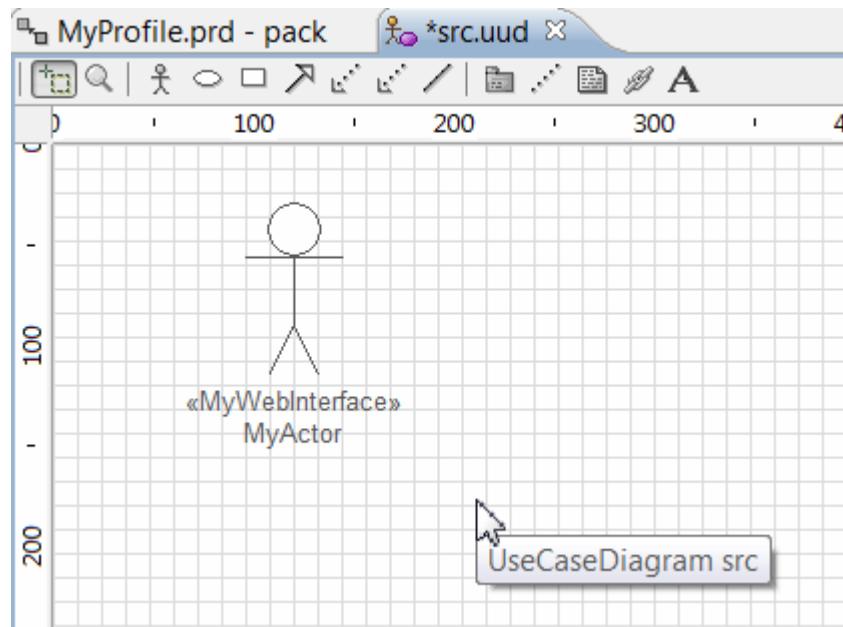
Select New Stereotype button



Browse the profile and find your new stereotype that you have just created in the profile diagram.



The new stereotype has been created in the diagram and in the UML 2 metamodel.



Refactoring

It is important to be able to refactor your Java code & model at any stage of the development cycle.

EclipseUML allows many advanced options when modeling Java or UML classes.

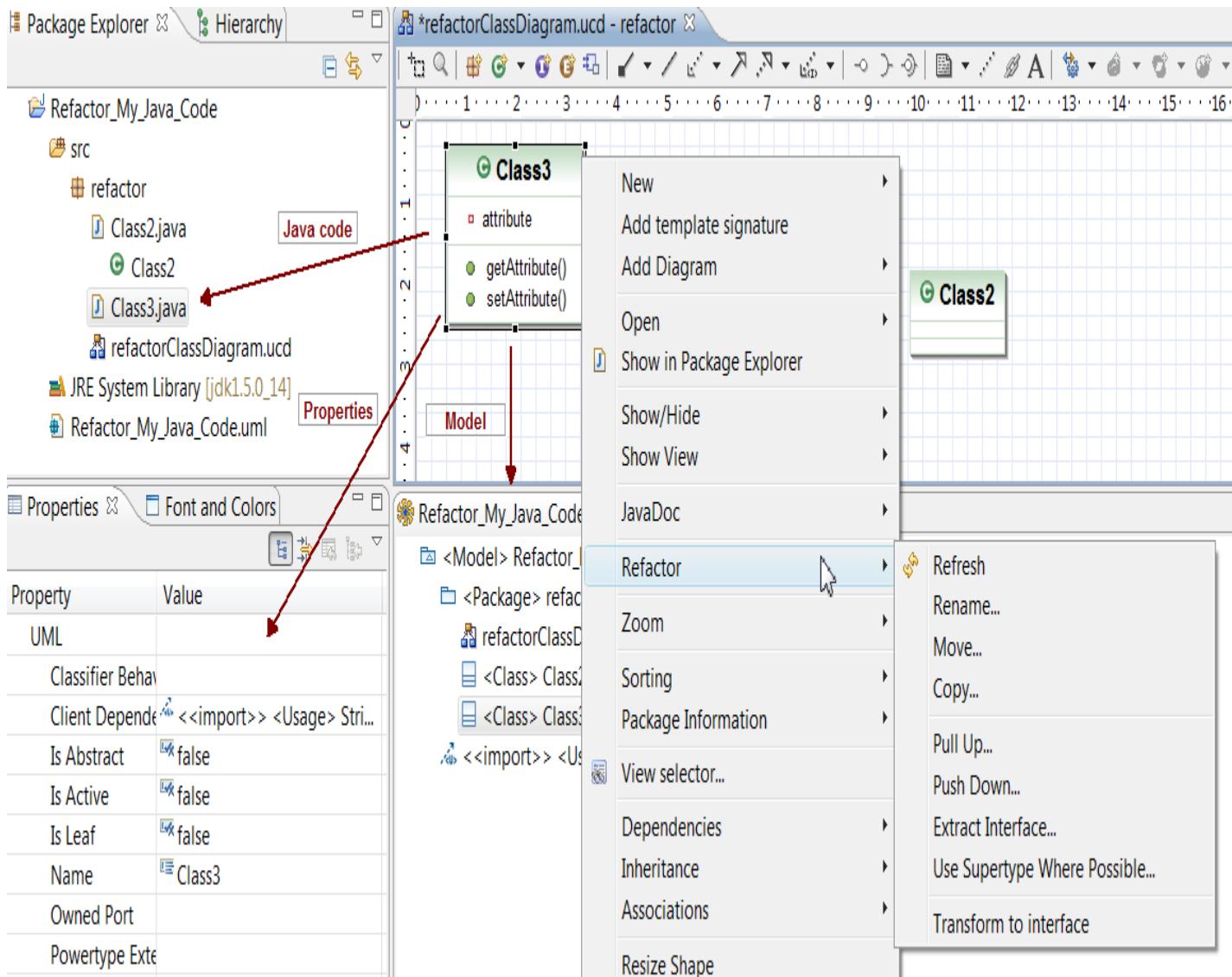
The following options are possible:

1. [Refactoring Java Code using the Class Diagram](#)
2. [Refactor Java Code when EclipseUML is switched off and update the model](#)
3. [Move UML diagram into a new package](#)
4. [Reverse Java code and refactor the model with no code generation](#)
5. [Refactor the UML project](#)

1. Refactor Java code using the Class Diagram

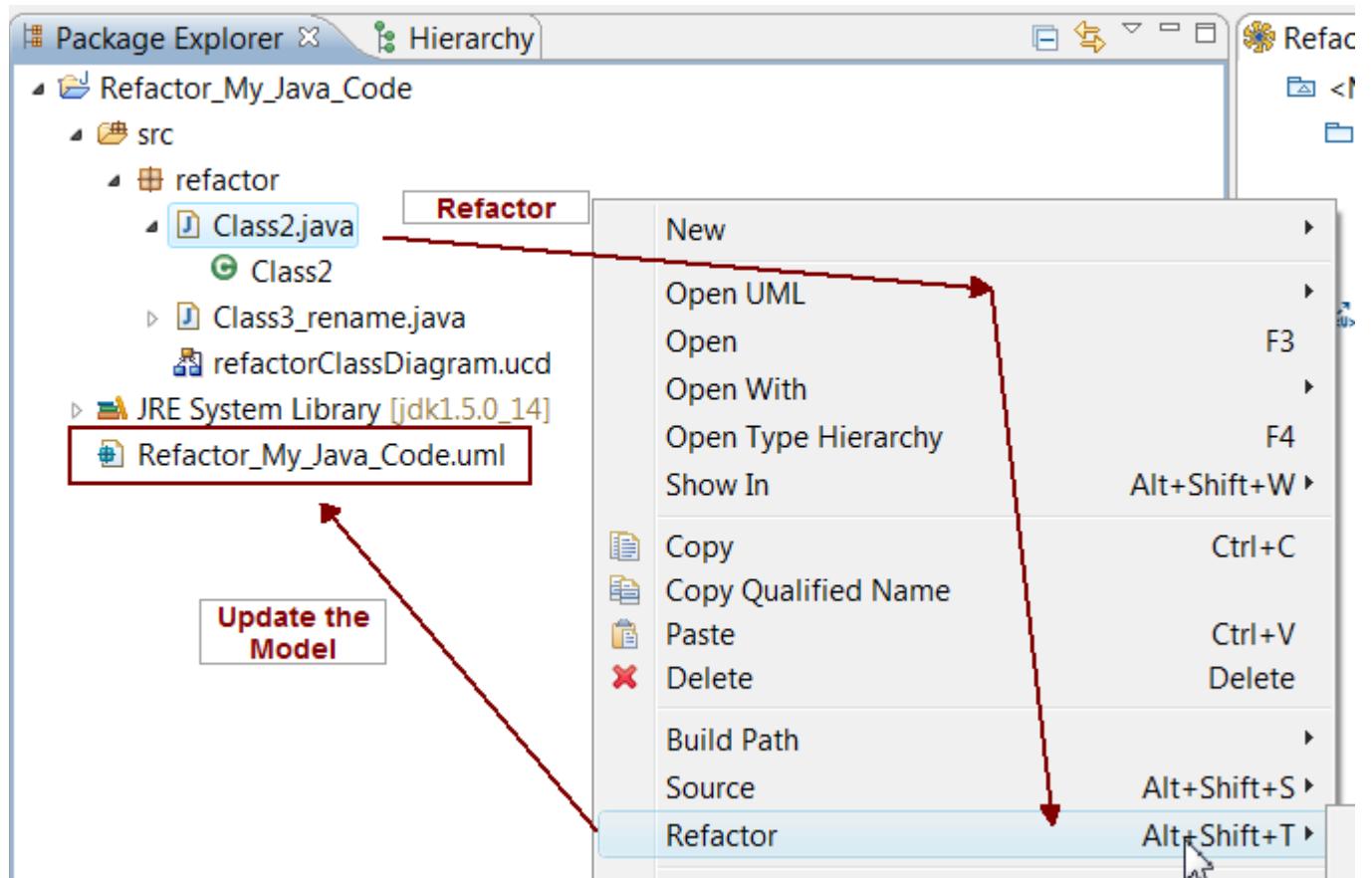
You can refactor your java code using the **Class Diagram** contextual menu > **Refactor**.

You can Rename, Move etc.... this information will immediately be updated in the Java code (e.g. Package Explorer) in the Properties View and in the UML Superstructure of your model.



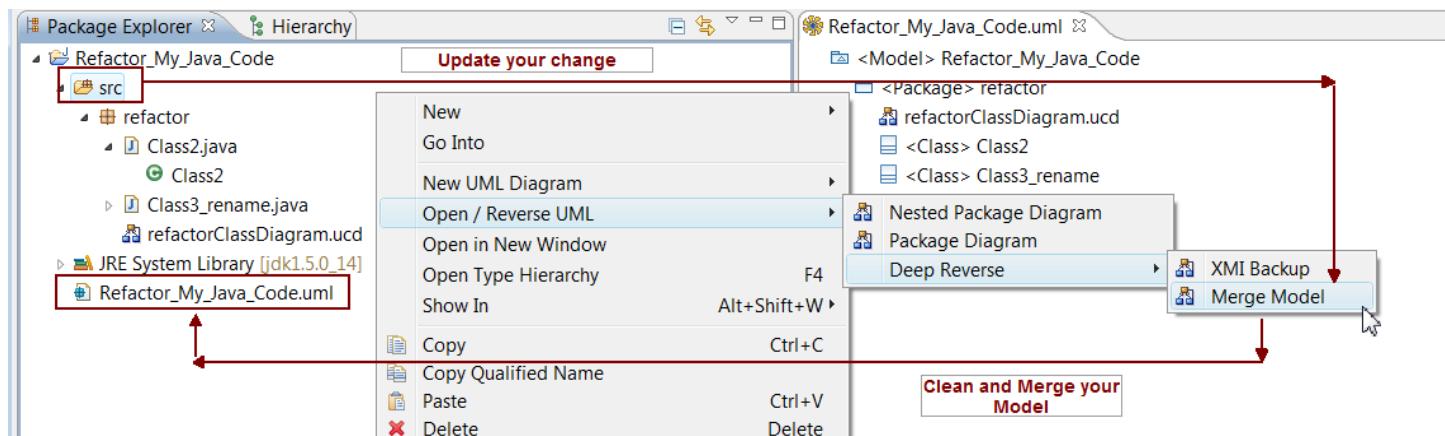
2. Refactor Java Code when EclipseUML is switched off and update the model

You can refactor Java code even if EclipseUML is closed and not loose your code and model synchronization. We recommend to always refactor **using the Package Explorer** because EclipseUML has a listener in this view which will immediately update any change from the Java code to the UML superstructure of your model (*e.g. the model is at the root of the Project in the Package Explorer and have the name of your project with an .uml extension*). Note that the UML Superstructure of the model should always be at the root of your project. If you use CVS/SVN and get a modified code from a non registered EclipseUML user then you need to merge your model in order to update all classifiers which have been added, deleted, changed or moved in your project.



Note that we have disabled the Java Editor listener in order not to slow the speed of manual java code typing. You need therefore to refactor using the Package Explorer and if not please don't forget to use the model merge feature in order to update your model.

In the Package Explorer click on the **src > Open / Reverse UML > Merge Model**

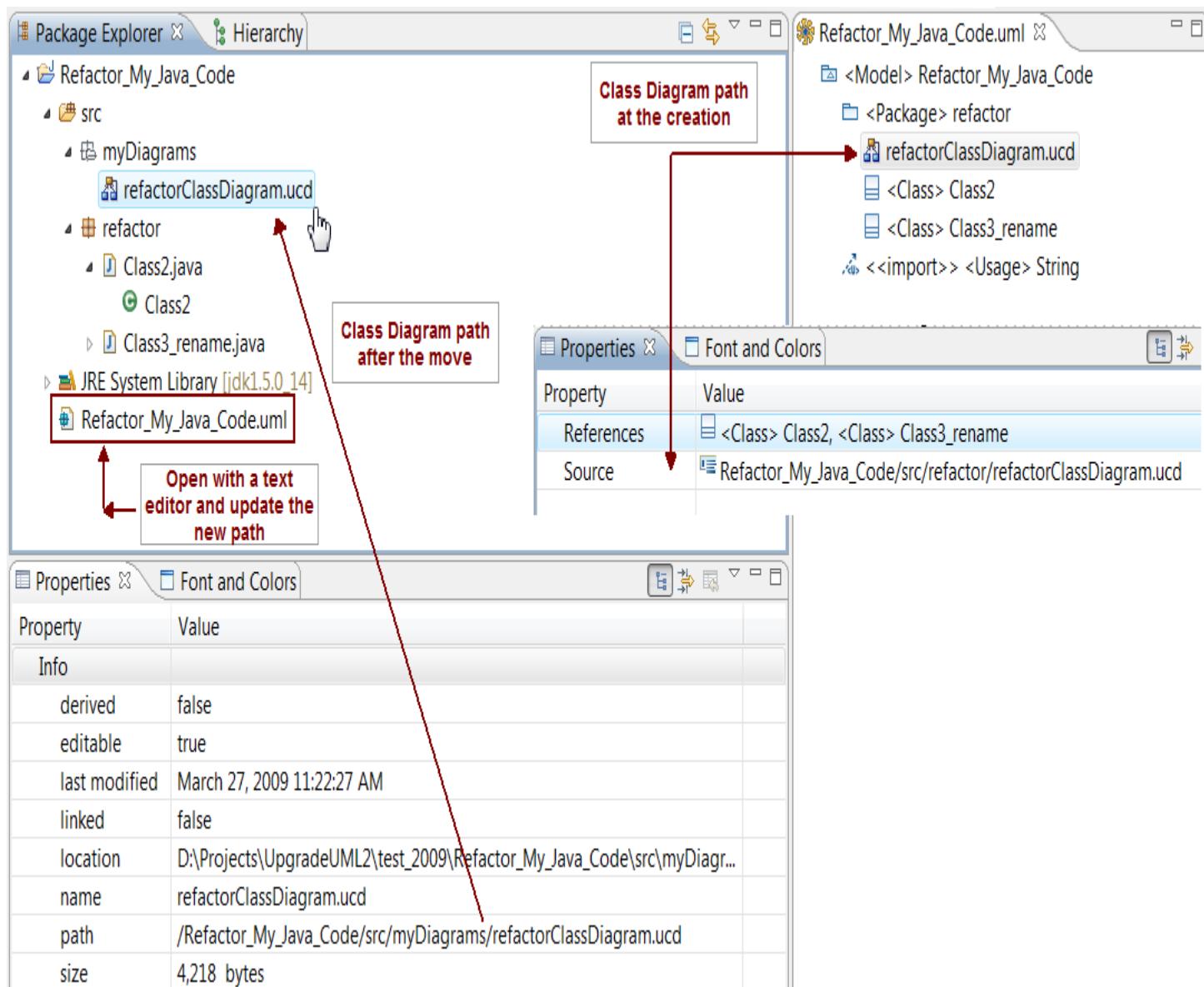


3. Move UML diagrams into a new package

UML Diagrams can only be saved in a java/jee nature project at the root of src or a package. You can create your model structure just using the UML superstructure of your model but not save any diagram inside your XMI.

Graphical and model information are always saved into two different files. Model information is saved in the UML Superstructure of your model which is available at the root of your project and graphical UML Editor files are saved on the hard disk inside the project.

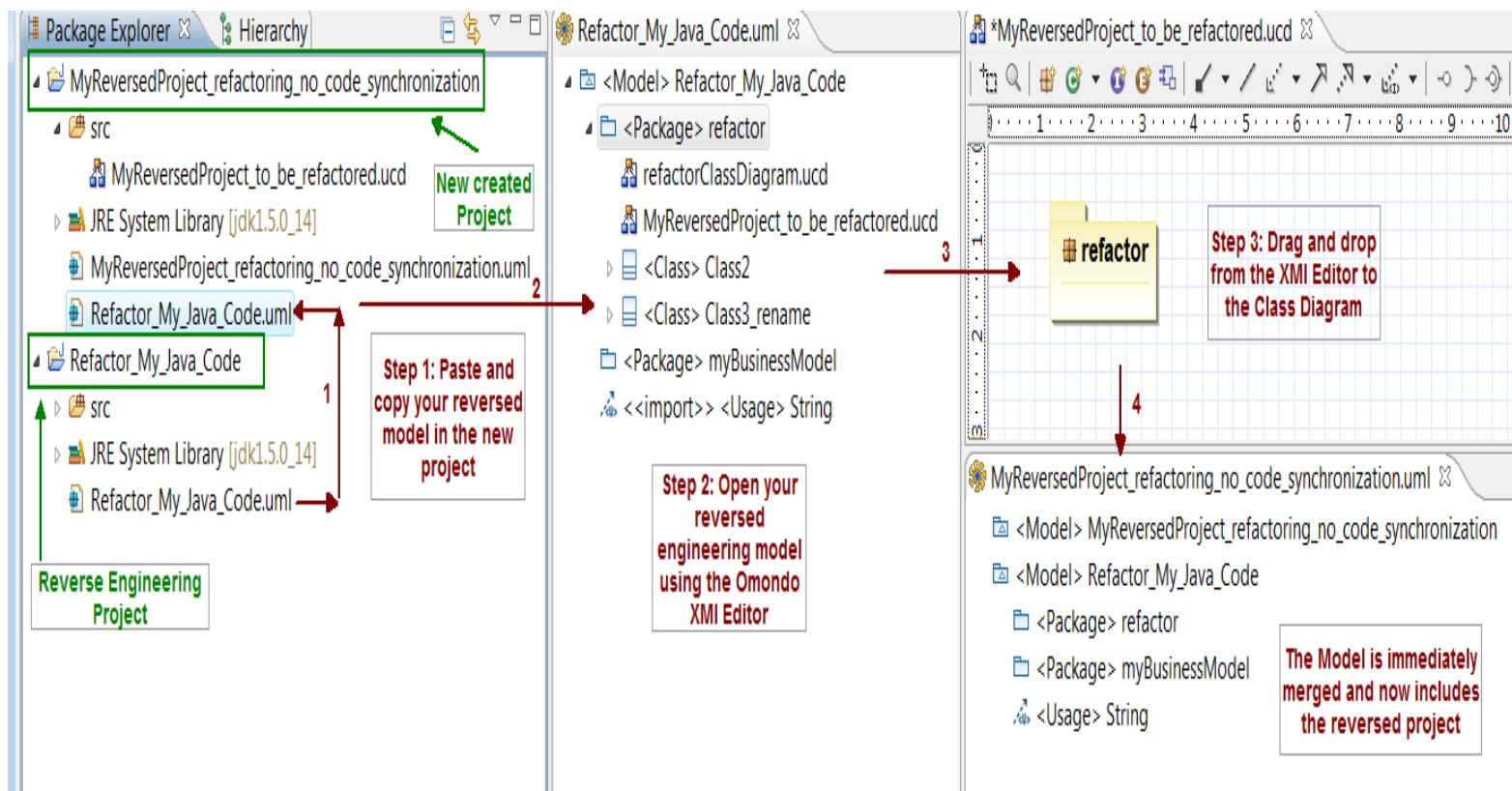
For example if you create a new package inside your Package Explorer named myDiagrams and then move your diagrams using the mouse (*the paste and copy feature is also possible*) inside this package then there will be a path problem to be solved in order to keep traceability. The related .ucd file in the UML Superstructure of your model is only available for traceability purposes and is representing a link to your already saved graphical file. If you move your graphical diagrams after its creation then the path is incorrect. It means that you need to manually update paths using for example the find and replace feature of a text or xml editor. *Note that myDiagrams package is still not in the model because it has been just created and not merged with the model. To merge it with the model you need either drag and drop the new created package inside a diagram or use the Merge Model feature.*



4. Reverse Java code and refactor the project with no code generation

If you reverse a Java project then your model and your UML Superstructure model remain synchronized. You can not therefore refactor your reversed project without refactoring your Java Code. The work around to be able to refactor your project (e.g. *work at model abstract level*) and not start the implementation Java code is to:

1. **Paste and copy your project.uml** UML Superstructure model into another project (*create a new project having a java, jpa, webservices nature and use it for modeling purposes*)
2. Open your reversed project.uml model (e.g. *the one which has been copied*) by selecting the **.uml file > Open With > Omondo XMI Editor**
3. **Create a new class diagram in the new project and drag and drop one classifier** inside the newly created diagram
4. Wait for the **automatic merge to be completed** (e.g. *the merge of your model coming from your reversed java project with this new UML Superstructure Model project*).



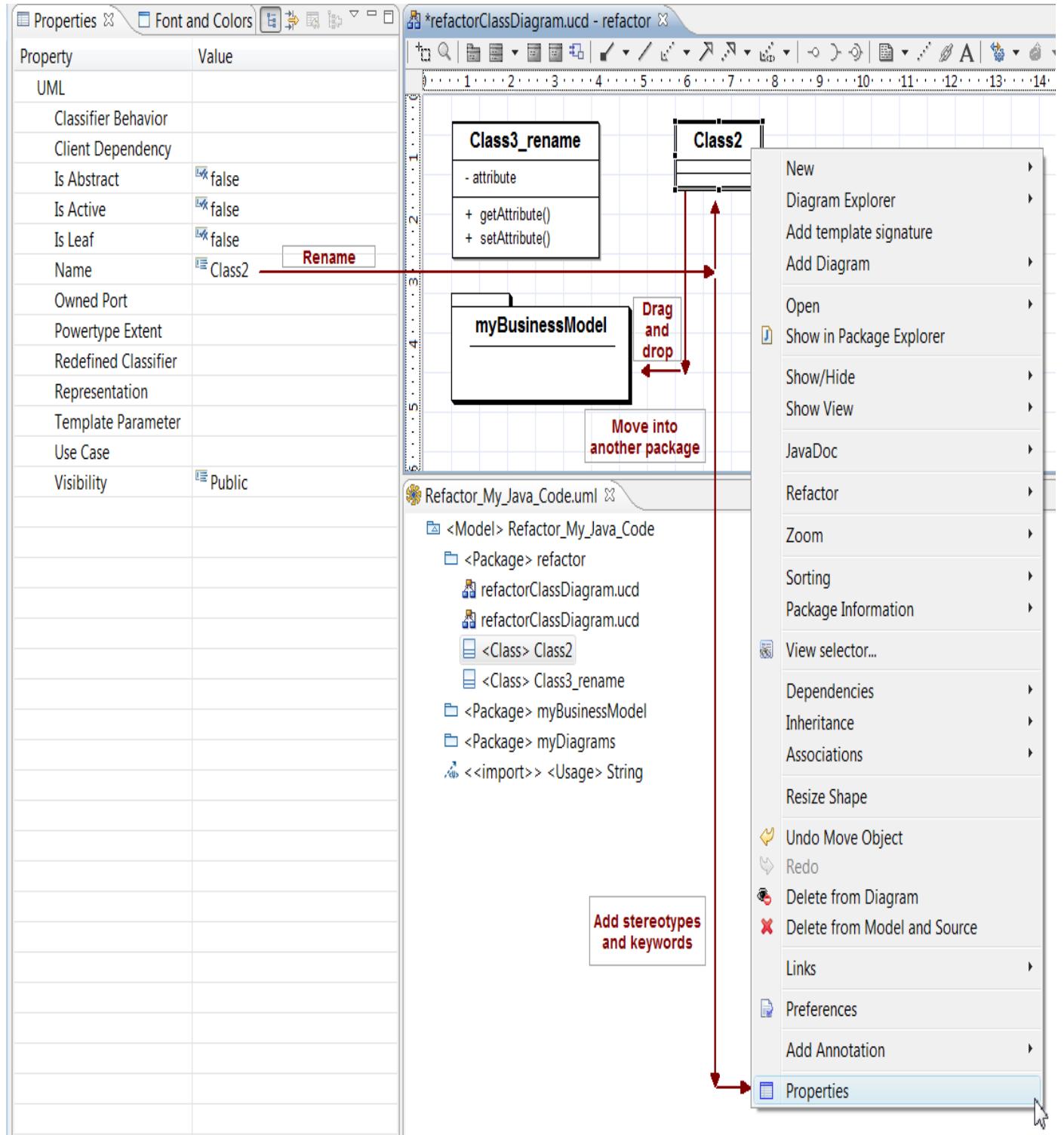
Please note that you can drag and drop as many java project as you want and consolidate all models inside the UML Superstructure of the newly created model. This model would therefore be a kind of "Superstructure multiple models" composed by small models which represents one to many java projects. **This is an important feature when you need to model a large project which is composed by more than one project.**

Once the merge has been completed you can erase the other model because all the model information has been already extracted to the UML Superstructure of the newly created project model.

5. Refactor UML project elements

You can only refactor UML elements (e.g. *not related to java*) with the Properties View or the contextual menu. **Classifiers related to Java should only be refactored using the Java refactoring feature.** The **Properties View allows you to rename** UML classes and change properties as well as the contextual menu. The Properties View is not an Omondo view and could

be used with any other plugin sharing the UML Superstructure model such as the UML Model Editor. You can move classes inside a package by drag and drop inside the UML Editor. Do not use any other manipulation because EclipseUML has added all model synchronization cycle inside the diagram.



Note that UML diagrams are for Omondo viewers of the UML Superstructure model. You can therefore refactor your model directly with the UML editor (e.g. no need to use xml editors). We don't recommend to refactor your model using the Eclipse Model Editor (e.g. only use the UML Editor or the Properties View) because this editor don't have the Omondo Java Classifier logic. EclipseUML allows the use of other plugins therefore it is easy to erase, change or had wrong information in the UML Superstructure of the model. If using any trick to manipulate your model

then Omondo will not be responsible for any model information losses. Btw, if you loose your model you [can restore your previous model by selecting the .uml file in the project explorer.](#)

Working with Diagrams

This section presents the general features of EclipseUML. These features are available for all the EclipseUML diagram editors.

The following areas are covered:

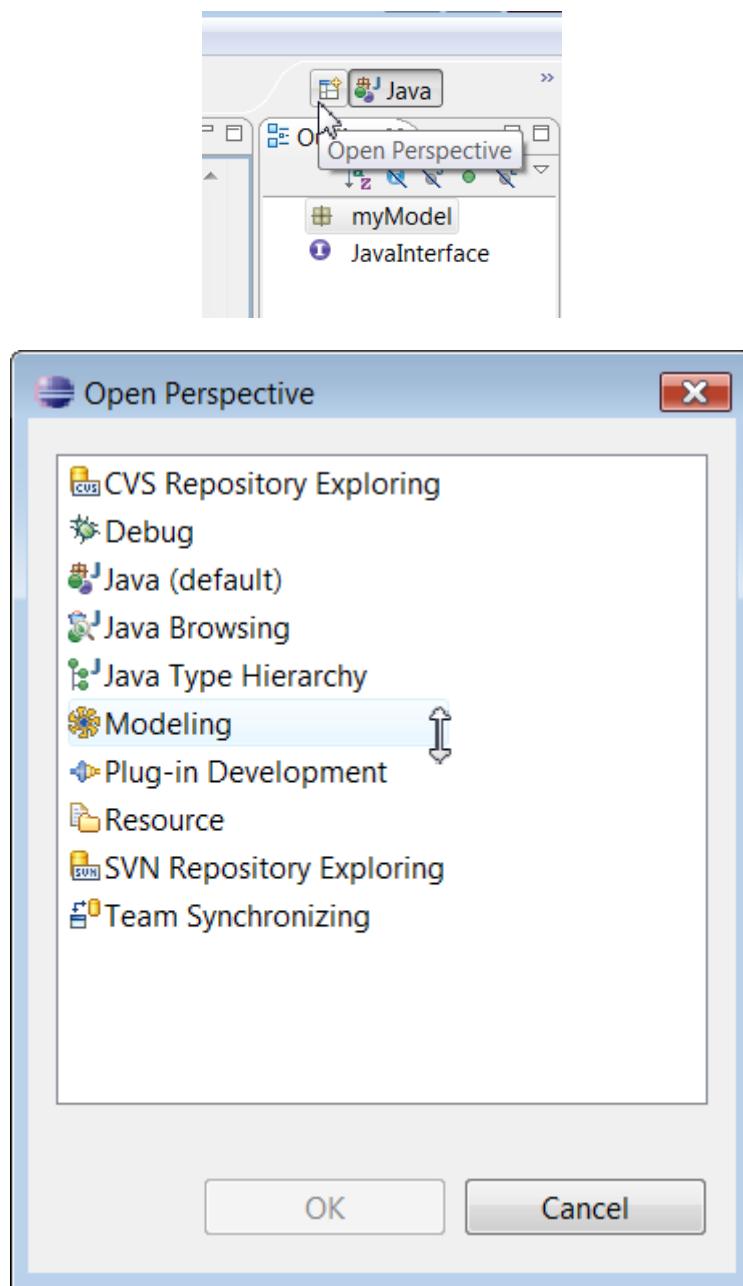
- [Modeling Perspective](#)
- [Rearranging Diagrams](#)
- [Font and colors](#)
- [Class Diagram Presentation](#)
- [Format](#)
- [Show/Hide Compartments](#)
- [Wire](#)
- [Link Thickness](#)
- [Zoom](#)
- [Print](#)
- [Export As Image](#)
- [Customize Perspective](#)
- [Notes](#)
- [Comments](#)
- [Constraints](#)
- [Select](#)
- [Labels](#)
- [Links](#)
- [Key Bindings](#)
- [Property View](#)
- [Documentation View](#)

Modeling Perspective

This section describes how to activate and use the modeling perspective.

1. How to activate the Omondo Perspective

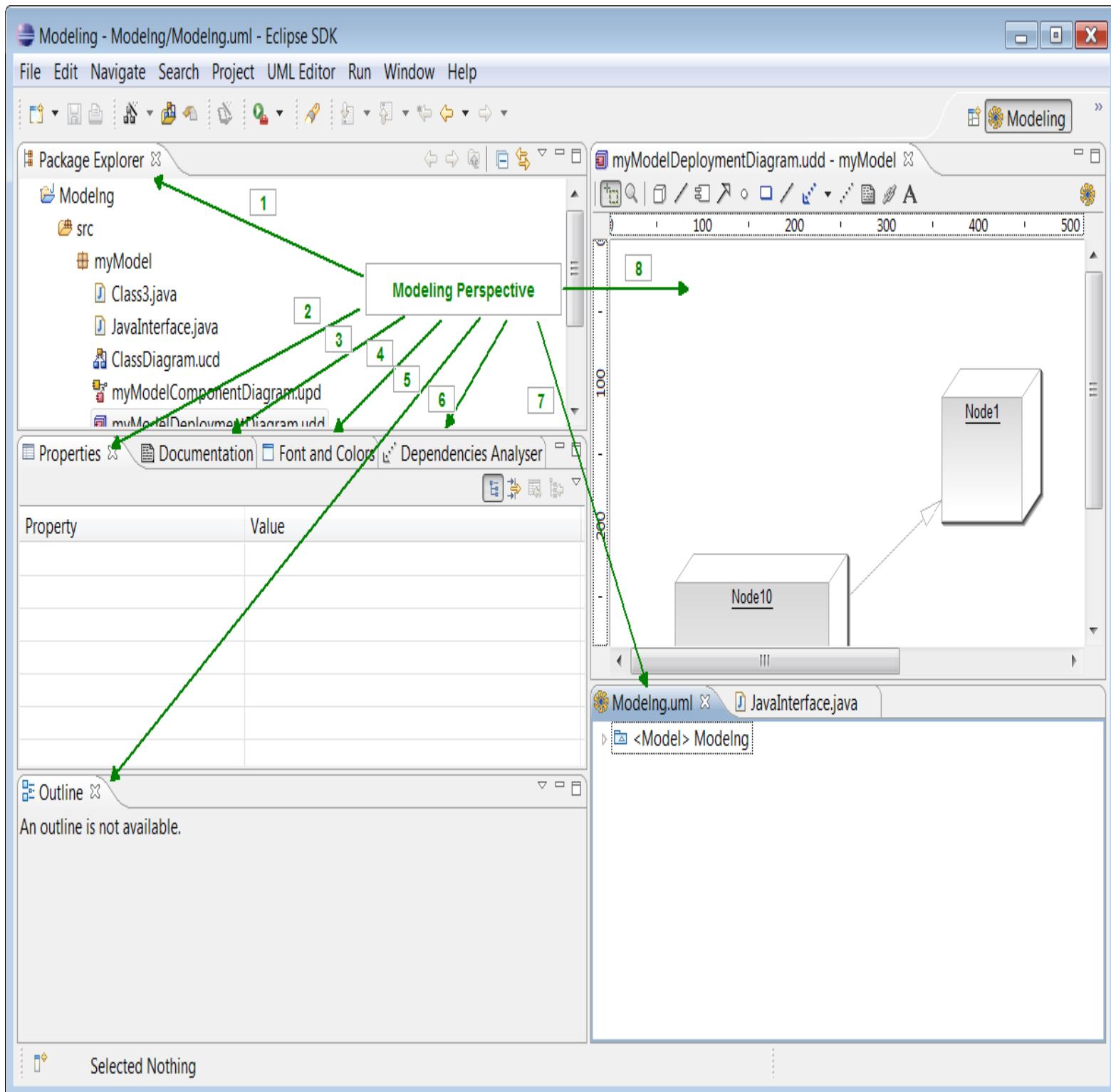
Select the Perspective icon in **the upper right corner of Eclipse > Modeling**.



2. The modeling perspective is composed by:

1. Package explorer
2. Properties View
3. Documentation View
4. Font and Colors View

5. Outline
6. Dependency Analyzer View
7. Model Editor
8. UML Editor



Rearranging Diagrams

In this section you will learn how to rearrange your existing diagrams.

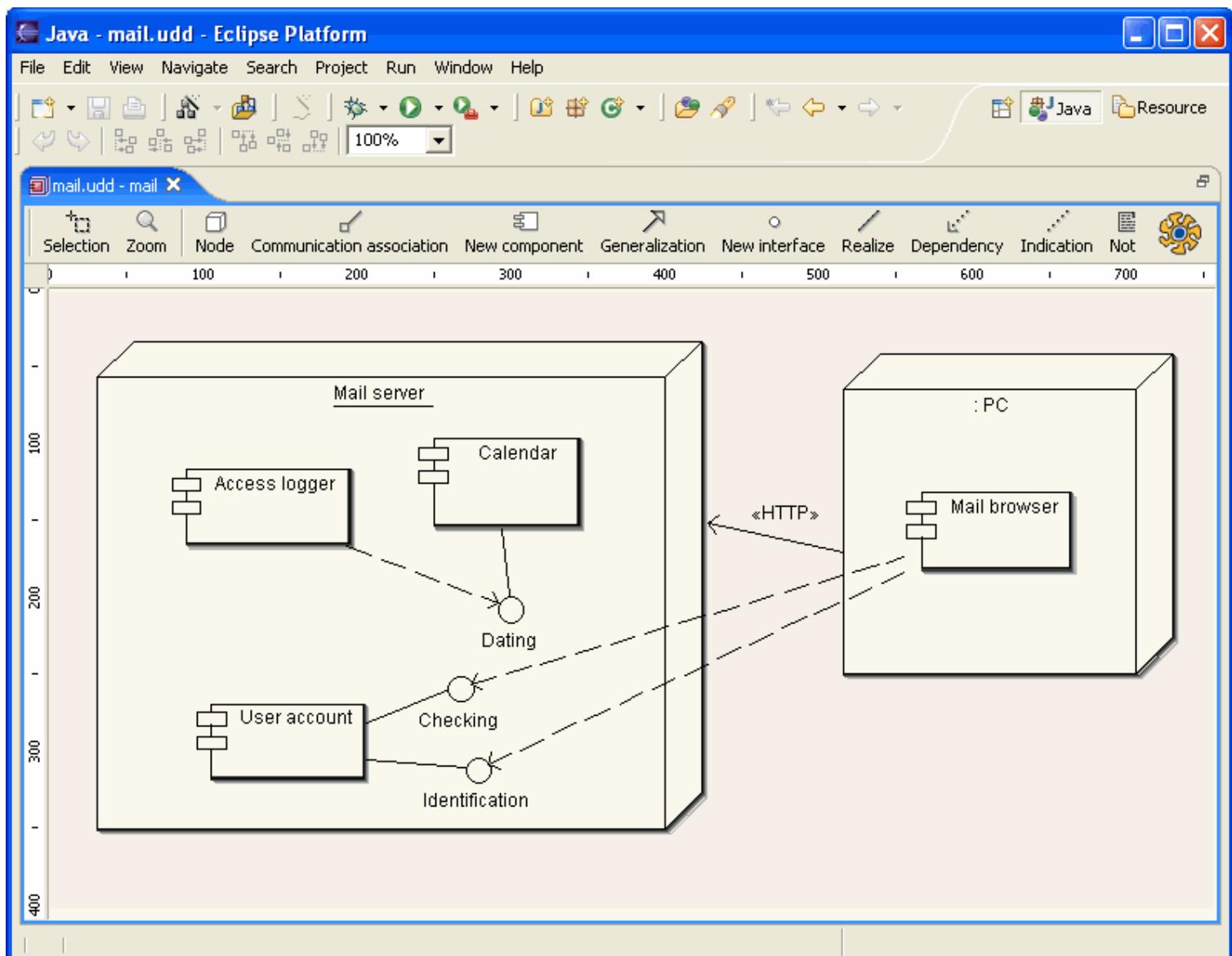
You can work on your diagram to adapt the organization using the following elements:

1. [Full Screen View](#)
2. [Layout](#)
3. [Moving elements](#)
4. [Routers and Anchors](#)
5. [Alignment](#)

1. Full Screen View

It is possible to work on a full screen diagram editor by double-clicking the diagram file tab.

The following example was created by double clicking on the com.udd deployment diagram, then selecting the zoom icon on the toolbar and finally double clicking inside the deployment diagram editor.



2. Layout

EclipseUML allows you to use layout functionalities. We recommend using Layout for a class diagram reverse and then manually reorganizing your diagram.

The layout will reorganize your diagram using mathematical algorithm.

To open the layout function, open the diagram editor's pop up menu and select layout.

3. Moving elements

Advanced multiple selections are proposed, as well as group/ungroup and key shortcuts for more precise functions.

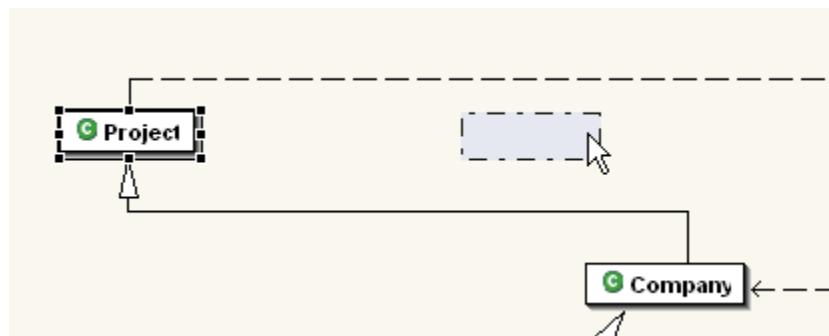
Selection of an element:

EclipseUML allows you to move every element of the diagram using the mouse or the keyboard.

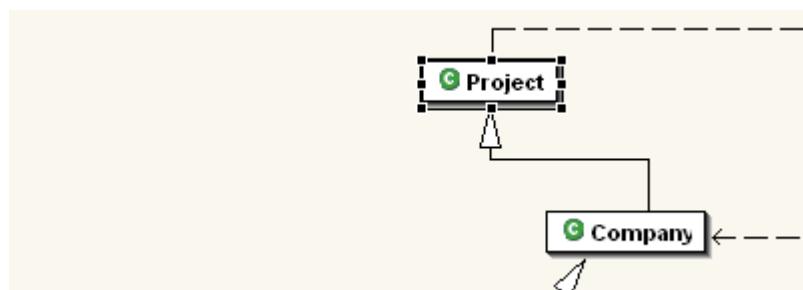
Moving is possible with:

1. Mouse: Left click and keep your finger on the button
2. Keyboard: Ctrl+Alt+arrows

The following example shows how to move the Project class using the mouse:



When the mouse button is released, the Project class has moved.

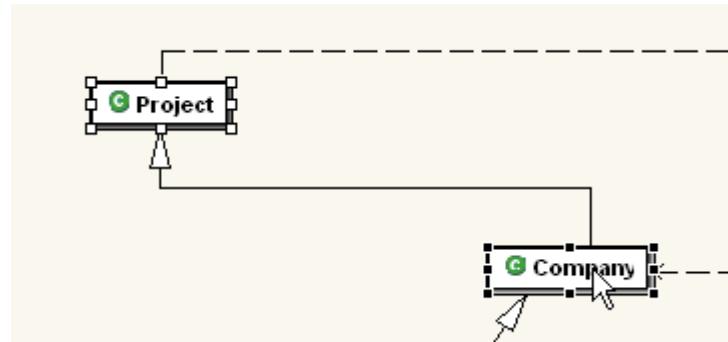


Selection of a group of elements:

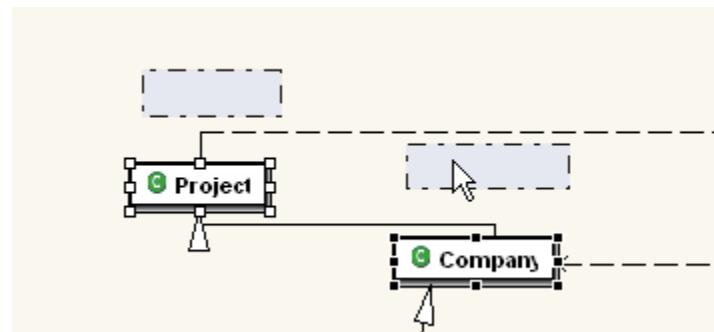
Moving is possible with:

1. Mouse: Shift+left click on each element then move using the mouse.
2. Keyboard: shift+arrows then move using the arrows (if you press Ctrl+arrows, the element will not be selected so use shift+arrows to select the element).
3. Click on selection in the tool bar and crop elements.

The following example shows how to move Project and Company classes using the mouse:



When the mouse button is released, Project and Company classes have moved.



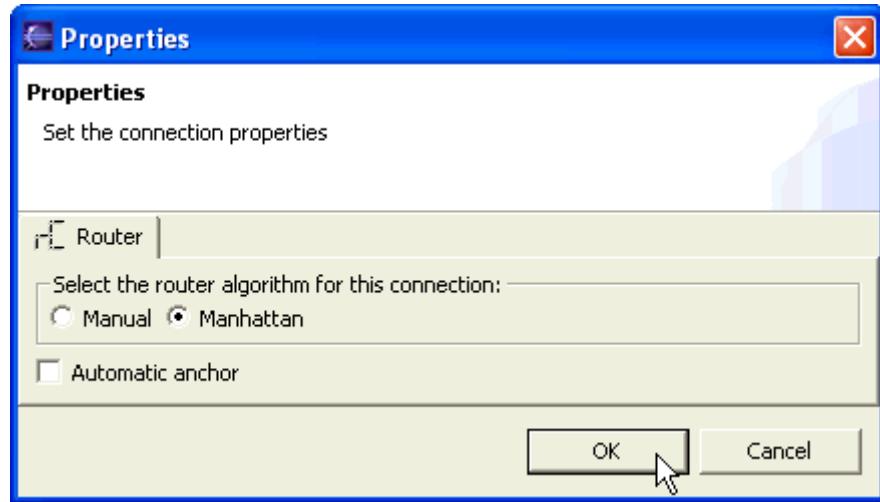
4. Routers and anchors

You can customize every link inside your diagram editor.

Right Click on a link between two elements >open the pop up menu>Properties

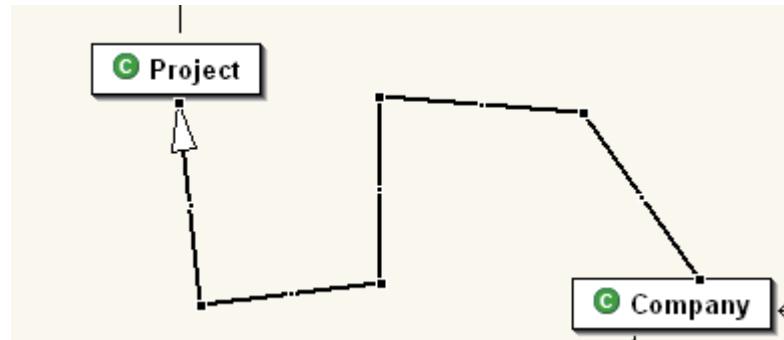


You can select the router algorithm and Anchor for each link



Router:

Manual allows moving the extremity of each link.

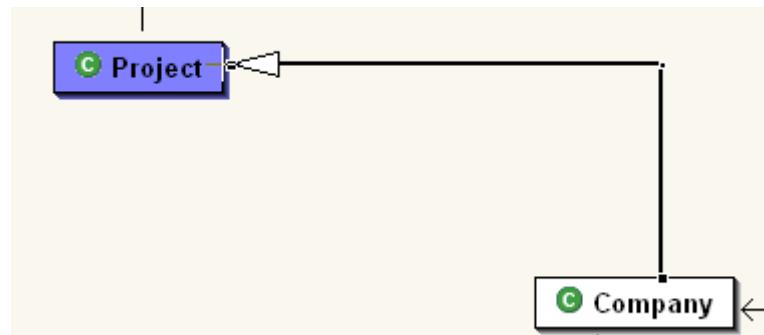


Manhattan uses automatic functions to design the link.

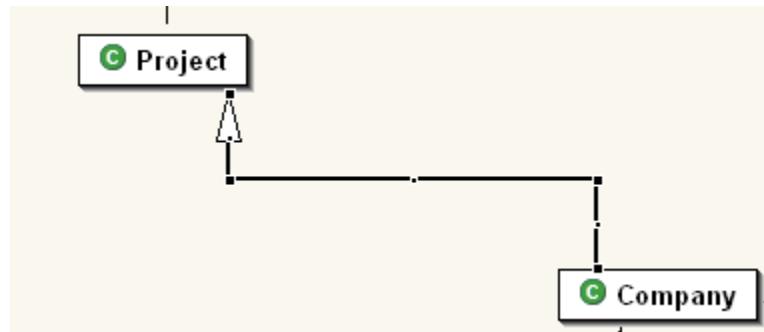


Anchor:

Unselecting Automatic anchor allows fixing the extremity anywhere. Move the anchor inside the element which will become blue when the anchor is properly fixed.



Automatic anchor uses algorithm to fix the anchor automatically (the anchor cannot be moved)



5. Alignment

EclipseUML allows you to select a group of elements and to arrange the alignment.

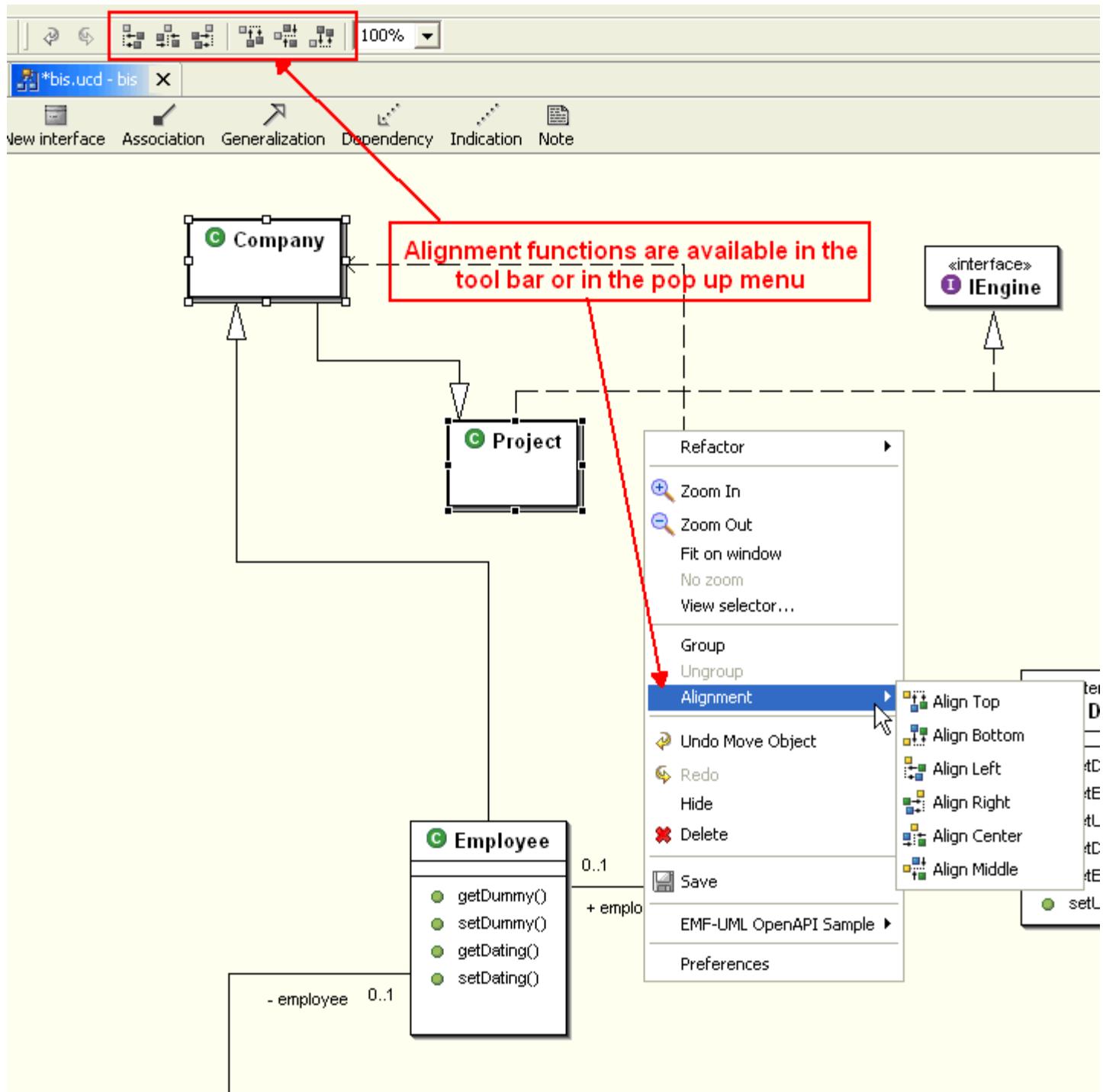
Six alignments are possible:

1. Align Top
2. Align Bottom
3. Align Left
4. Align Right
5. Align Center
6. Align Middle

The following example shows where you can select the alignment function:

Select the alignment function from:

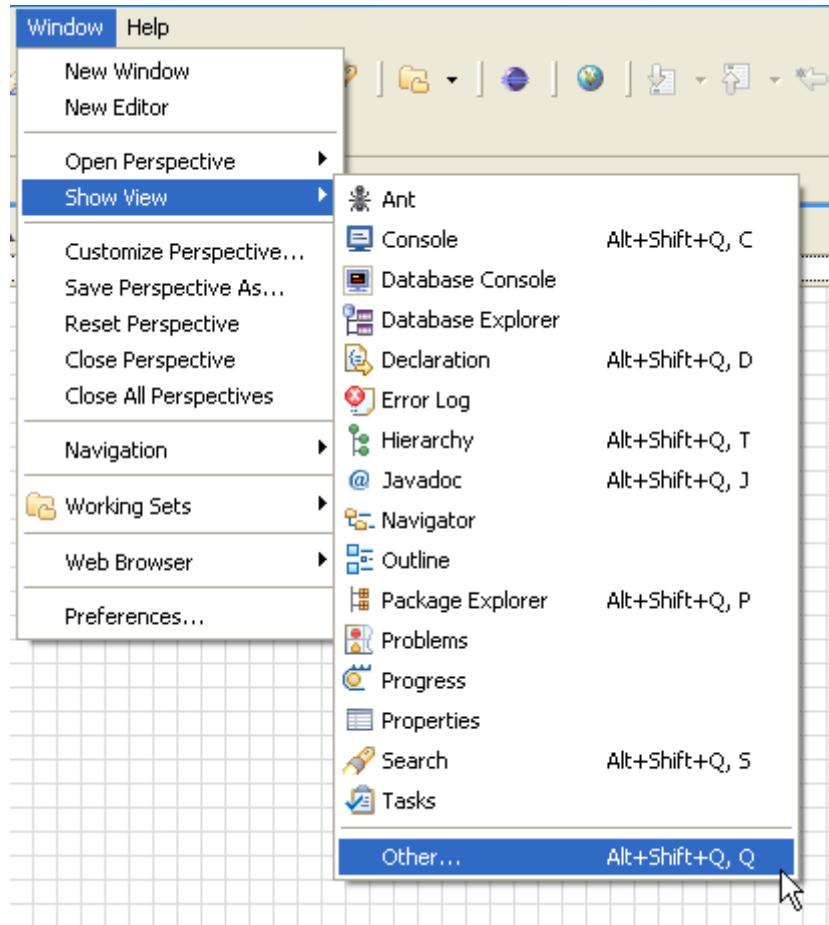
1. The workbench window's toolbar
2. The diagram editor's pop up menu



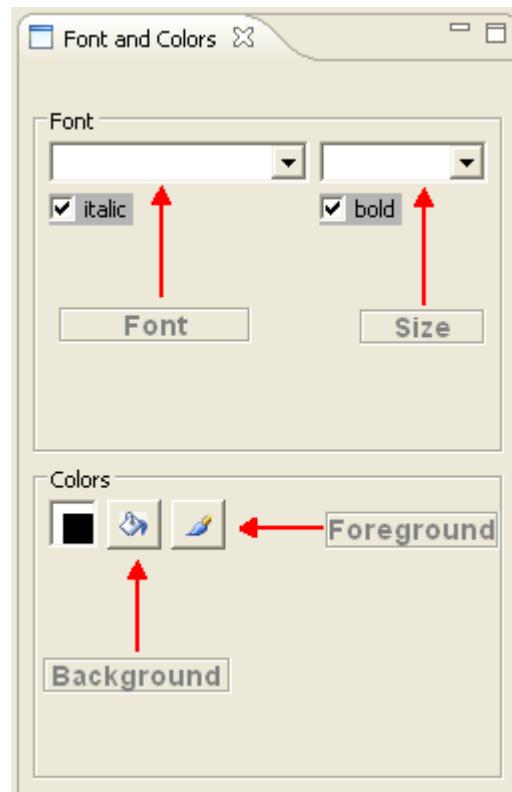
Font and Colors

This section describes how to change Font and Colors in any UML diagram.

The fonts and the colors can be changed by selecting: Window > Show View > Other > EclipseUML > Font and Colors



The new Font and Colors View will be available:



Class Diagram presentation

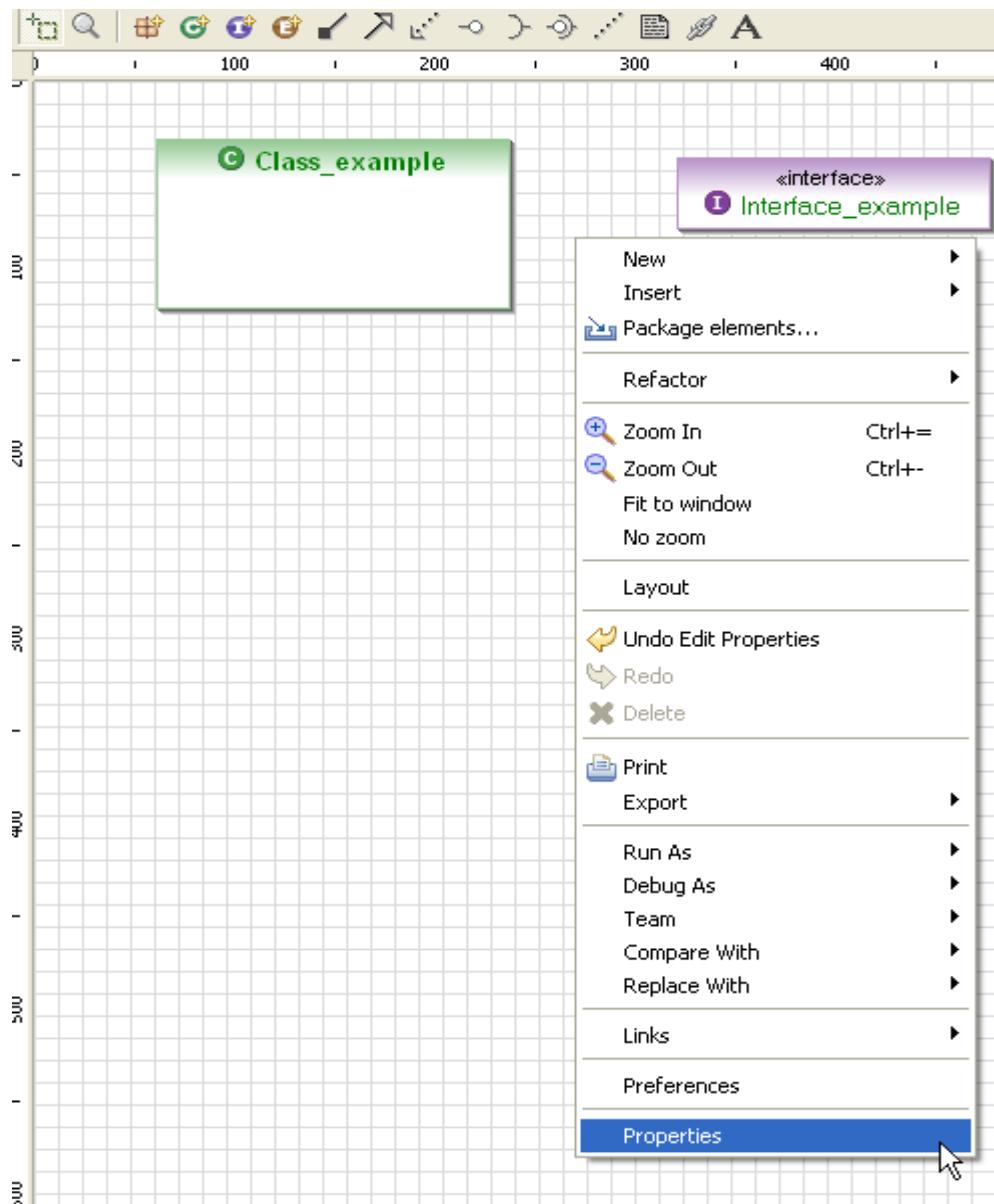
You can customize your class diagram perspective using the following elements:

1. [Diagram Properties](#)
2. [Class Diagram Font and Colors](#)

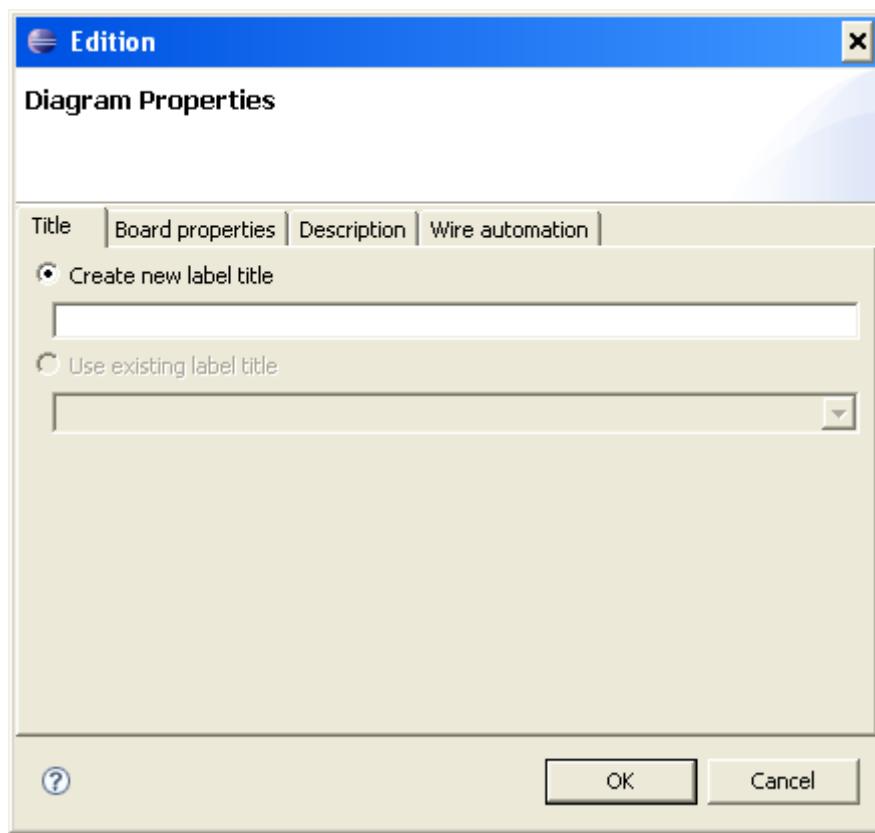
1. Diagram properties

Select the class diagram background, using the left button of the mouse.

Open the popup menu and select **Properties**

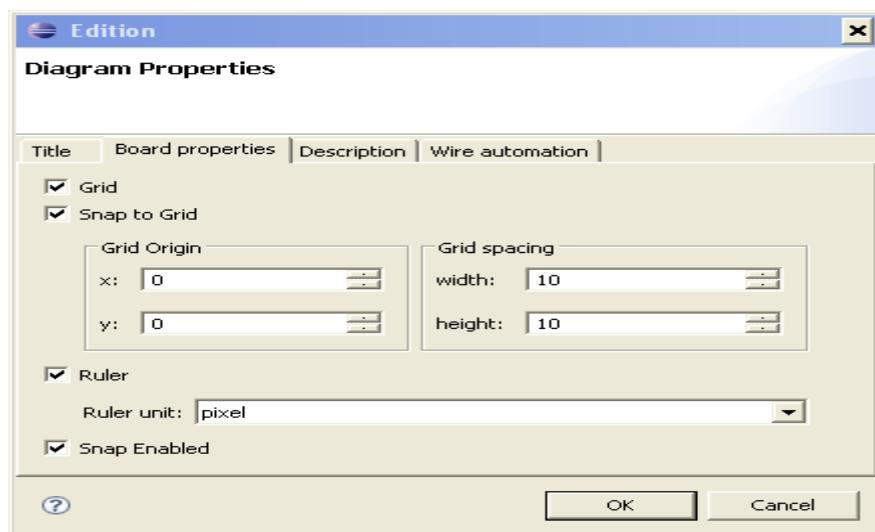


The following Diagram Properties wizard will be shown:
You can add a new label inside your diagram

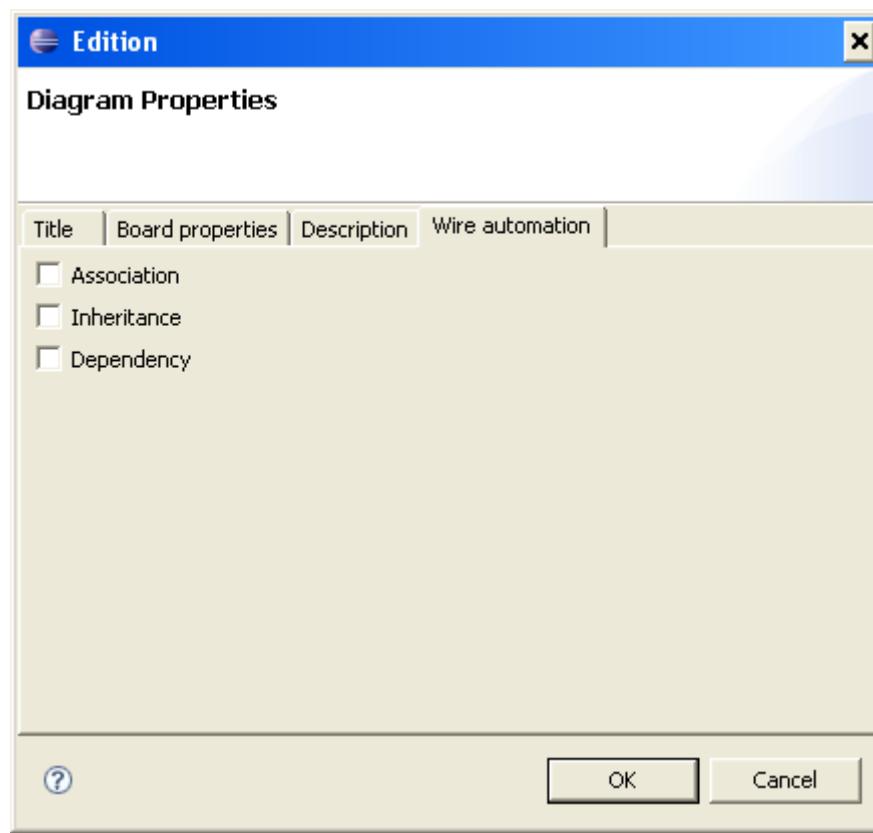


You can customize the class diagram presentation by using the following options:

- Grid (selecting Grid will add a grid in your diagram)
- Snap to grid (selecting snap to grid will move elements using the grid size)
- Ruler (selecting ruler will add a ruler in your diagram)
- Snap Enabled (selecting snap enabled will add a 2 blue lines up/down and right/left, when moving elements in the diagram)

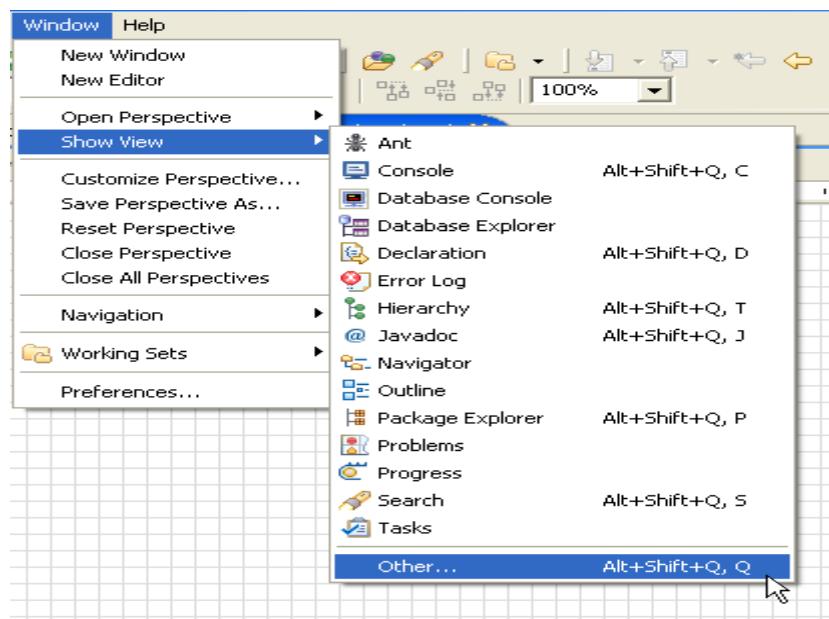


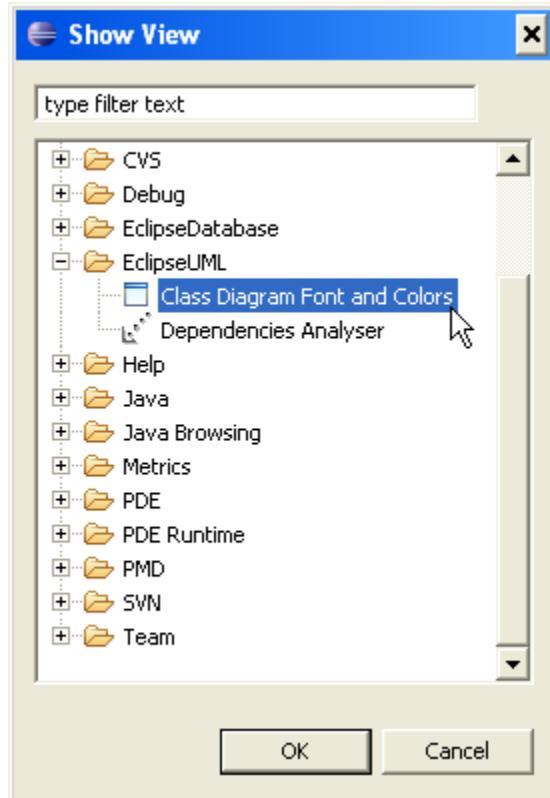
You can add additional diagram information related to your diagram by selecting Description This information is available in the project documentation. You can select the wire automation property for association, Inheritance and dependency



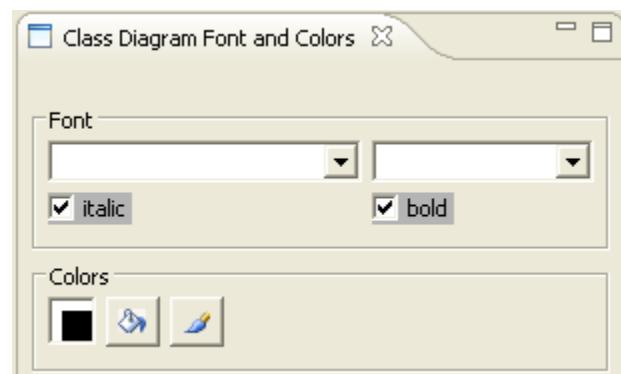
2. Class diagram font and colors

The font and colors of any class diagram element can be customized for UML needs. Select from the menu bar **Window > Show View > Other > EclipseUML > Class Diagram Font and Colors**





The following View will be shown:



If you want to select more than one element in the class diagram, then use the mouse and Ctrl.

Format

This section describes how to change class, interface or enum size.

Left mouse click to select one element

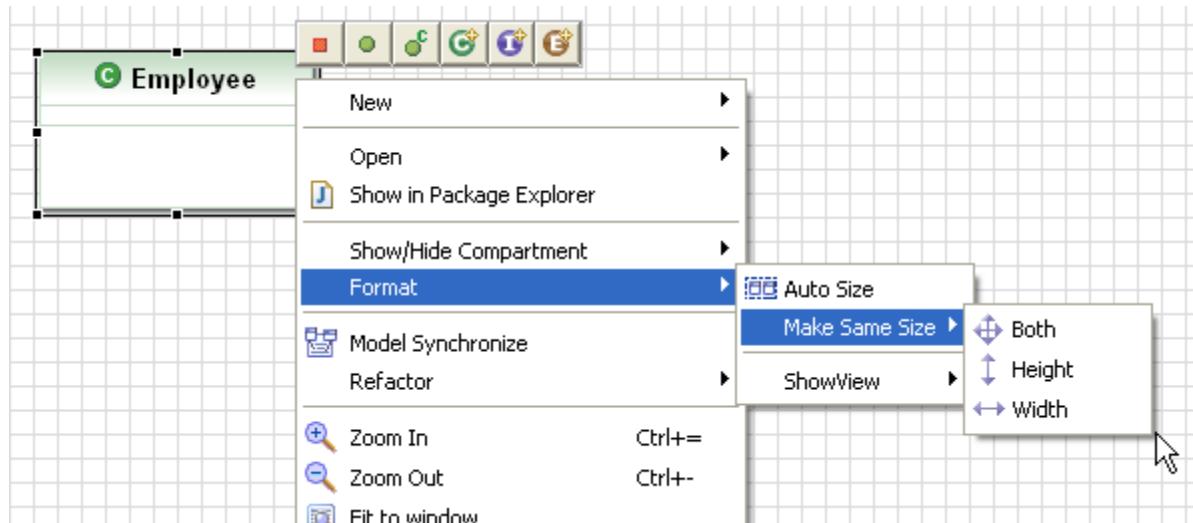
Left mouse clicks + shift to select multiple elements

Ctrl+a to select all the diagram elements

1. **The EclipseUML element Format allow to change an element or a group of elements existing size:**

Mouse left click in order to select the class, the interface or the Enum > popup menu > Format

- **Auto Size** will make the same size as the last selected class, interface or Enum elements. It means that:
 - all classes will have the same size as the last selected class.
 - all interfaces will have the same size as the last selected interface.
 - all enumerations will have the same size as the last selected enum.
- **Make Same Size**
 - **Both** will make the same size as the last selected element.
 - **Height** will make the same height as the last selected element.
 - **Width** will make the same width as the last selected element.

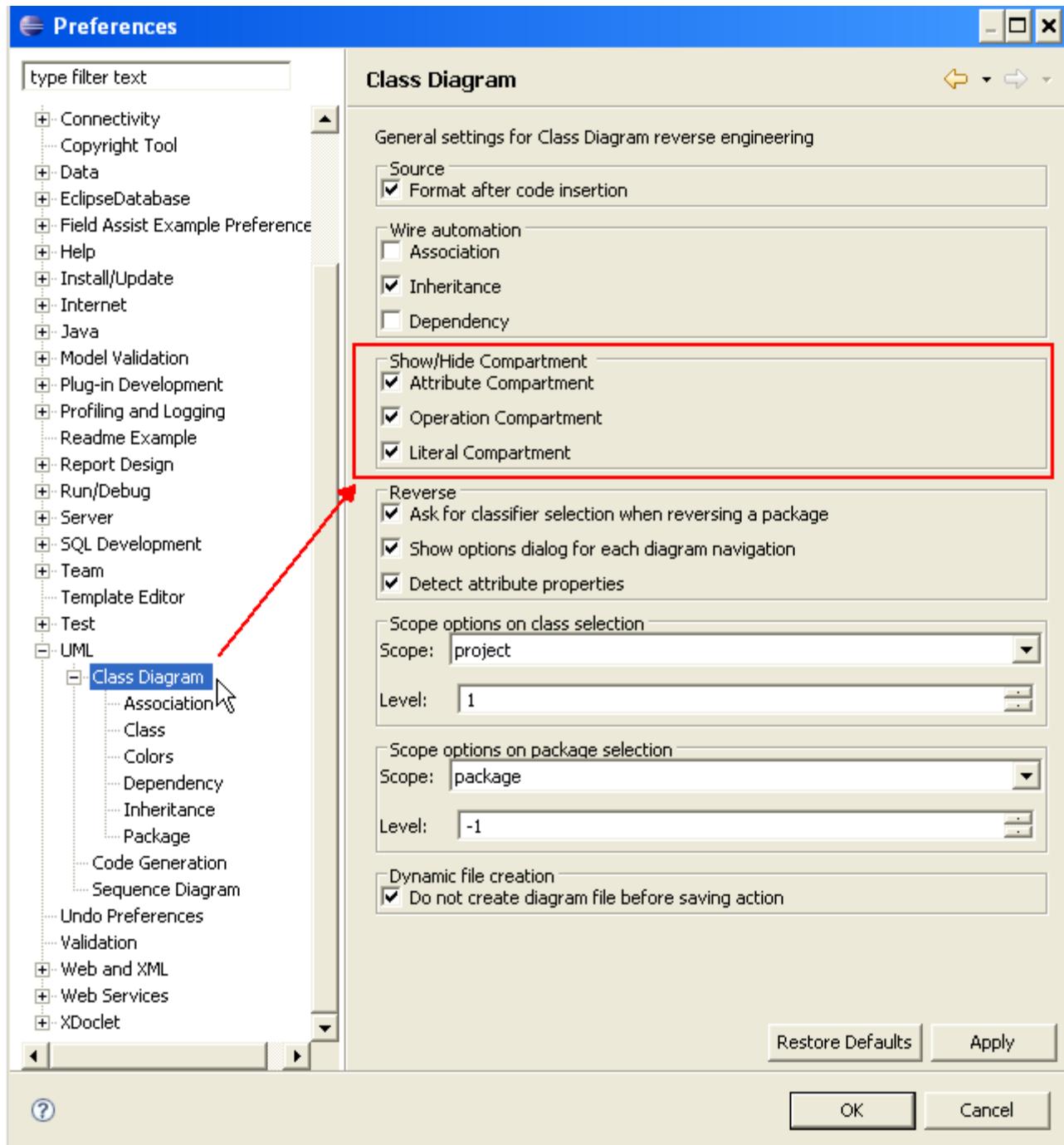


Show/Hide compartment

This section describes how to show or to hide compartments in the class diagram. The Show/Hide menu can be activated at:

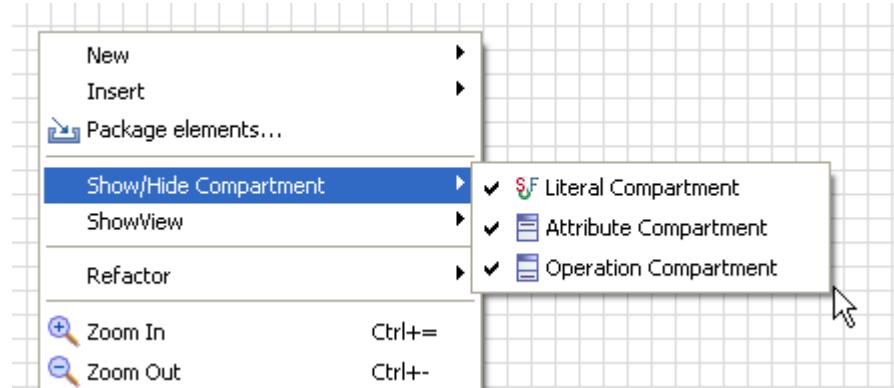
1. [EclipseUML global preferences](#)
2. [EclipseUML diagram preferences](#)
3. [EclipseUML element preferences](#)

1. The global preferences allow selecting the new diagram presentation:



2. The EclipseUML diagram preferences allows to change the existing preferences:

Mouse left click in order to select the diagram background > pop up menu > Show/Hide Compartment



- **Literal** Compartment will immediately show or hide all Enum literal compartment in the diagram
- **Attribute** Compartment will immediately show or hide all Class or Interface attributes in the diagram
- **Operation** Compartment will immediately show or hide all Class or Interface methods in the diagram

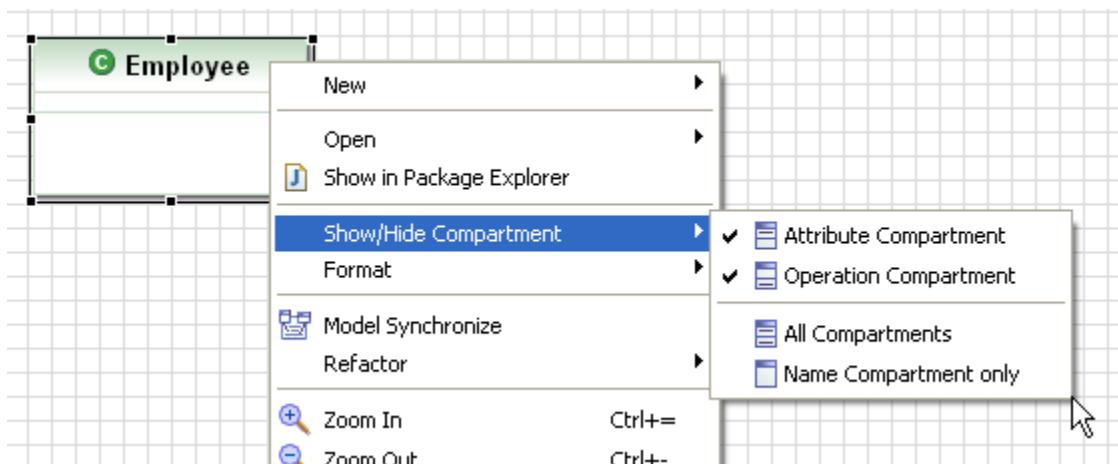
3. The EclipseUML element preferences allow to change an element or a group of elements existing preferences:

Mouse left click in order to select the class, the interface or the Enum > popup menu > Show/Hide Compartment

Left mouse click to select one element

Left mouse clicks + shift to select multiple elements

Ctrl+a to select all the diagram elements



- **Attribute** Compartment will show or hide the selected Class or Interface attributes in the diagram
- **Operation** Compartment will show or hide all Class or Interface methods in the diagram
- **All Compartments** will immediately show for the selected Class or Interface all Attributes and Methods compartments
- **Name compartment only** will immediately hide or the selected Class or Interface all Attributes and Methods compartments

Wires

In this section you will learn the wires associated features.

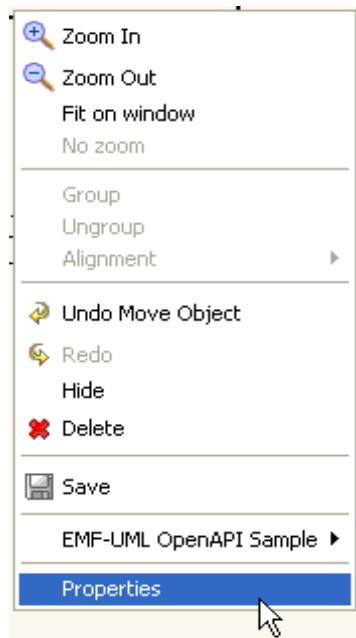
You can work on your diagram to adapt the organization using the following elements:

1. [Routers and Anchors](#)
2. [Wire colors](#)

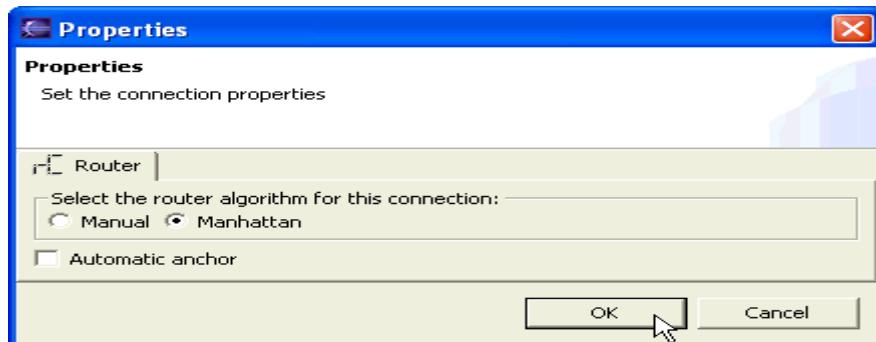
1. Routers and anchors

You can customize every link inside your diagram editor.

Right Click on a link between two elements->open the pop up menu->Properties

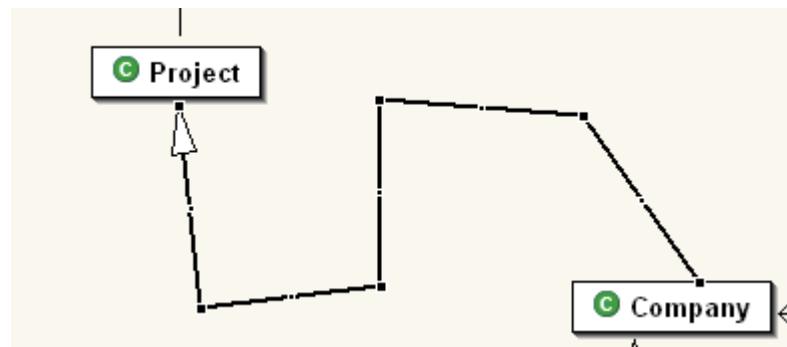


You can select the router algorithm and Anchor for each link



Router:

Manual allows moving the extremity of each link.

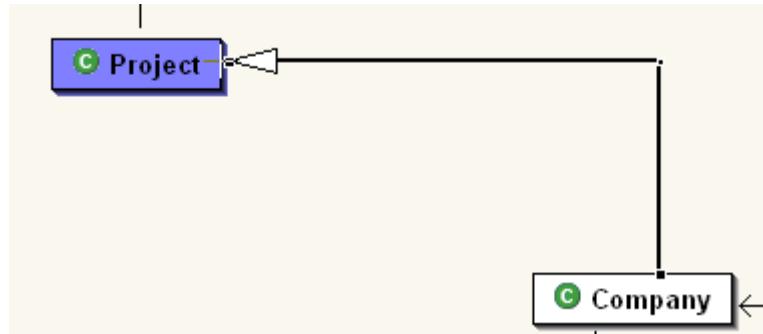


Manhattan uses automatic functions to design the link.

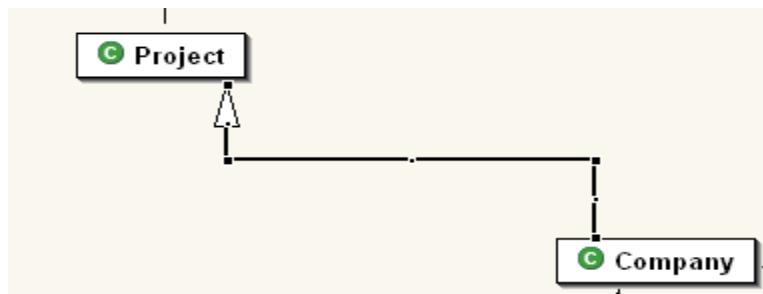


Anchor:

Unselecting Automatic anchor allows fixing the extremity anywhere. Move the anchor inside the element which will become blue when the anchor is properly fixed.



Automatic anchor uses algorithm to fix the anchor automatically (the anchor cannot be moved).



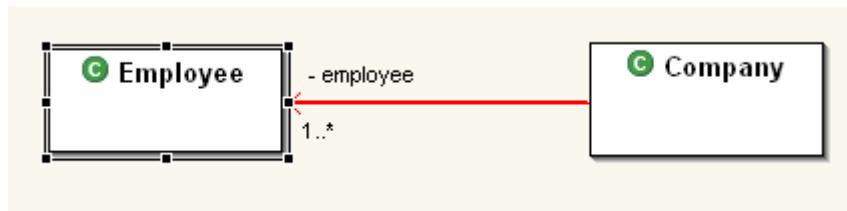
2. Wire colors

Selecting any element of a diagram will automatically colorize its related wires. In complex diagrams, this feature allows a better visibility of the immediate environment of a selected element.

Colors are different depending on the wire direction.

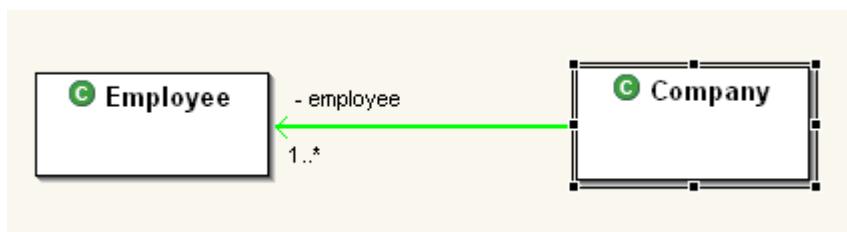
2.1 Incoming wires

All incoming wires will be displayed in red when selecting an element.



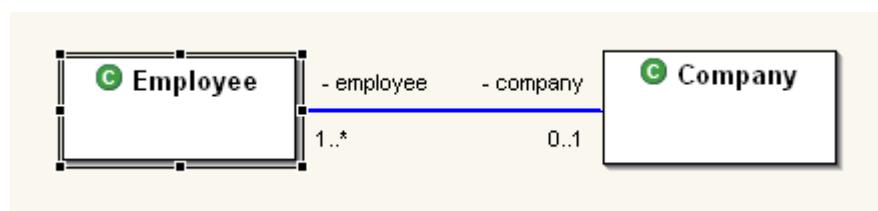
2.2 Outgoing wires

All outgoing wires will be displayed in green when selecting an element.



2.3 Bi-directional wires

Bi-directional wires will be displayed in blue when selecting an element.



Link Thickness

This section is about the OMONDO Link thickness menu.

You can highlight the important connectors and change the connector thickness

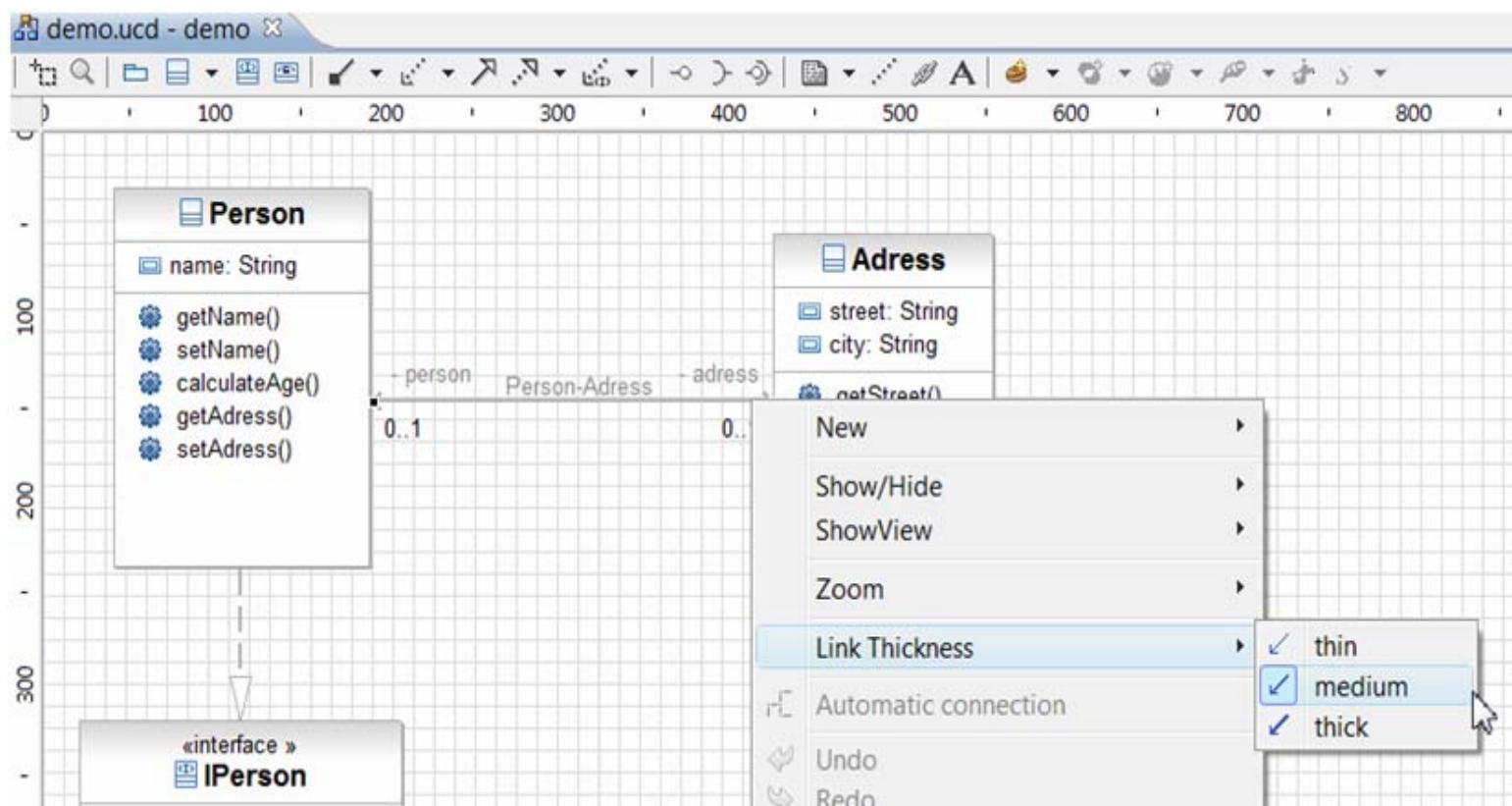
This option is working at diagram and element level.

Selecting the Link Thickness menu from the diagram contextual menu will automatically change all links thickness.

Selecting the link thickness menu from the element contextual menu, will only change the link thickness of this selected element.

Click on the link, open the contextual menu and select one of the Link thickness option:

- Thin
- Medium
- Thick



Zoom

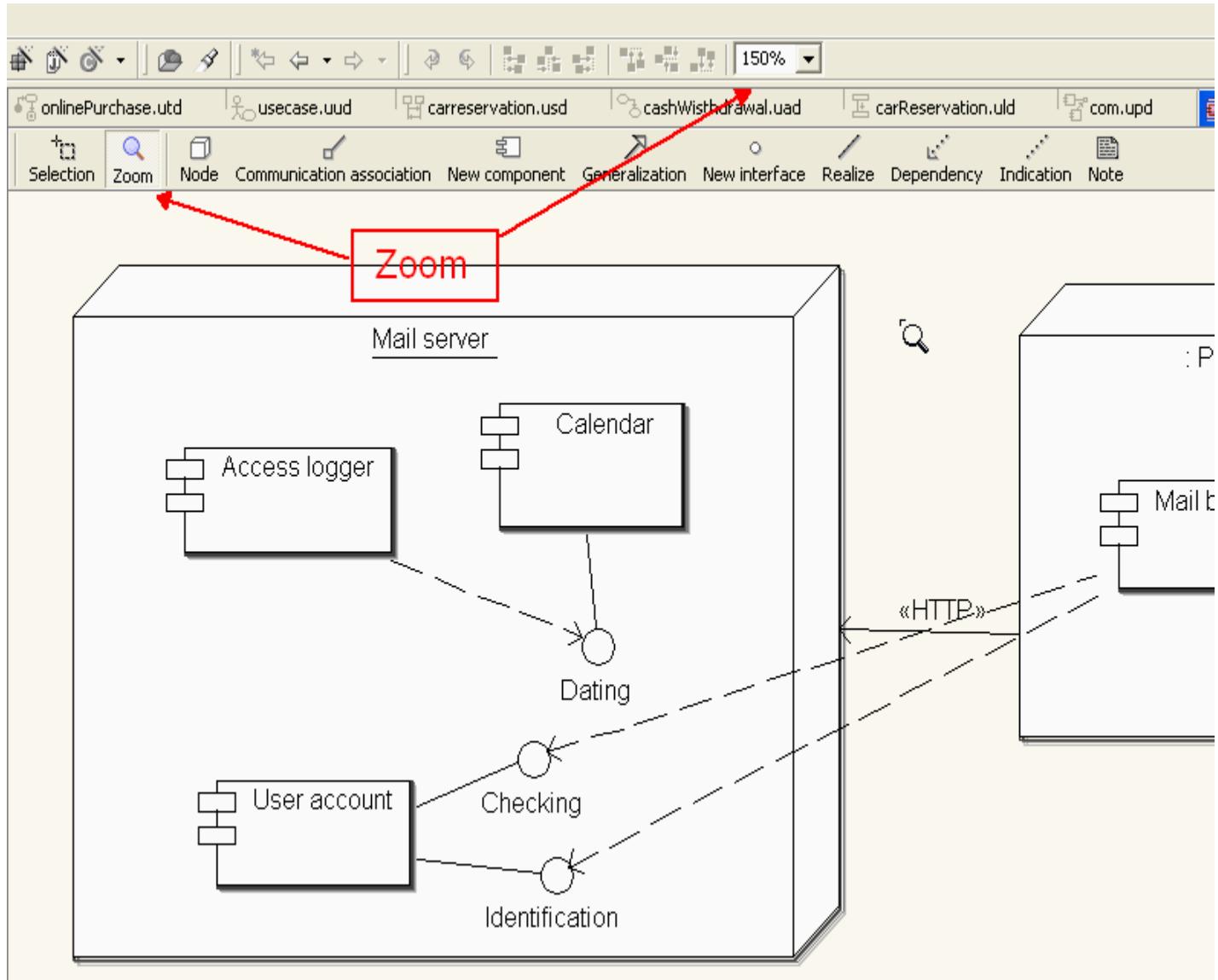
EclipseUML allows you to zoom in the diagram editor. By **selecting the zoom mode**, you can work in a more efficient way with your diagrams and be focused on your diagram, not only on your Java code.

Zooming is possible by selecting:

- the zoom icon in the toolbar
- the zoom percentage in the Workbench window's toolbar
- the diagram editor's popup menu

Three major functions are available within the Zoom function:

1. [Zoom in and out](#)
2. [Overview](#)
3. [Cropping](#)



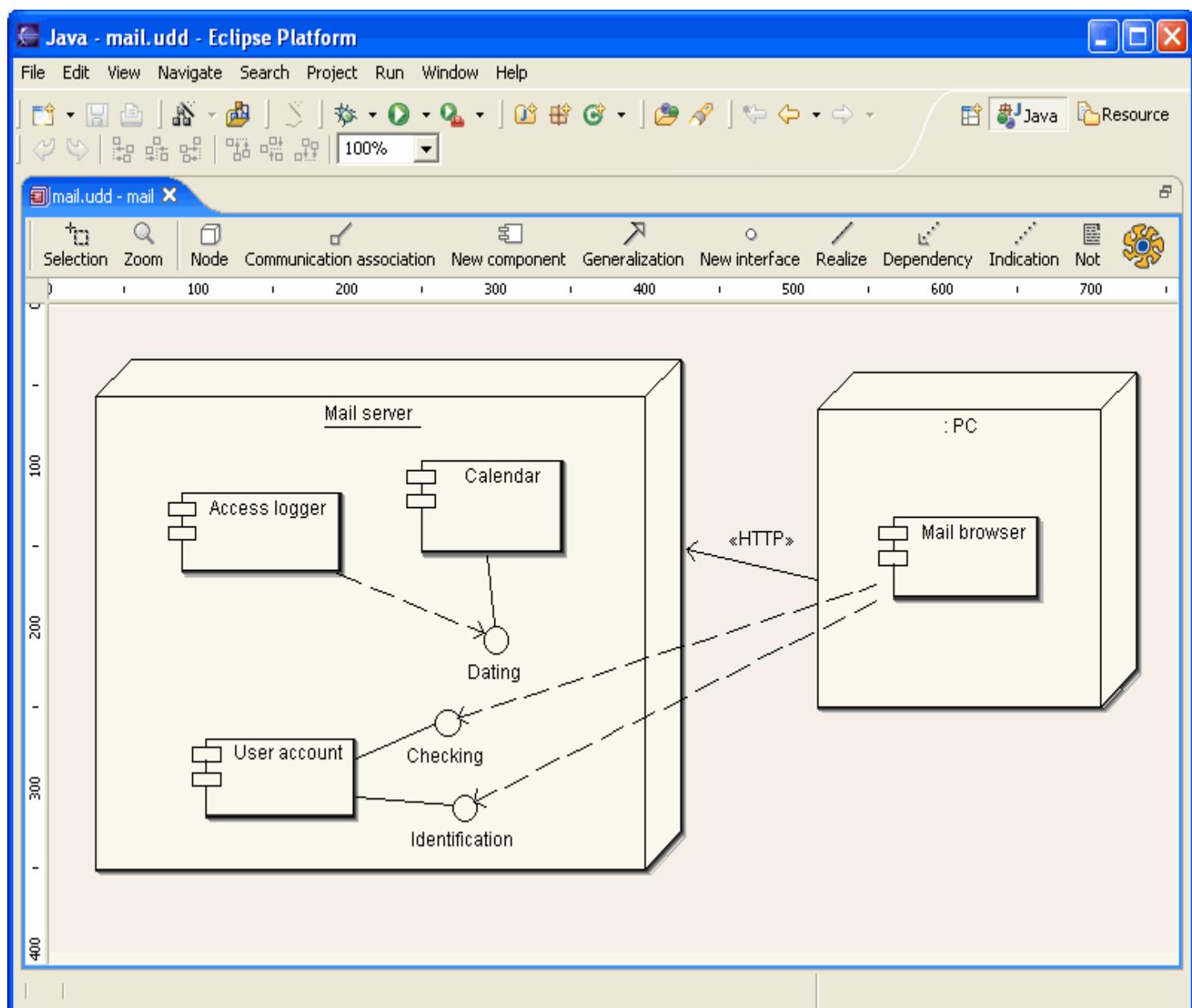
1. Zoom in, Zoom out

Selecting **the zoom icon in the toolbar** allows you to use the following zoom functions:

1. zoom in > right click
2. zoom out > left-click

2. Overview

You can see a complete **overview** of your diagram by selecting the zoom icon in the tool bar and **double clicking inside the diagram editor**.



Another double-click brings you back to the previous state.

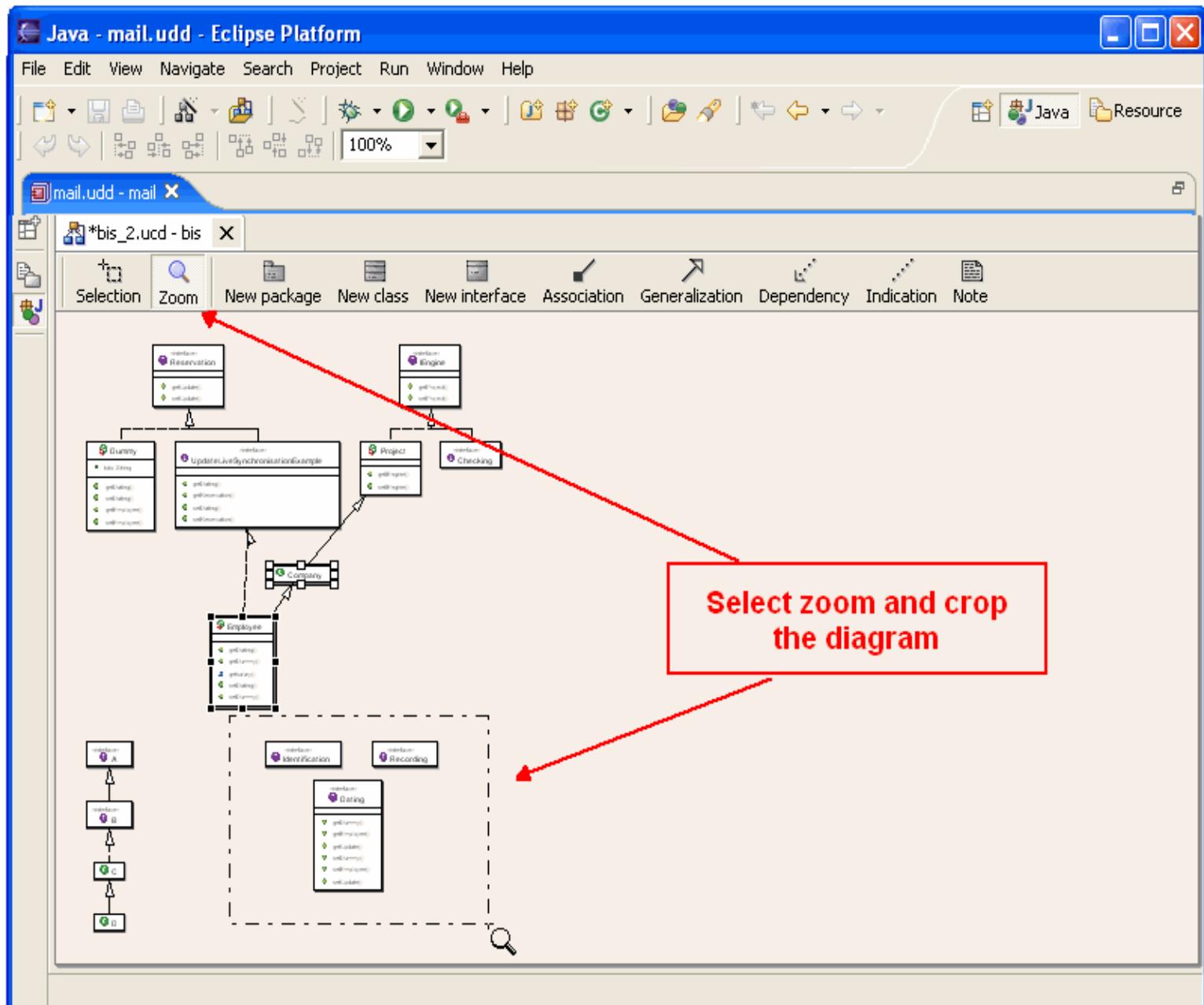
3. Cropping

You can select a part of the diagram that you want to zoom in.

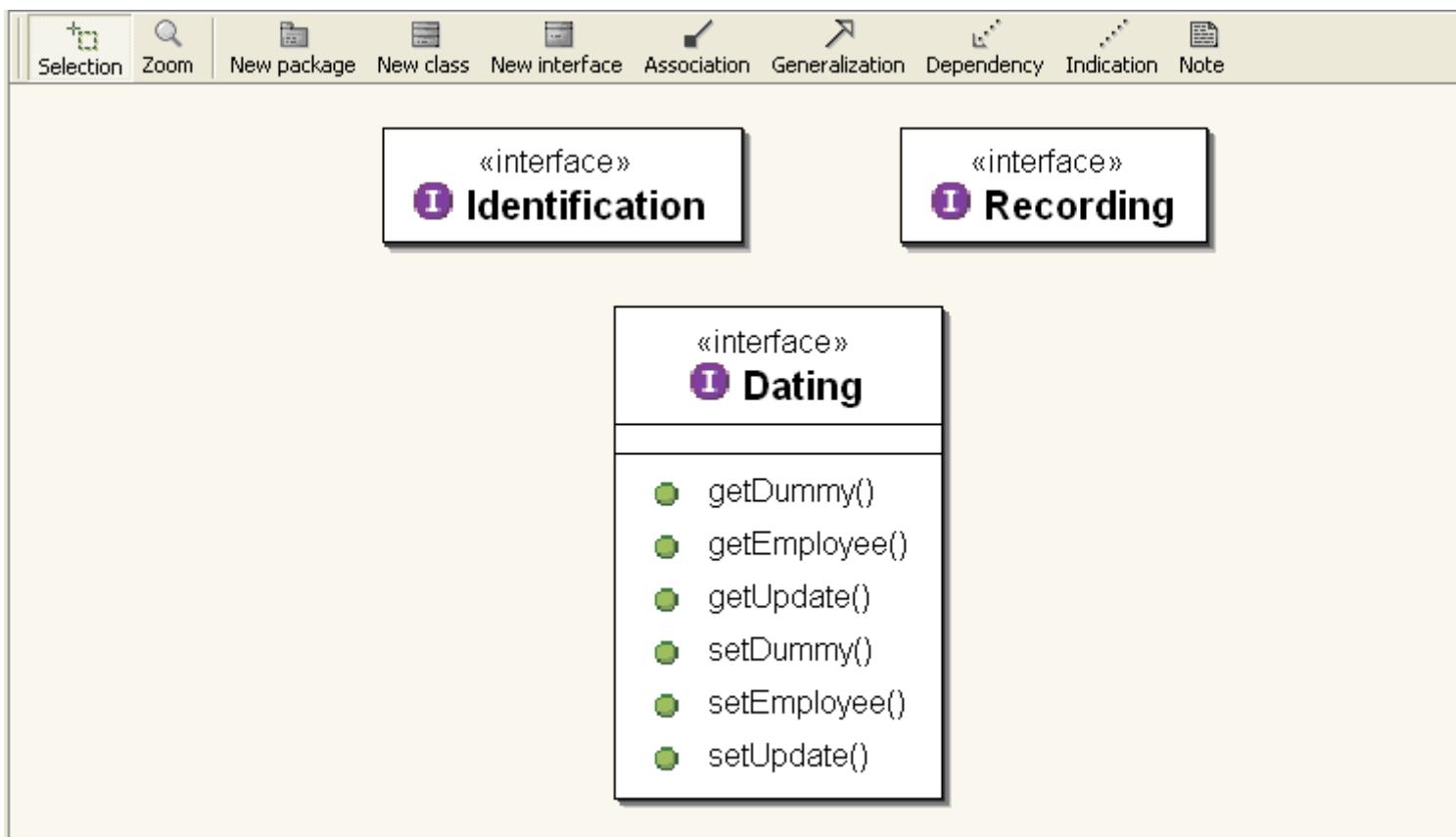
Select the Zoom icon in the toolbar and crop inside the diagram editor.

In the following example, we have an overview of a class diagram. We need to zoom inside this diagram and decide to crop.

Zoom in the toolbar must be activated, then use the mouse to crop.



As soon as the mouse button is released, the selected area is enlarged.



Print options

The print function is available from:

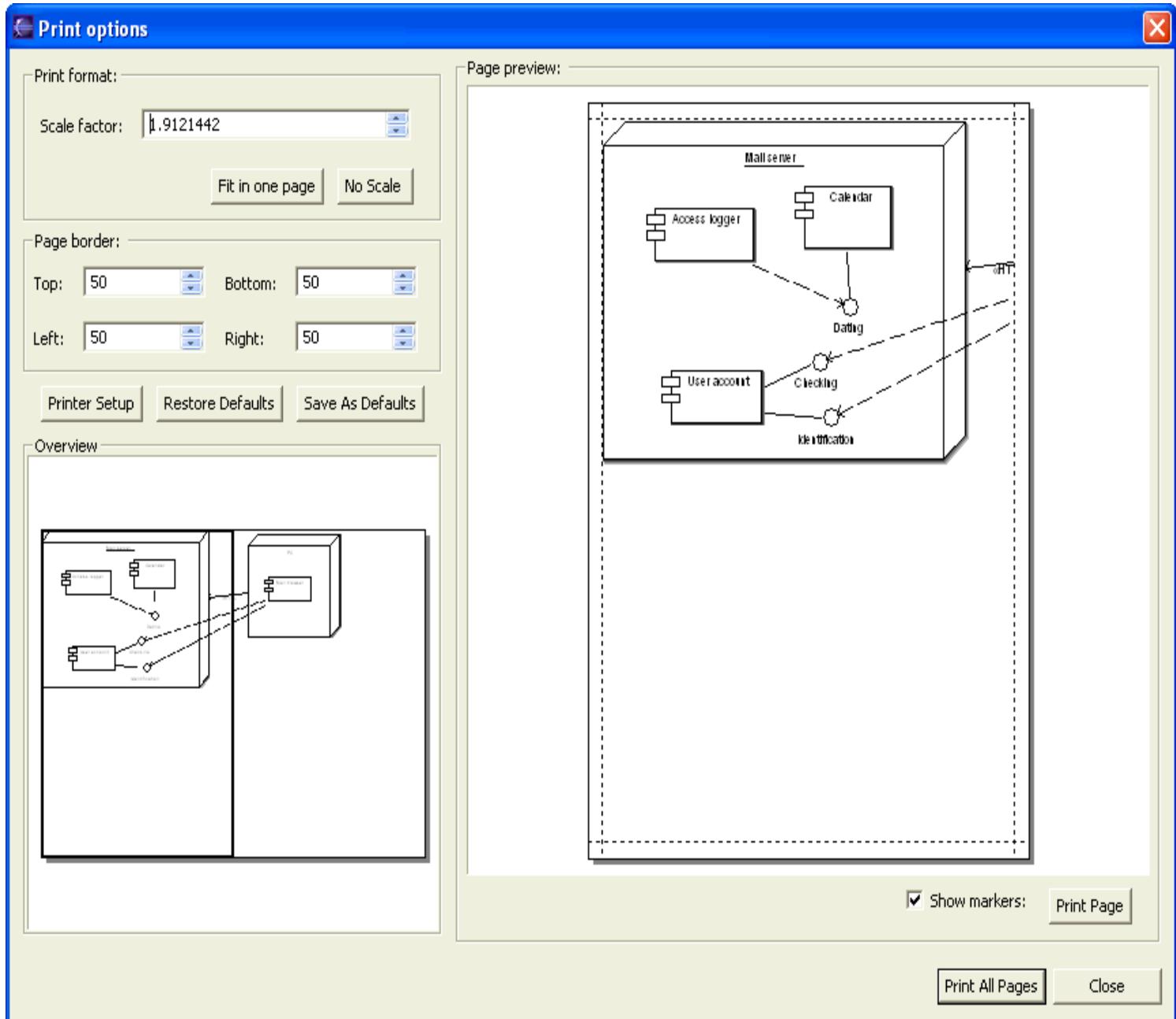
- The diagram editor's popup menu > print (to open the popup menu, right click inside the diagram editor and be sure not to select any element).
- From the menu bar select **File > Print**

Print option allows you to choose four different parameters:

1. **Print format:** adapt the scale factor and have a look at the page preview to see the result. Fit on one page will automatically select the appropriate scale.
2. **Page border:** indicates the border customization of your diagram editor image and its export to your printer.
3. **Overview:** shows all the printing pages of the document. Selecting a page in the overview will automatically select this page in the Page preview window.
4. **Page preview:** shows the page before it is printed.

Diagrams can be printed using Print option or keeping the global [Print preferences](#).

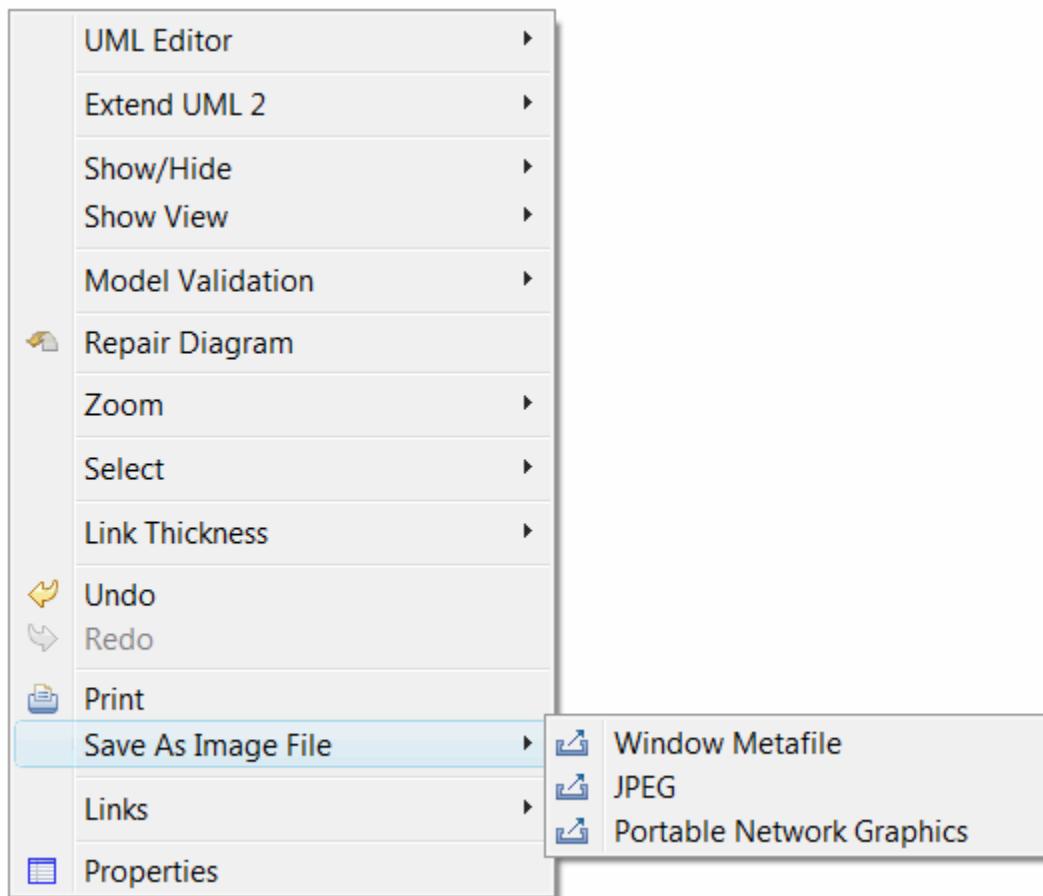
The following picture is an example of the Print option window.



Export as Image

Different exporting formats are available.

You can access this feature from the diagram editor's popup menu by right-clicking on the background of a diagram.



Several formats are available for export:

- Window Metafile (WMF)
- JPEG
- PNG

Special Characters Support

Diagrams using special characters such as Japanese, Korean, Chinese, etc... need to use an appropriate font. The export font can be configured in the [preferences](#).

Customize perspective

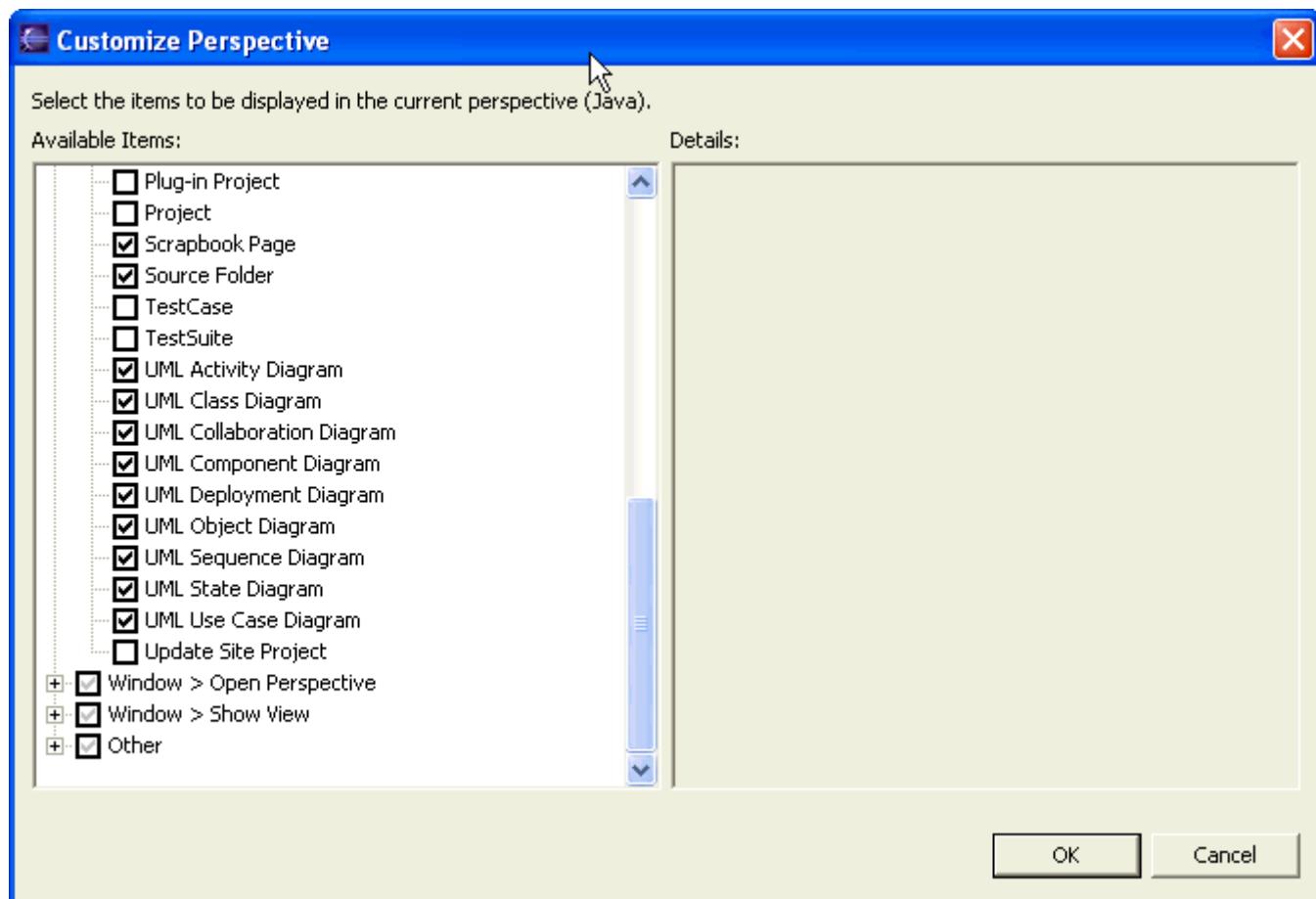
It is important to customize your Eclipse workbench perspectives.
EclipseUML diagrams and functions can be customized using the Customize Perspective.

You can customize your diagram perspective using the following elements:

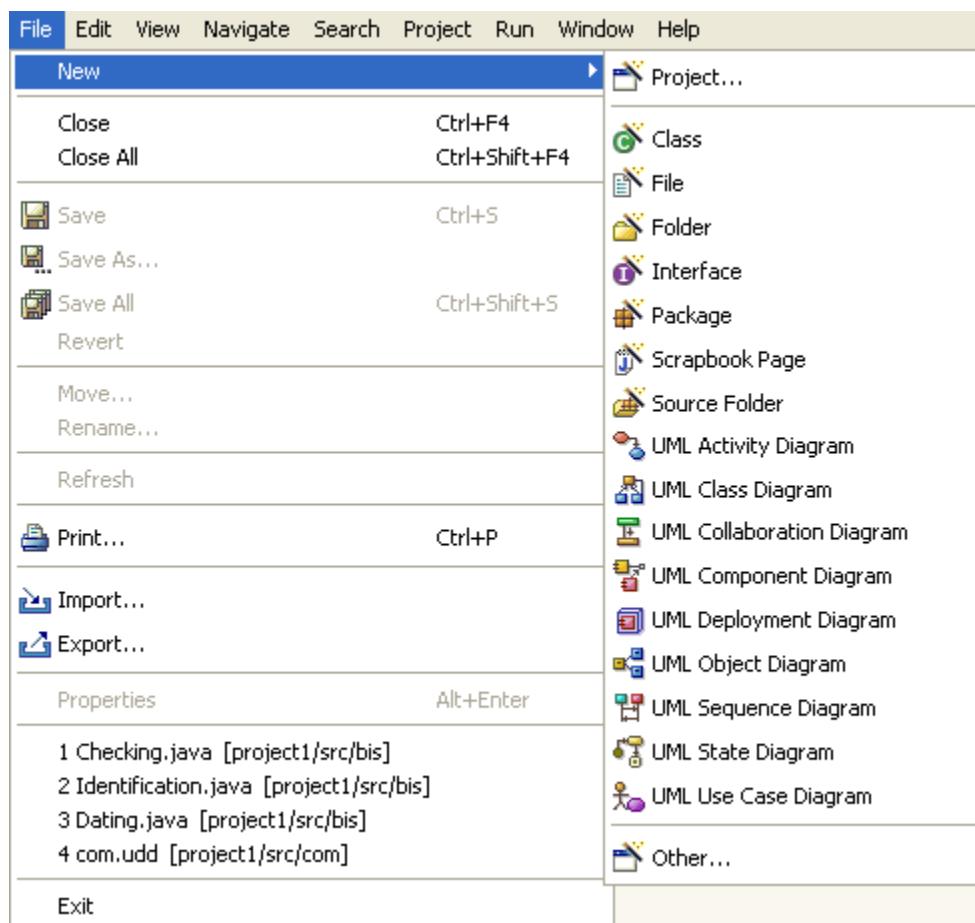
1. [Customize Perspective](#)
2. [Toolbar](#)

1. Customize Perspective

Select in the menu bar **Window > Customize Perspective > File > UML** submenu



Every selected diagram will be available directly from the menu bar: select File > New submenu

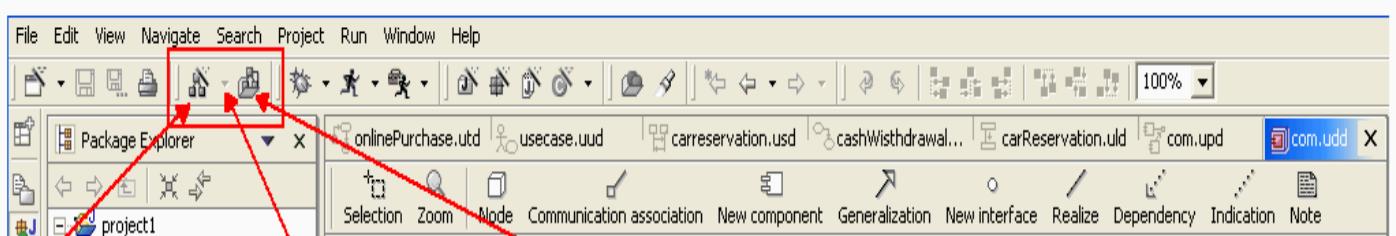


2. Toolbar

The workbench window's toolbar can be customized for UML needs. Select from the menu bar **Window > Customize Perspective > Other > UML**

The three following icons will be shown in the Workbench window's toolbar.

1. Create a class diagram by clicking on the icon
2. Create UML diagrams by activating the drop-down menu (simply click on the down arrow)
3. Open an existing diagram by clicking on the icon



Create Class Diagram

Create UML Diagram

Open an existing Diagram

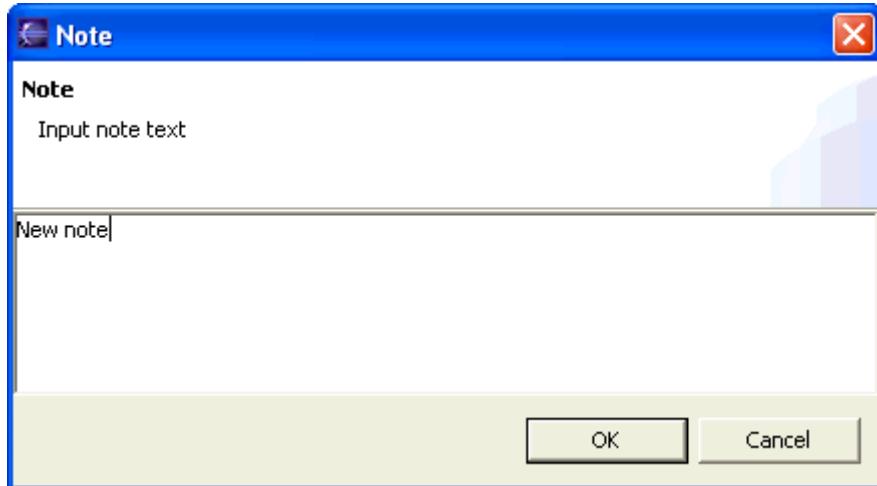
Notes

This section describes how to create notes in a diagram.

New note

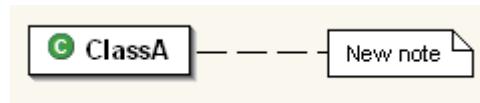
Select the note icon in the tool bar and click inside the diagram. The note will be created at the mouse position. A dialog allows you to edit the note.

You can change the content by double clicking on the note in the diagram.



New indication

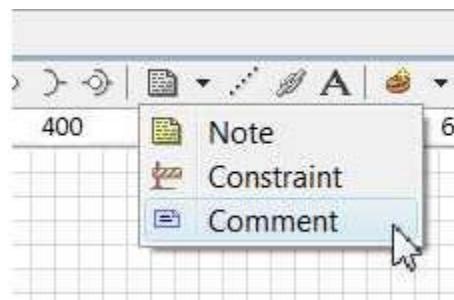
Select the indication icon in the tool bar. The indication will link the note to any diagram element.



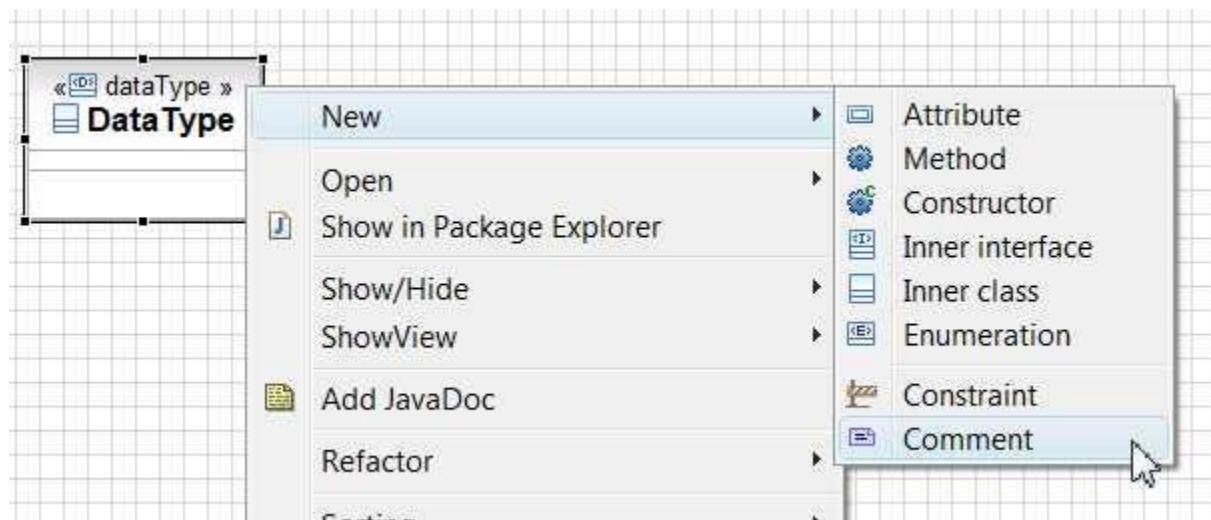
Comments

This section describes how to add comments in any UML diagram on any element.

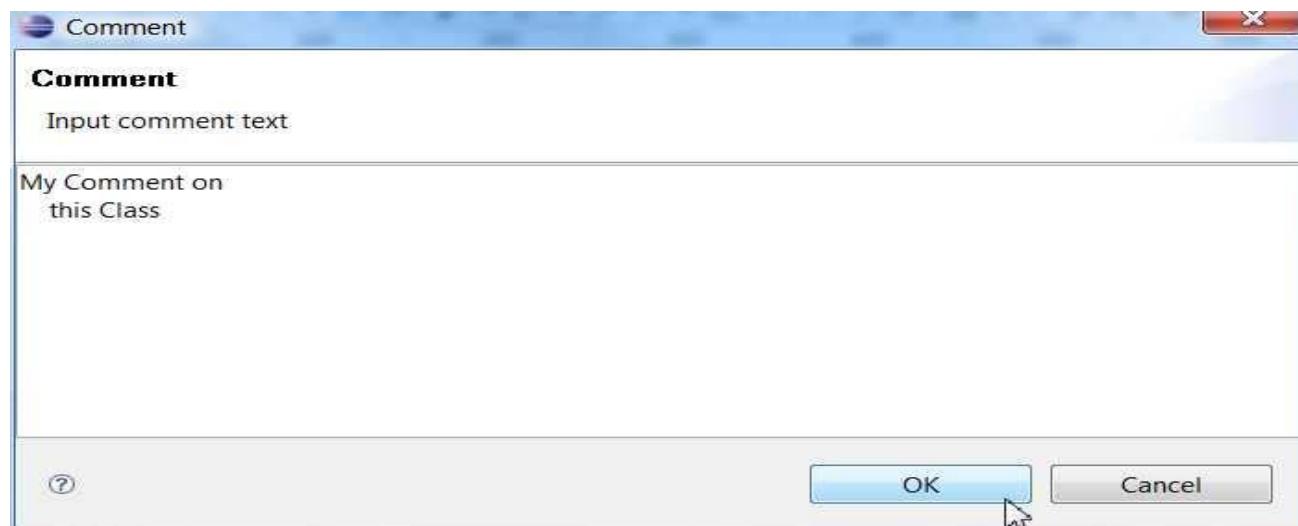
You can add Comment inside the diagram using the toolbar and then drag and drop from the toolbar to the UML editor.



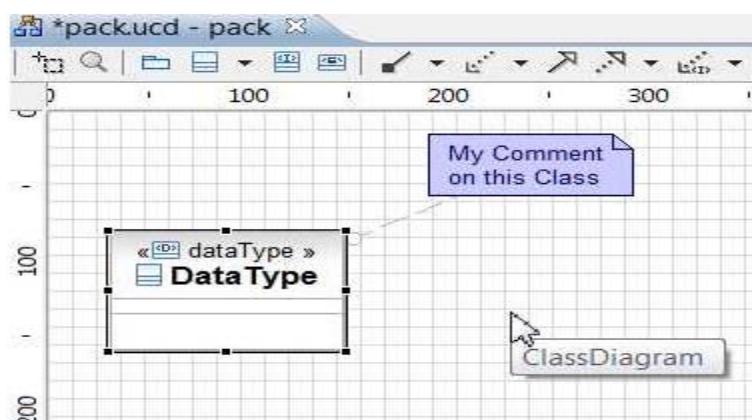
Comments can also be added on any element of any UML diagram. Select the element and open the popup menu > **New > Comment**



Insert the comment in the text field and click on the Ok button. You can have a two line comment by clicking not the enter button.



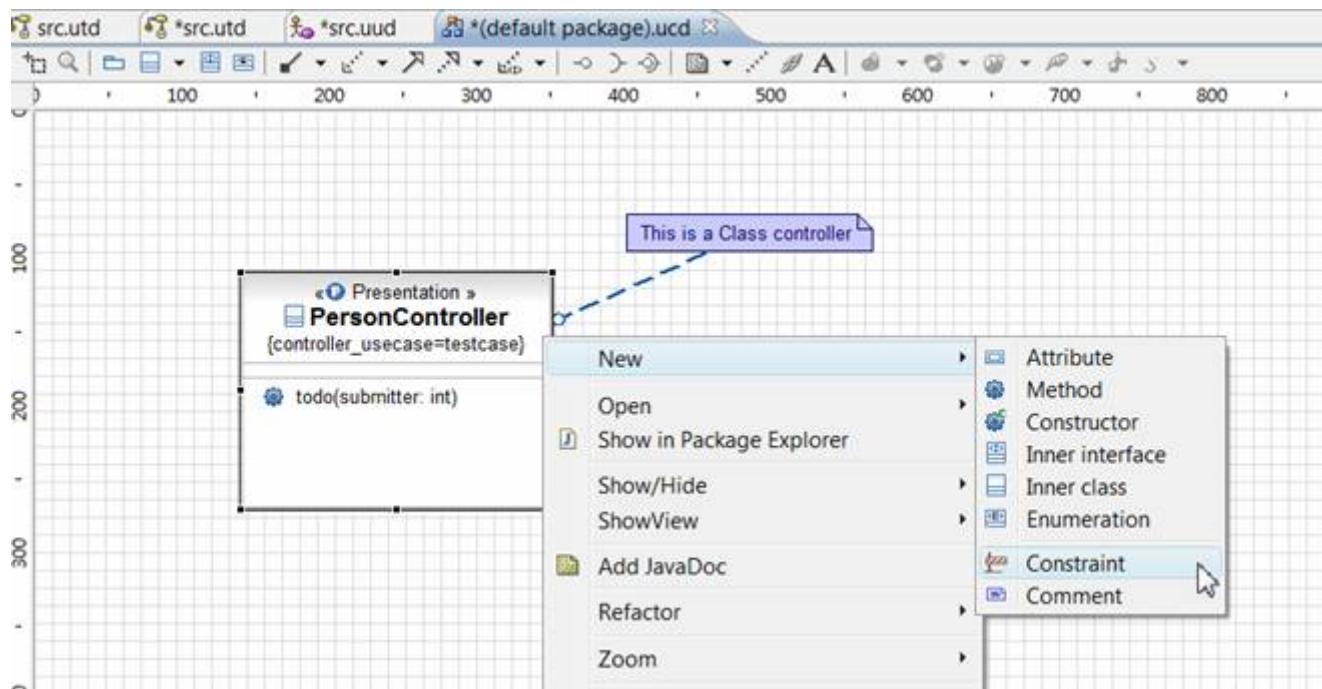
Get your comment in your class diagram



Constraints

This section describes how to add constraints in any UML diagram on any element. You can add Constraint inside the diagram using the toolbar and then drag and drop from the toolbar to the UML editor.

Constraints can also be added on any element of any UML diagram. Select the element and open the popup menu > **New > Constraint**



Select

This section describes how to select shapes and connectors.

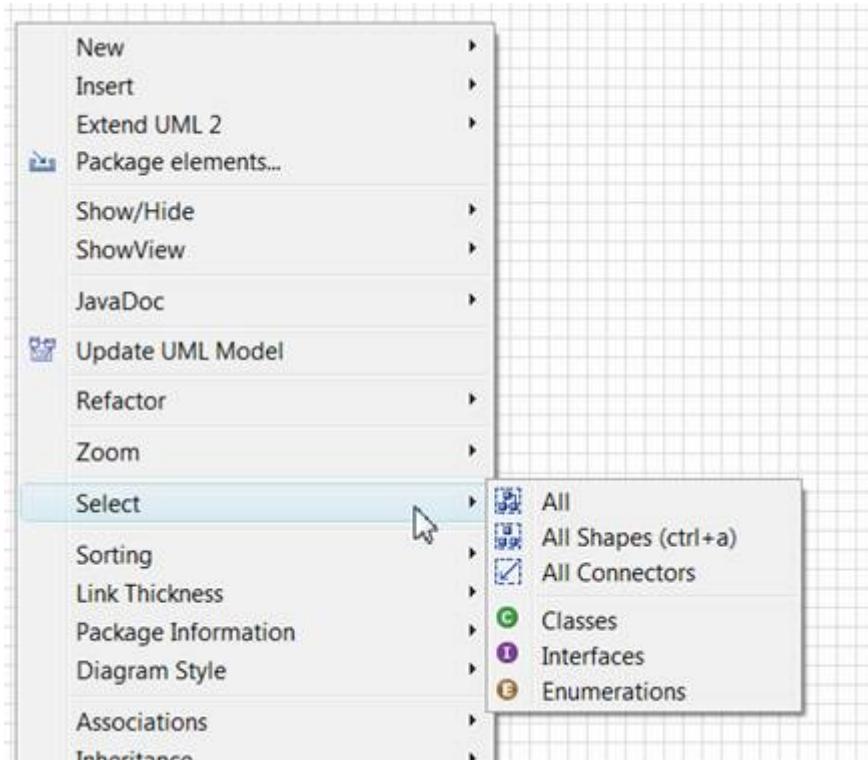
The Select menu is available from the diagram popup menu.

It works as a group of selected elements.

For example a cool feature is select > All Connectors and open the Font and colours view. This will allow you to immediately colour all the links from the diagram

Different options are available:

- **Select > All** will select all diagram elements
- **Select > All Shapes** will only select all shapes
- **Select > All Connectors** will only select all links
- **Select > Classes** will only select Classes
- **Select > Interfaces** will only select Interfaces
- **Select > Enumerations** will only select Enumerations



Label

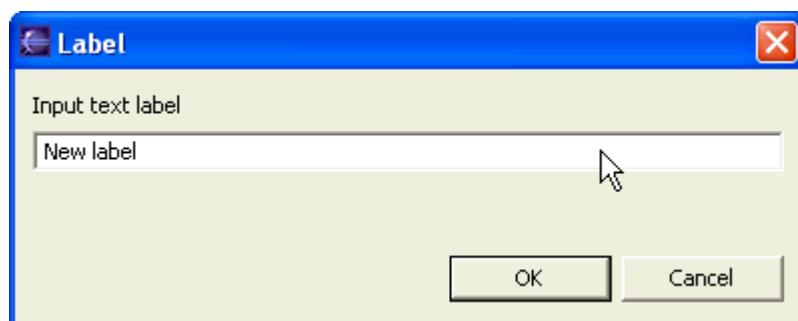
This section describes how to create labels in the diagram.

A New label

Select the label icon in the tool bar and click inside the diagram.

A dialog allows you to edit the label.

You can change the content by double clicking on the label in the diagram.



The label will be created at the mouse position.



Link

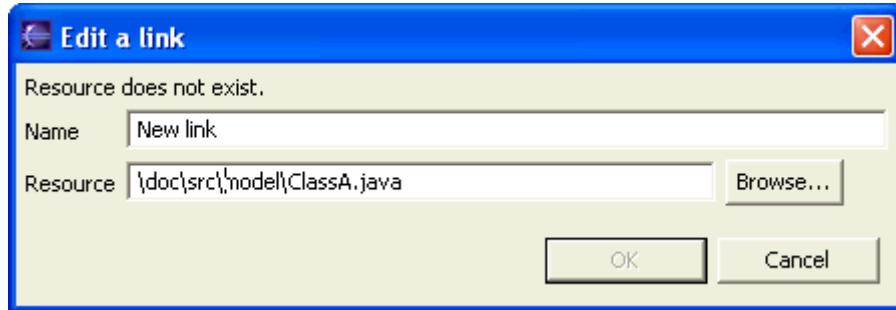
This section describes how to create links in the diagram.

New link from the tool bar

Select the link icon in the tool bar and click inside the diagram.

A dialog allows you to edit the link.

A link can be associated to any element of the workspace.



The link will be created at the mouse position. The displayed icon indicates the kind of resource which has been linked.

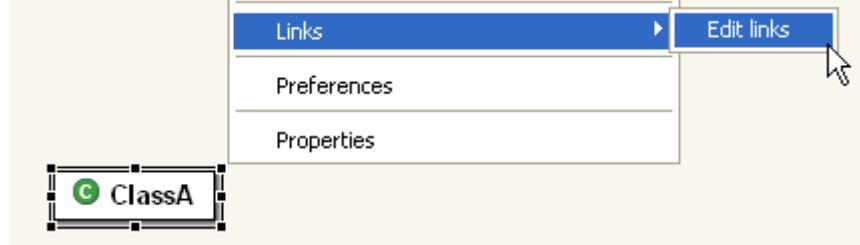


The link can be accessed and edited through the diagram pop-up menu.

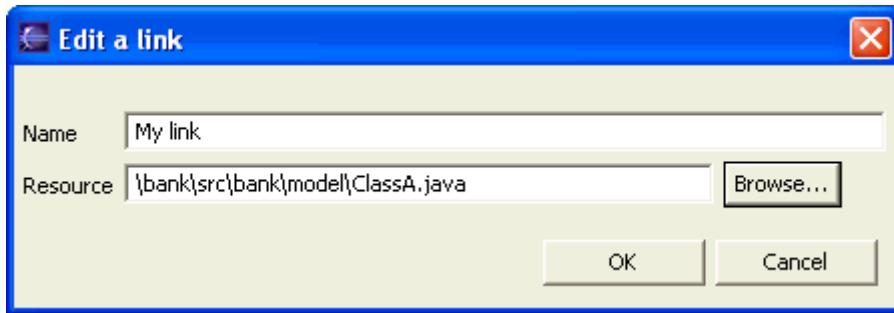


New link from a diagram element

Select any element inside your diagram, open the **popup menu** > **Links** > **edit links**



Enter the name of the link in the Name field and select the resource which will be linked to your diagram element.



For example, ClassA has a new link named My link.

You can navigate from ClassA element to another element of the workspace.

Open the **popup menu** > **Links** > **My link**



Key bindings

Here are the most useful key bindings to manipulate EclipseUML diagrams

Keys	Description
Ctrl+s	Save
Ctrl+a	Select all
Ctrl+z	Undo
Ctrl+y	Redo
Ctrl+p	Print
Ctrl+n	new wizard
Ctrl+F4	Close the diagram editor
Delete	Delete the selected figures with model
.	Enter the resize and moving mode. Repeating the key can switch different operations such as resize to left/right/up/down and moving.
Escape	Exit the resize and moving mode
tab	Change the tool bar selection
Ctrl+space	select or de-select a figure
Arrow	Change the figure selection with auto-scrolling
Shift+arrow	Append the figure selection
Ctrl+arrow	Change the focus
Ctrl+Alt+arrow	Move the selected figure
Ctrl+Shift+D	Launch the "Open Diagram" window to select a diagram in the project and open it

Property View

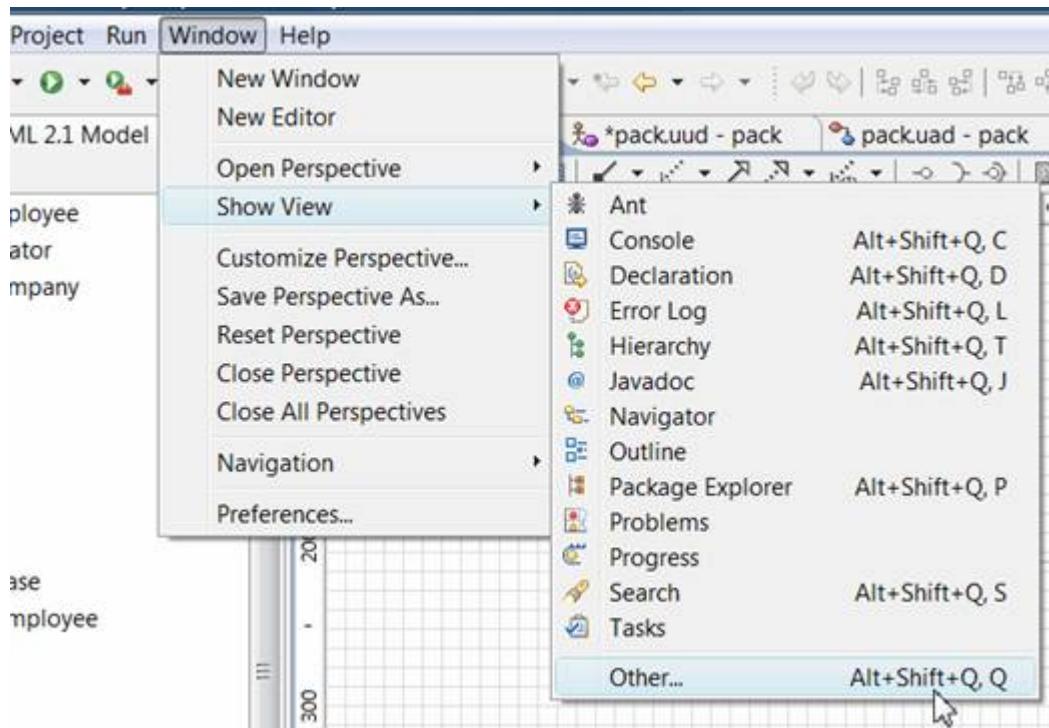
This section is about the OMONDO property view.

The property view gives information on your diagram elements.

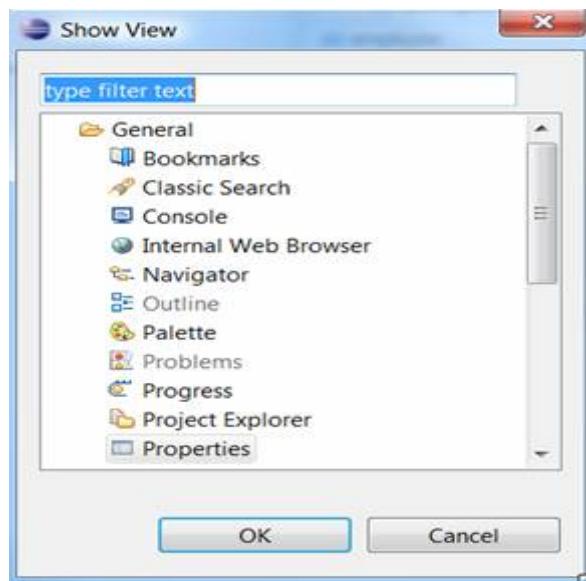
Only the class diagram is currently integrated.

You can open the property view by:

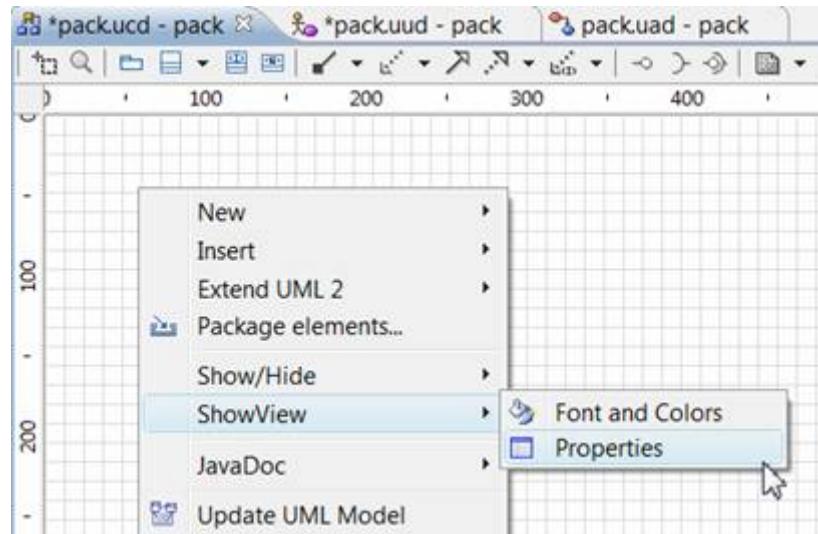
Selecting **Window > Show View > Other**



General > Properties



Or by selecting the class diagram contextual menu > **Show View > Properties**



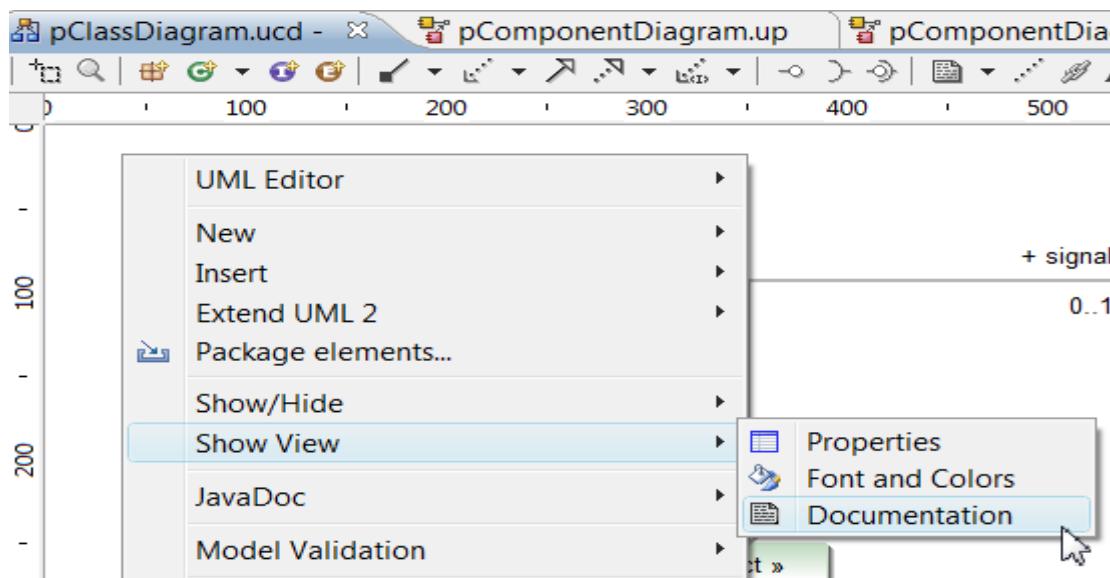
Documentation View

This section is about the OMONDO documentation view.

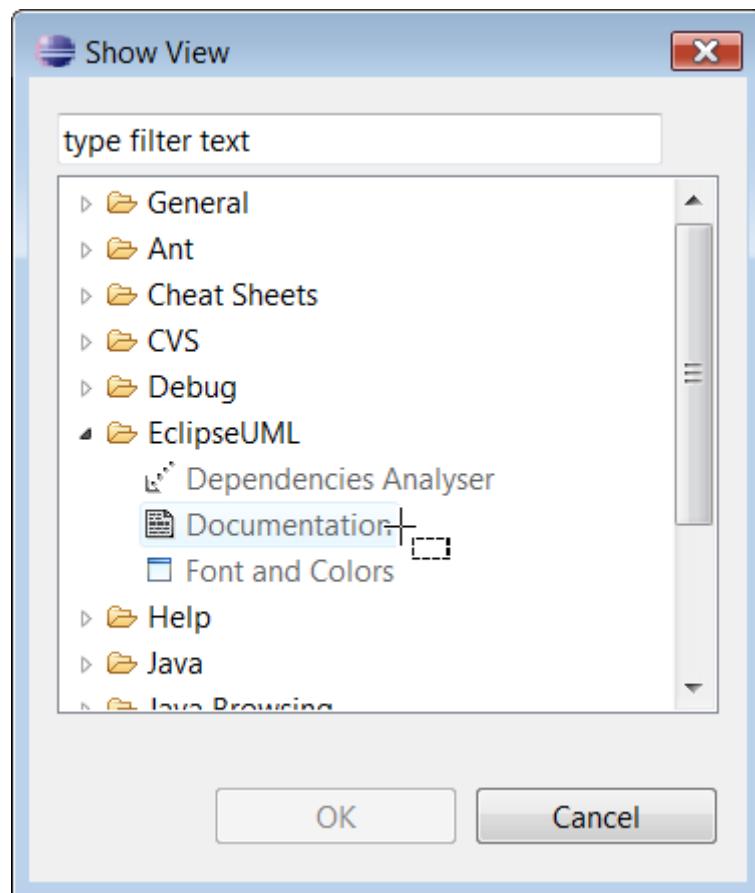
1. How to activate the Documentation View
 1. [Using the diagram contextual menu](#)
 2. [Using the Eclipse selection](#)
 3. [Using the Modeling perspective](#)
2. [How to use the Documentation View](#)

1. This section is about how to activate the Documentation View:

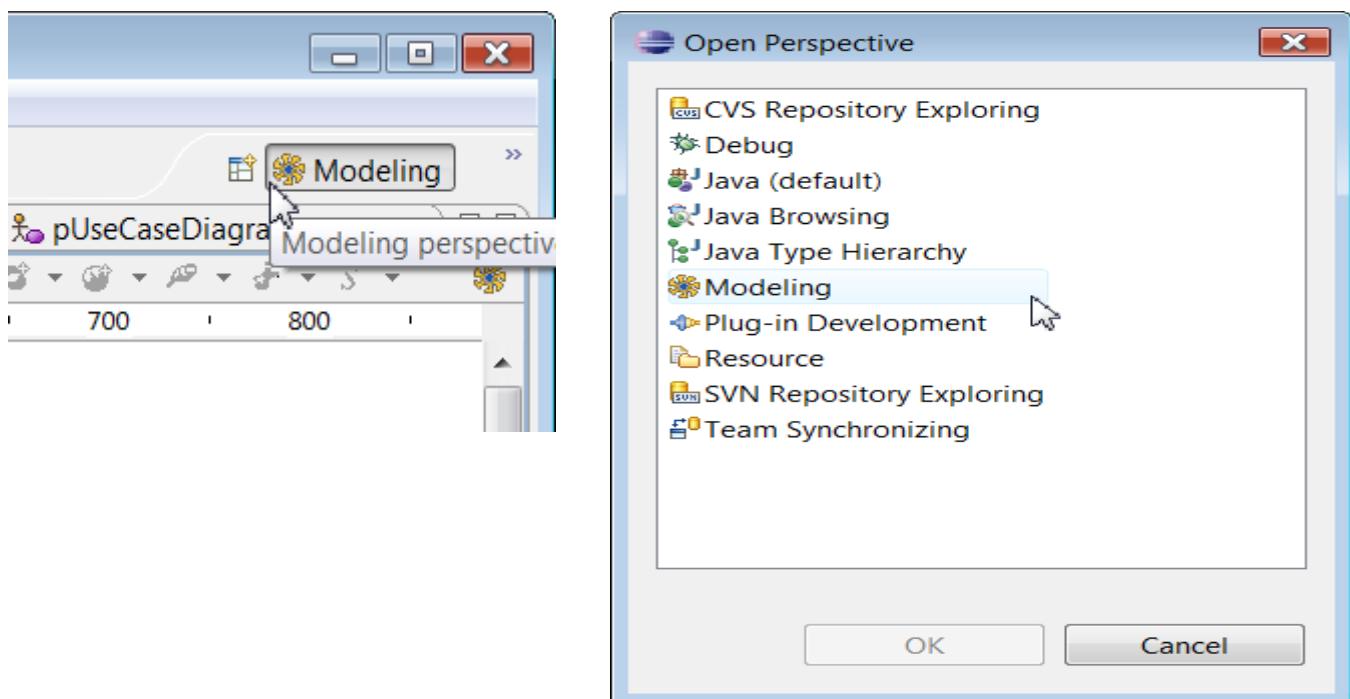
1. You can activate the Documentation View using the diagram contextual menu. Click on the diagram background > **Show View > Documentation**



2. You can also activate the Documentation View using the **Window > Show View > Other > EclipseUML > Documentation**

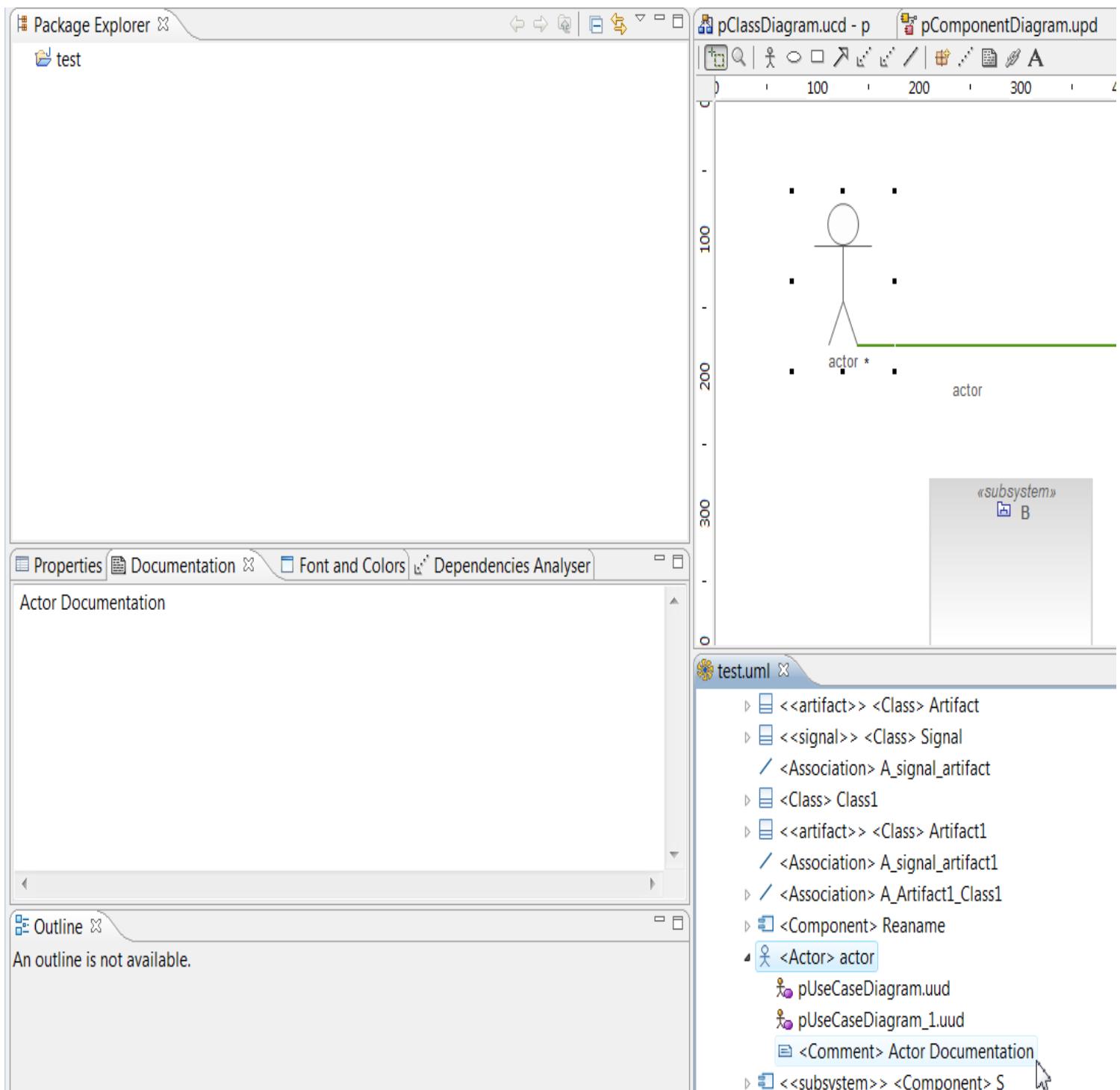


3. Note that the Documentation View is also included in the Modeling Perspective.
To activate the Omondo Modeling perspective, click on the Perspective tab in the upper right corner of your Eclipse and select Modeling.



2. This section is about How to use the Documentation View:

You need to select an element in the diagram by using the mouse left button. Then you need to open the documentation view by click on the documentation View tab. Finally you need to click inside the view and add your documentation.



Each UML element has its own documentation saved in the UML metamodel.

If no information is saved in the Documentation View then the Comment in the model editor is not created.

EclipseUML use the element OwnedComment metamodel information to save the documentation which is written in the documentation view.

Reverse Engineering

- [Reverse Engineering](#)
- [Reverse Engineering example](#)
- [Java Reverse Engineering](#)
- [Reverse Engineering Architecture](#)
- Reverse Engineering (*5min*): [Flash demo](#) / [exe file](#)

Reverse Engineering

1. Java Code

Import your java code inside Eclipse then select one of the following options:

Don't forget to launch the reverse engineering once the full project has been imported and compiled by eclipse.

If you launch the reverse before the compilation end then the reverse will fail.

1. Java code
 1. [Create a detailed model including your full project information](#)
 2. [Create a view of your project](#)
 3. [Merge exiting model with modified java code](#)
2. [JPA annotations](#)
3. [Database Reverse](#)
4. [Sequence diagram reverse](#)
5. [Reverse engineering Example](#)

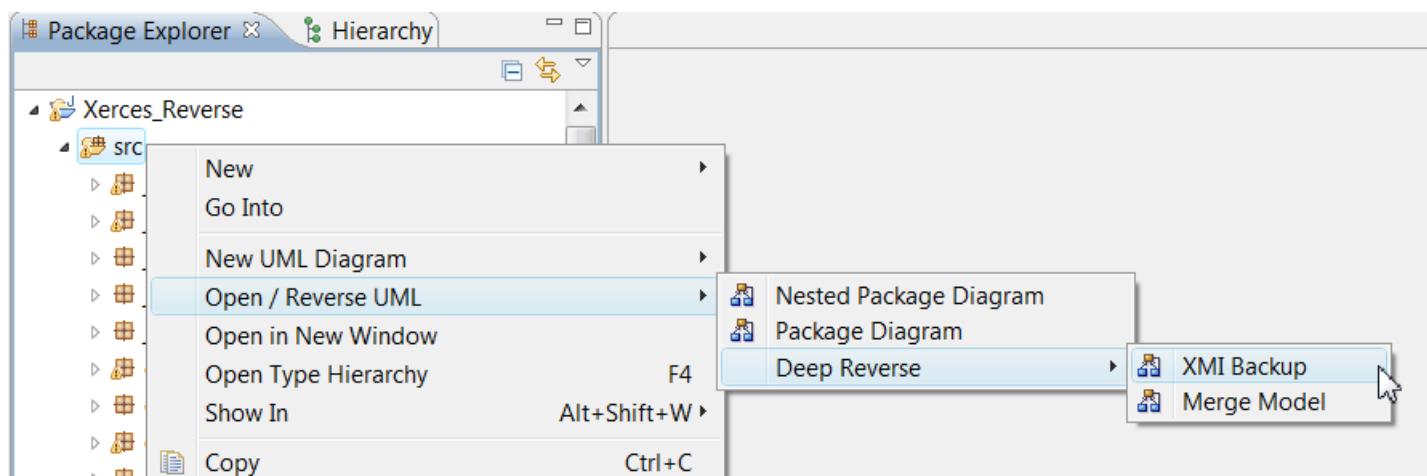
1.1 Create a detailed model including your full project information

We recommend mapping all your project information inside your model; btw you can also only use the automatic detection.

The XMI Backup will immediately save the following java project information:

- Classifiers
- Inheritance
- Associations
- Dependencies
- JPA Annotation

The XMI Backup menu is available in the Package Explorer by selecting **src > Open / Reverse UML > Deep Reverse > XMI Backup**.



This is a heavy refactoring process which requires 12 minutes on my laptop (*Intel duo core 2 GHz - ram 2048*) for Xerces (*large project size composed by over 1 000 classes&Interfaces*). This process will map each piece of Java information into a UML 2.2 model. This means that 100% of your Java or JEE project skeleton will be saved.

Please note that if you want to save business rules inside your model, then you need to manually reverse each method. This reverse is too heavy to be completed automatically (*will require between 12 to 24 hours for xerces*) and don't really have any sense to be done for each method. The best approach is to select strategic business rules methods and only [reverse these methods](#).

1.2 Create a view of your project

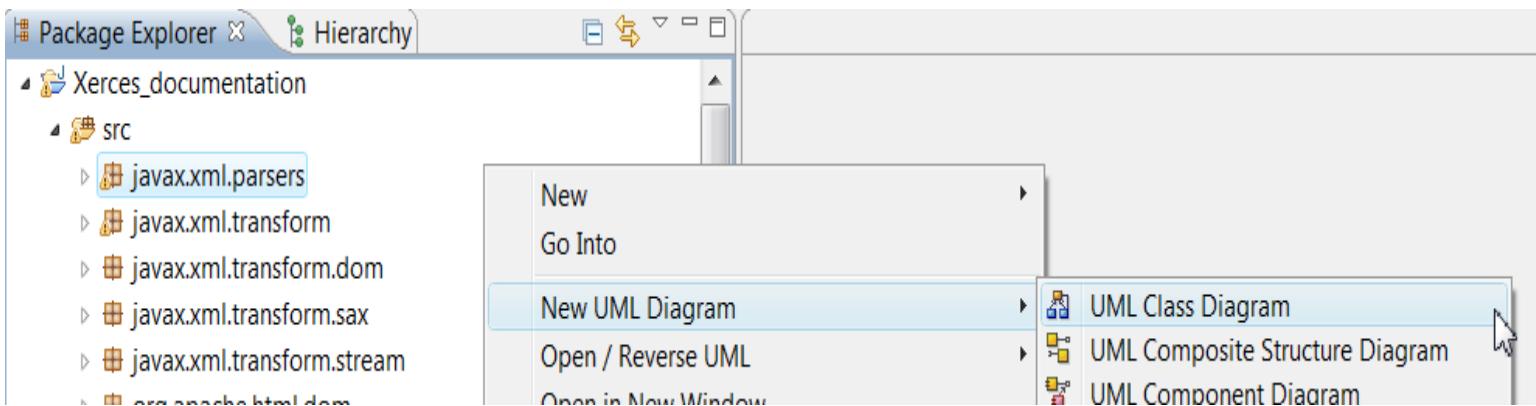
The create a view of your project requires to have java code and use automatic detection.

You can click on a package and reverse java code immediately at any time.

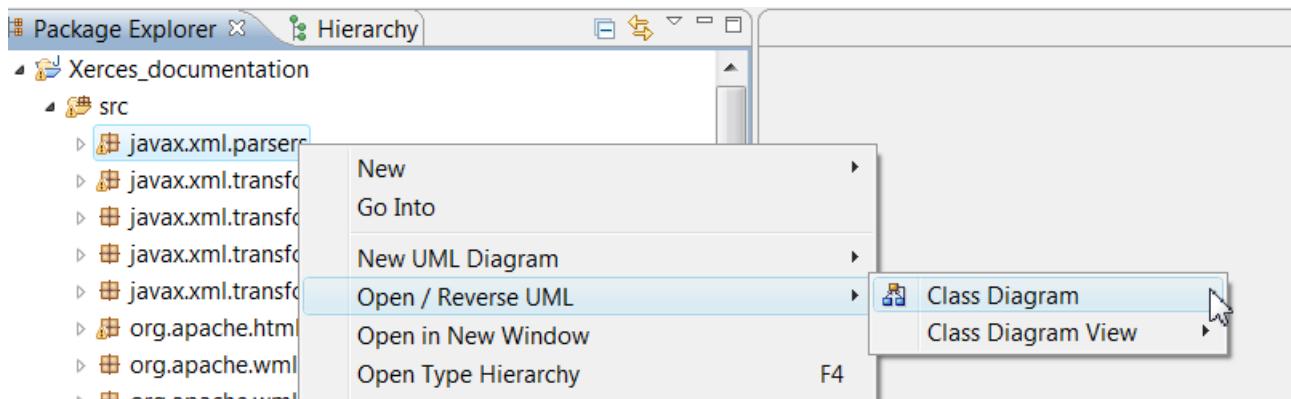
Note that only the created diagram represented by the package and classifiers will be saved in the model.

To reverse a package using automatic detection you can use three options:

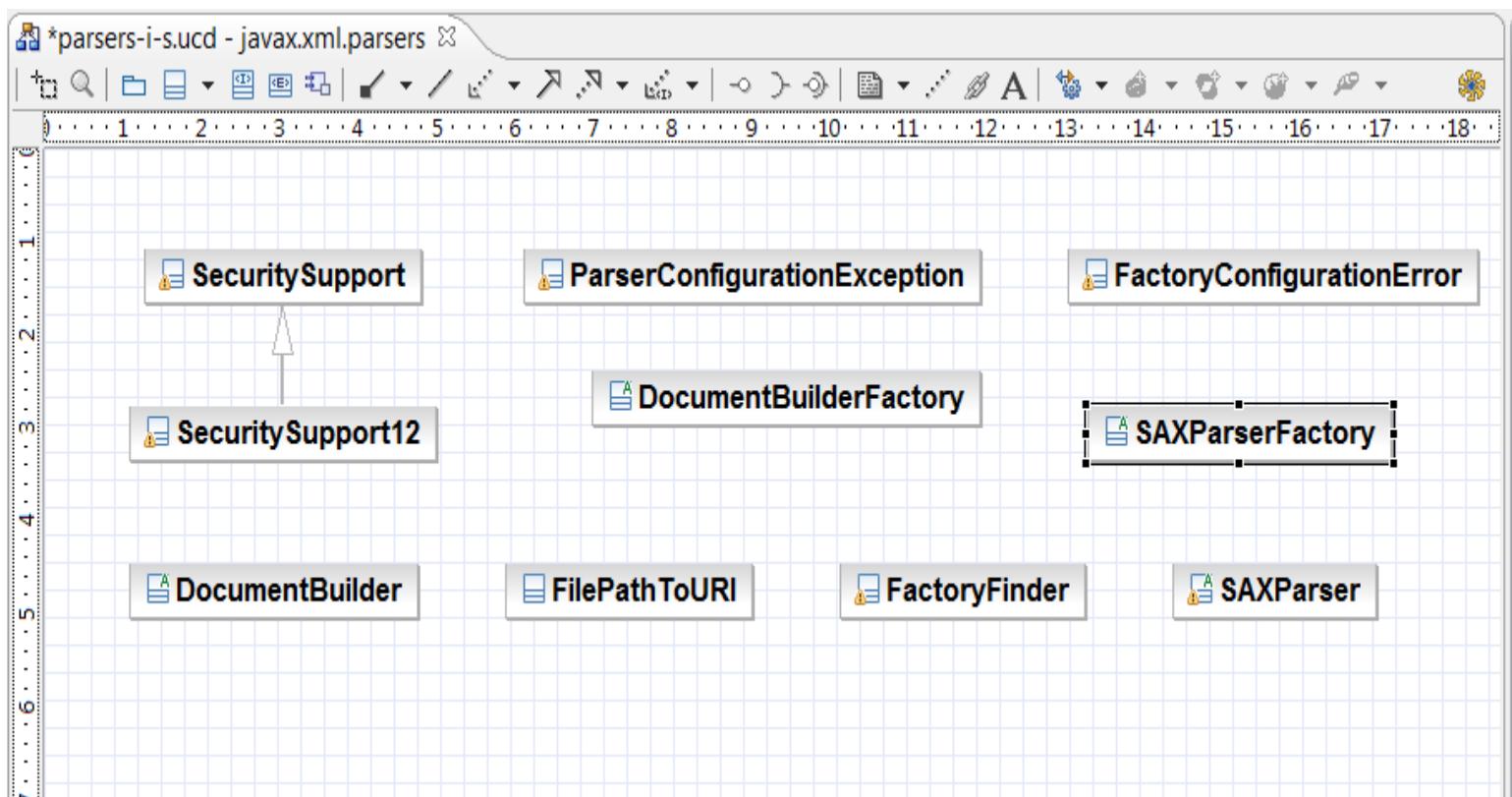
- click on the package > **New UML Diagram** > **UML Class Diagram** will allow you to select classifiers from a list and give a name to your diagram



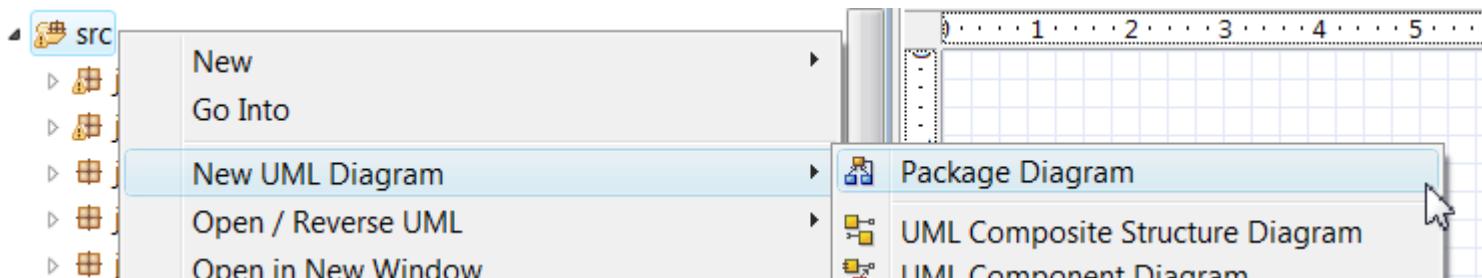
- Click on the package > **Open / Reverse UML** > **Class Diagram** or **Class Diagram View** will select all classifiers from the package and display the selected view (*e.g. inheritance, dependencies, associations and Classifiers with or without compartments*).

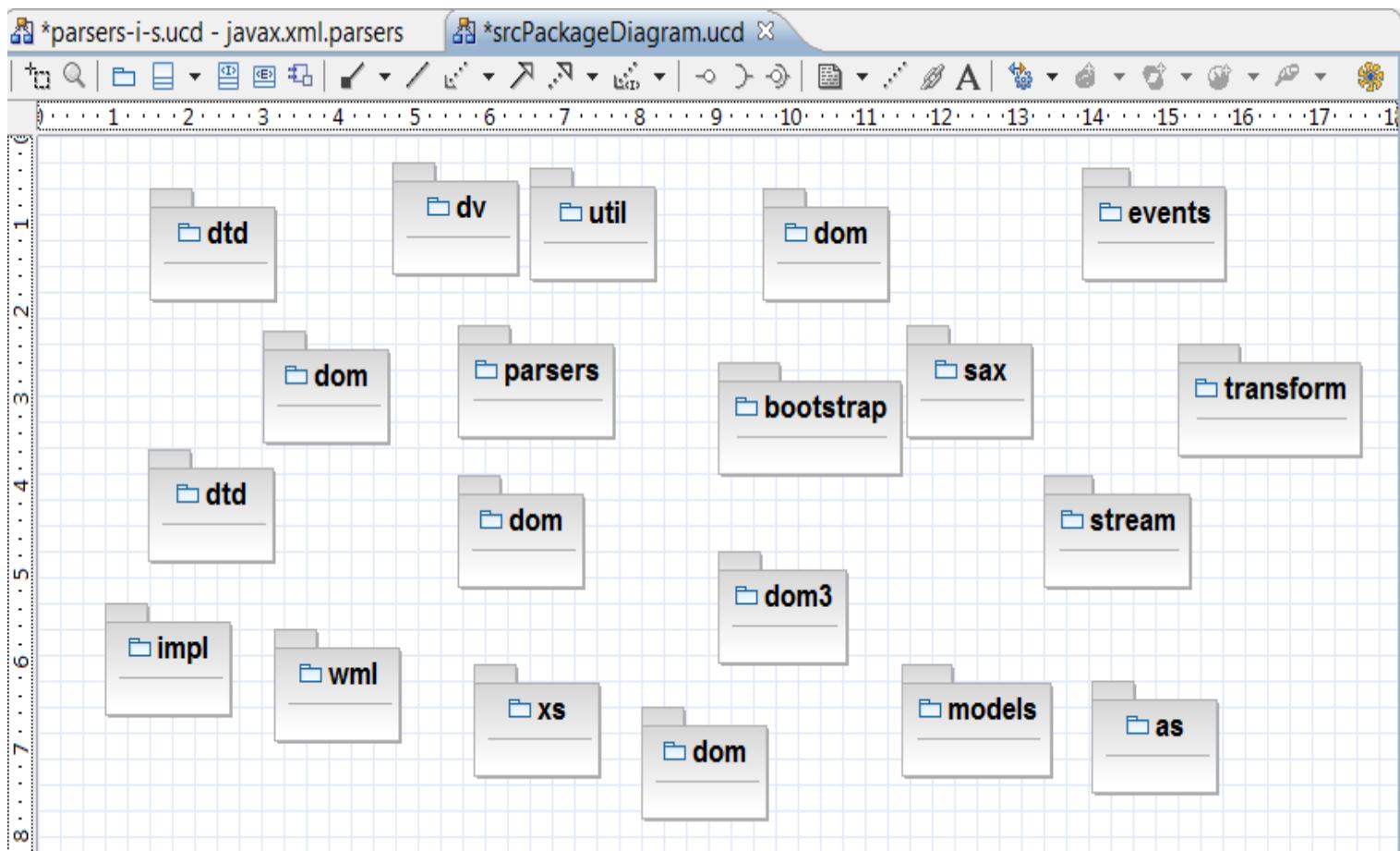


If you look at the model then you will notice that only the selected package classifiers and connectors have been saved in the model.



- select a package or a classifiers in the package explorer and drop them in a class diagram. You can create for example an empty package diagram if you select **src > New UML Diagram > Package Diagram** and drop inside existing packages from the Package Explorer.





Please note that the dependencies detection between packages is only activated if you reverse your project at src level (*e.g. you can not see dependencies between packages if you don't use src level reverse*).

1.3 Merge existing model with modified java code

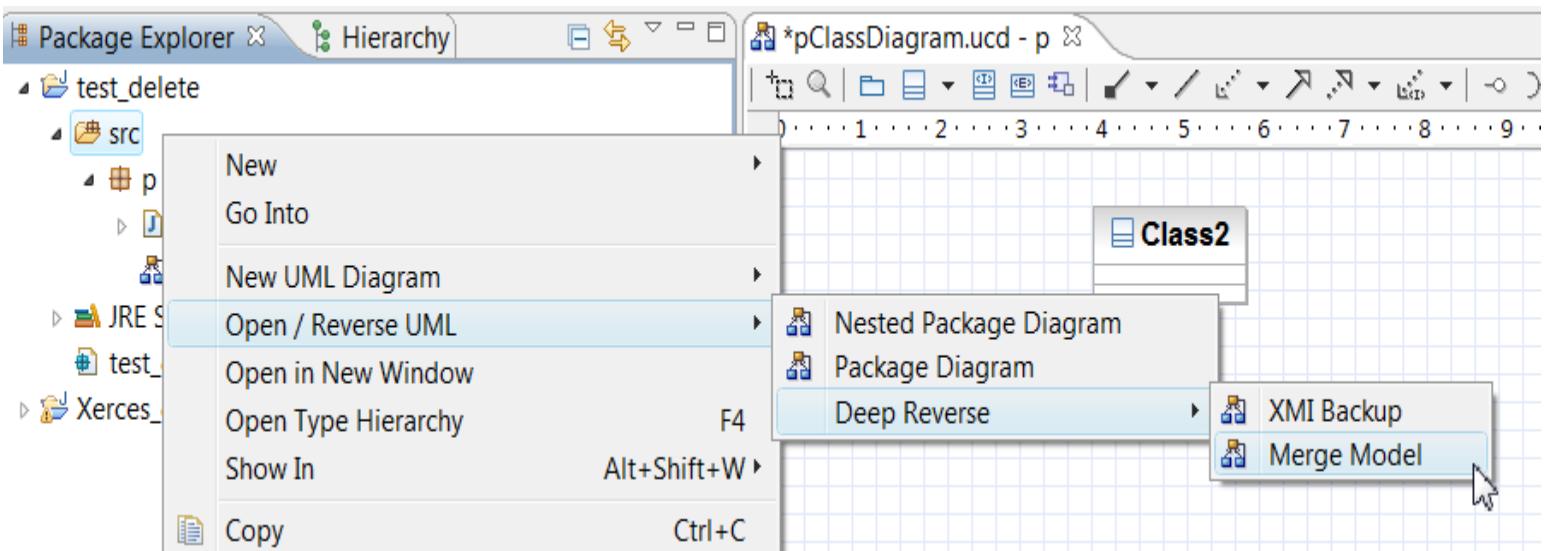
The merge option is an important concept if the java project is evolving with the creation of new UML diagrams.

The principle is to reverse the full project at the beginning of the modeling stage, then to create diagrams and close EclipseUML.

After few weeks of coding iteration stage is needed.

At this stage the model merge will add all detected new packages, classifiers and connectors to the existing model.

The merge option is available if you select the **src > Open / Reverse UML > Deep Reverse > Merge Model**.



The merge mechanism is working at XMI level and is respecting the following rules:

- Existing classifiers and connectors are never erased from the model.
- Each new package or classifiers are added to the model.
- Existing Classifiers which doesn't exist anymore in the java code are automatically transform into model classifiers and not deleted from diagrams or from the model.
- Existing classifiers which are refactored are also immediately renamed/moved in the model and in each UML diagram when you first time open the editor.
- If attributes or methods are added or deleted or rename then it is immediately mapped with the model (*e.g. it means that deleted attributes or methods from the java code are deleted from the model at this stage*).

Please note that if you copy and paste diagrams into another project then diagrams will not anymore be synchronized with Java or the model.

If you move diagrams inside your project then diagrams will keep the same properties and will always be synchronized.

2. JPA Annotations

JPA annotations are automatically reversed in both live code/model/UML Editor synchronization. You can use deep reverse menus (e.g. XMI backup or Merge Model).

For example if you reverse the following code:

```
package p;
import javax.persistence.Entity;
import javax.persistence.Id;

@Entity(name="name")
public class JPAEntity {

@Id
private long id;
```

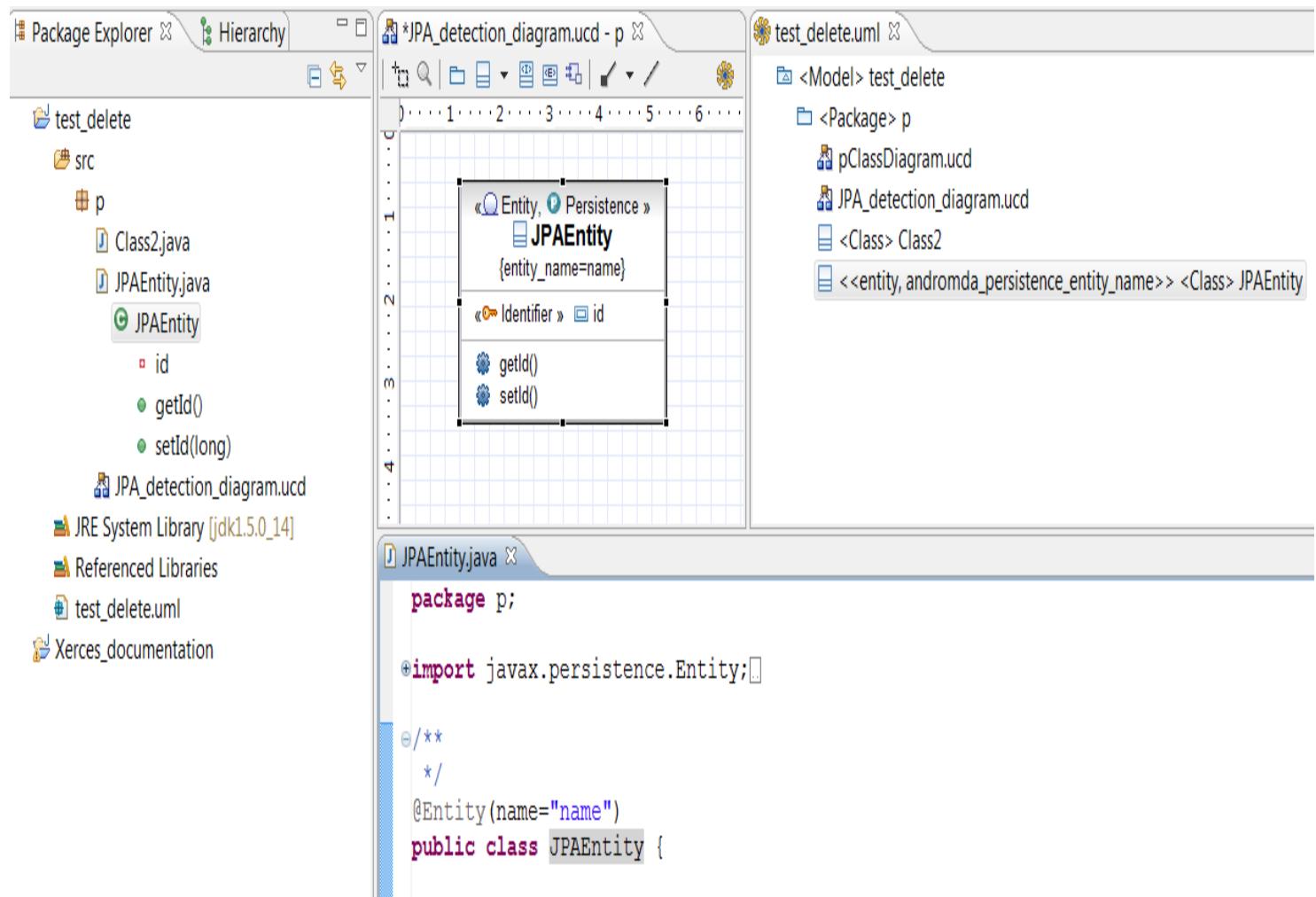
```

public long getId()
{
    return id;
}

public void setId(long id ){
    this.id = id;
}
}

```

You will get the following in your class diagram:



3. Reverse a database

The use of the Dali plugin with EclipseUML is required.

The concept is: Dali plugin connect the database and reverse it into Java code including annotations.

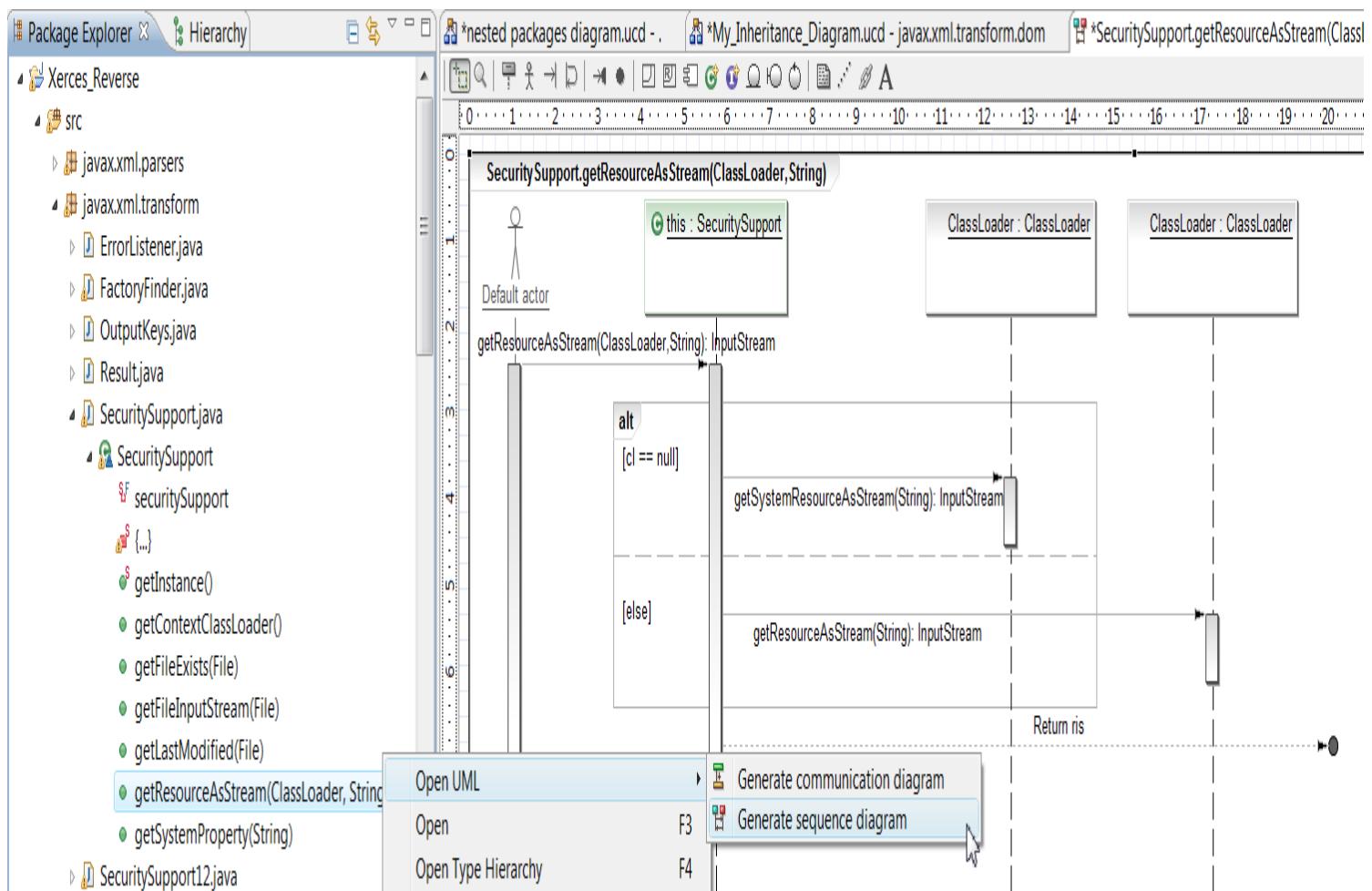
EclipseUML then reverse these annotations from the code into the UML Editor and the Model

More information at: [Persistence Development topic](#)

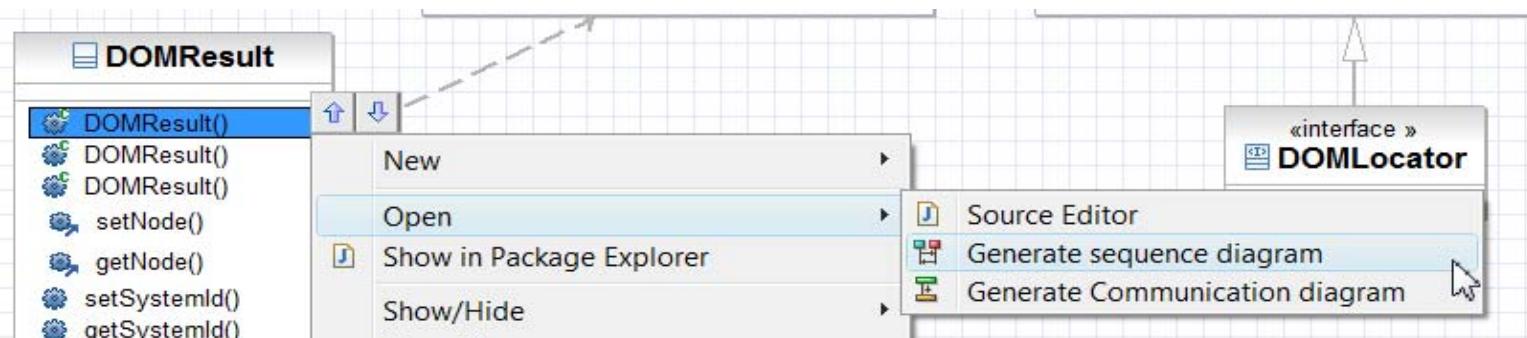
4. Sequence Diagram Reverse Engineering

You can reverse a method by selecting a method in:

- the Package Explorer



- inside the operation compartment of a class



Reverse Engineering Example

In this example we are going to reverse the full Xerces project java code information and map it to the model .uml file. We have decided for the purpose of this example to map all java information with the model. If you prefer just to have a view of the code and don't need to save all Java code project in your model then see the [Create a View of your Project section](#).

The example is composed by:

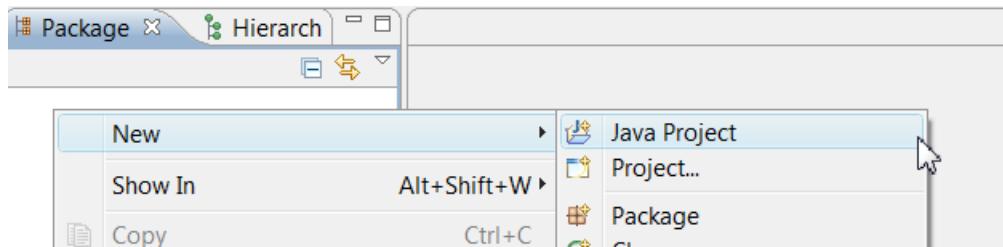
1. [Import your Xerces project inside Eclipse](#)
2. [Save all project information inside your model](#)
3. [View Nested Package View of your project](#)
4. [Reverse of a package](#)
5. [Reverse inheritances of Classes and Interfaces coming from different packages](#)
6. [Reverse Method and map this information to the model](#)
7. [Erase your java code and still use the model to navigate](#)

1. Import your Xerces project inside Eclipse

Create a Java project by selecting the Package Explorer > **New > Java Project**.

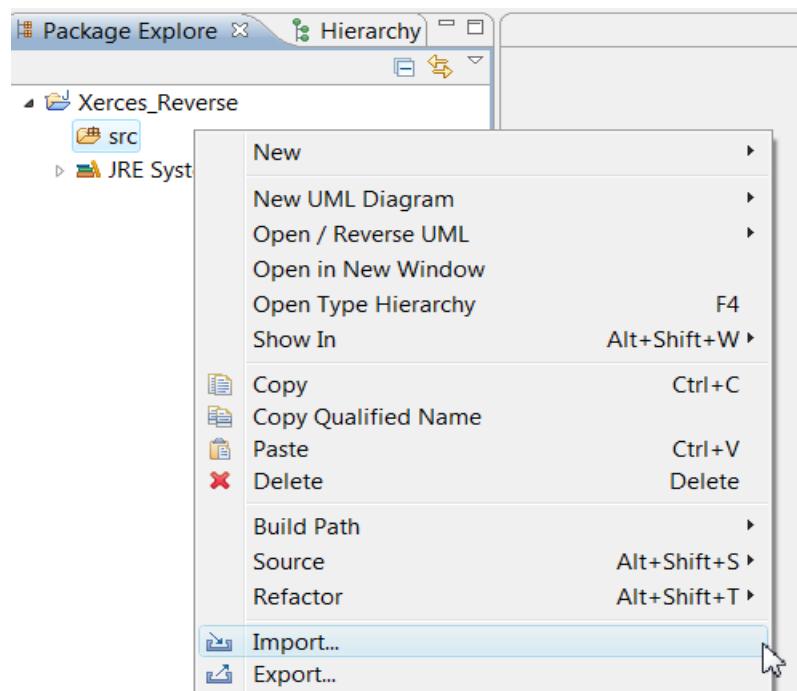
Enter the name of your project in the Project name field and select the Create separate folders as sources and class files option (e.g. create a src at the root of your project).

Click on the Finish button.

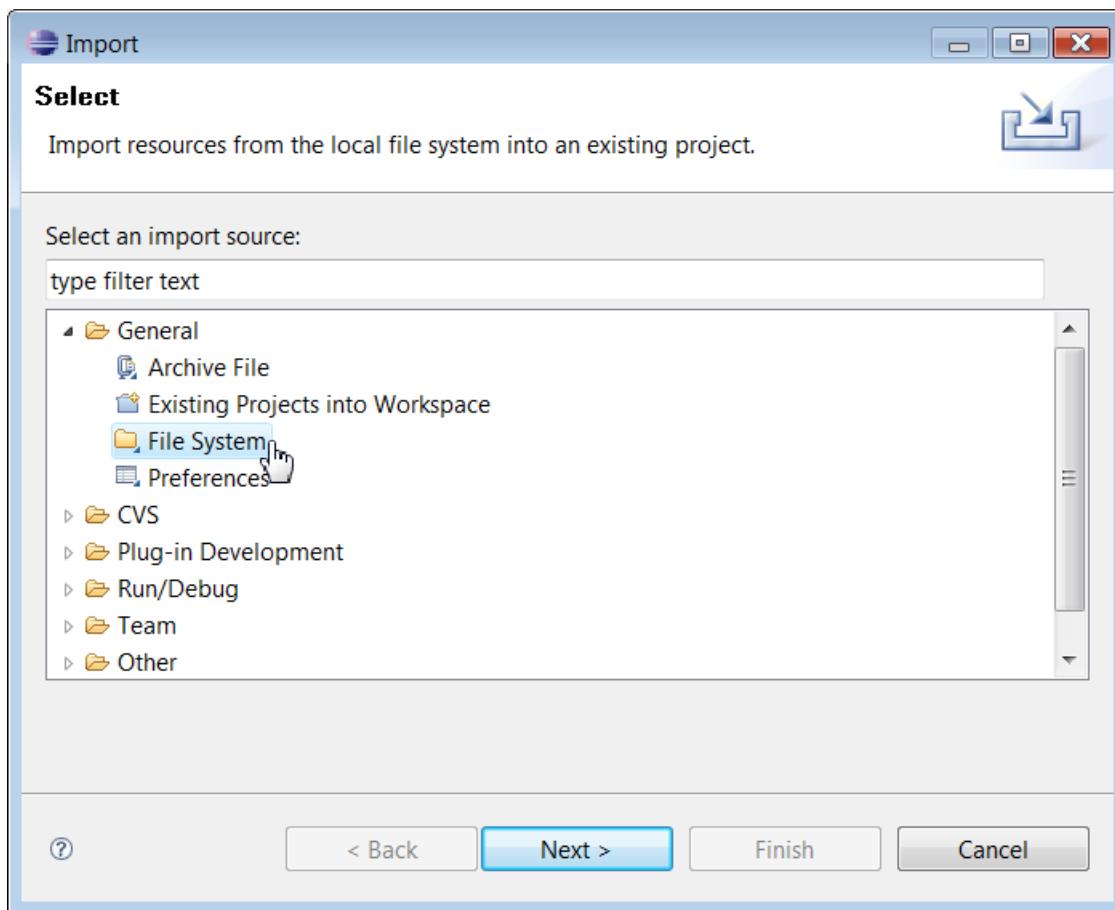


Your project structure has been created inside the Package Explorer.

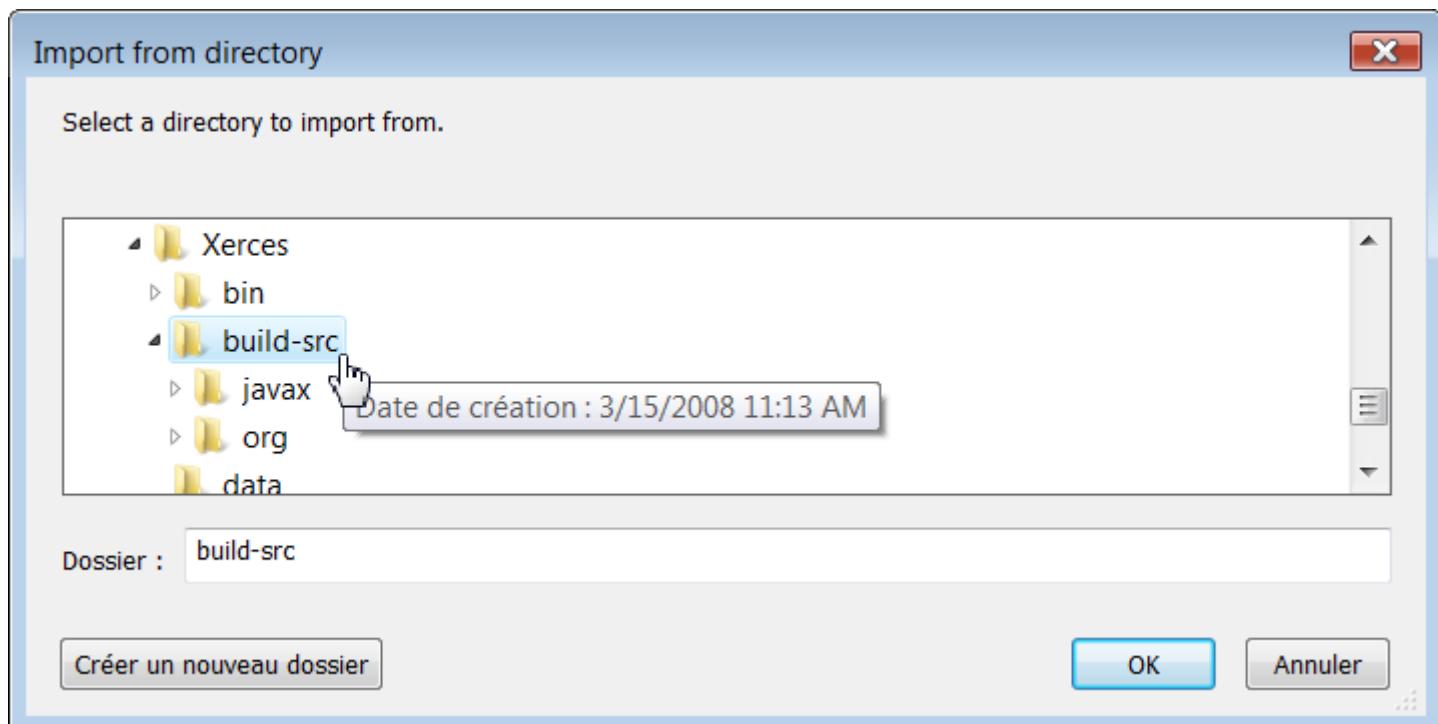
Click on the **src > Import...**



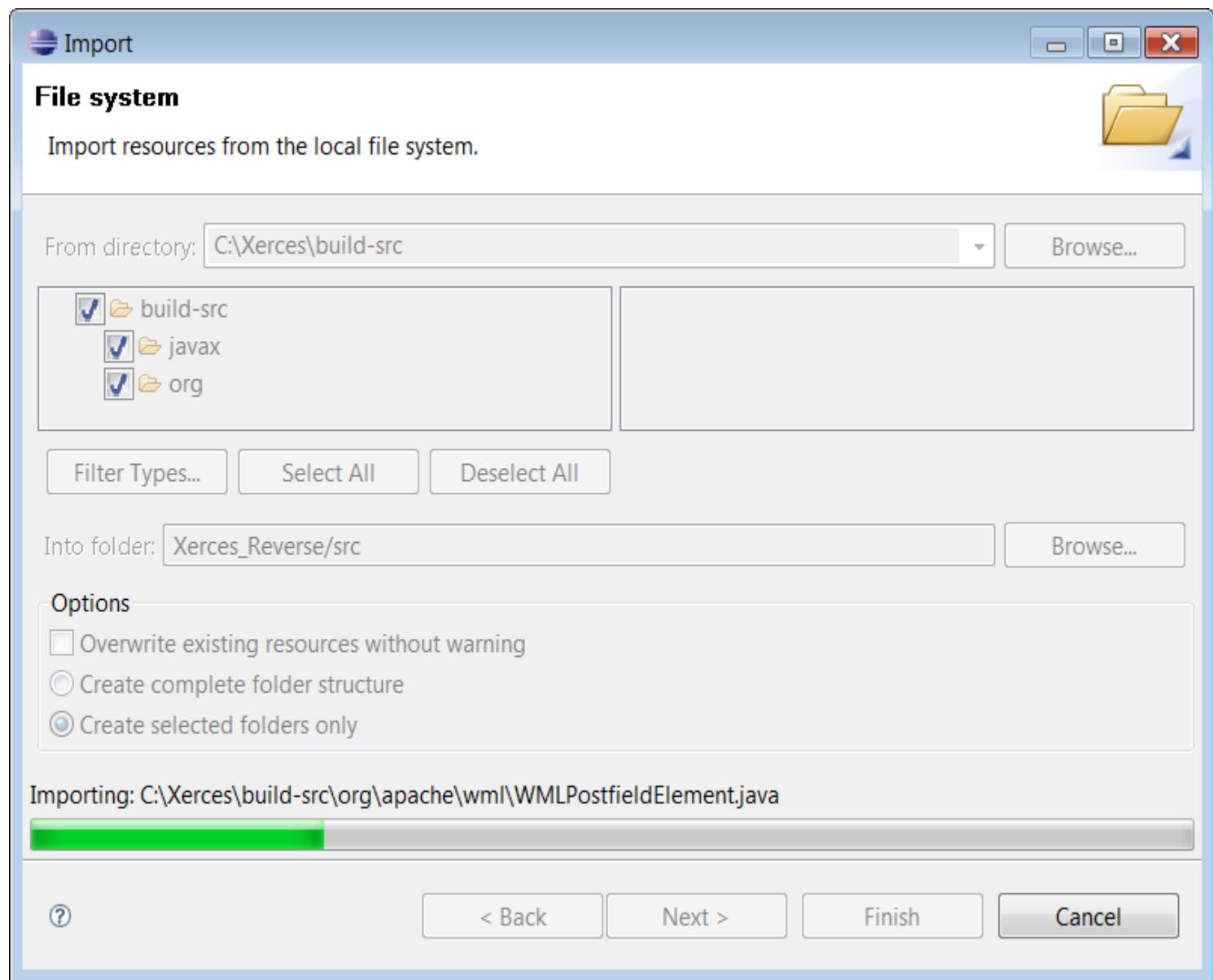
Select **General > File System**.



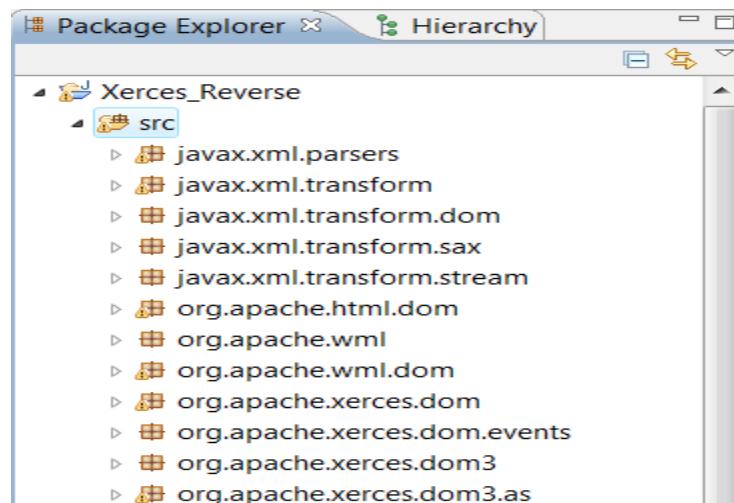
Browse your hard disk and select your project source code.
Click on **Xerces folder** and **select build-src**.



Select build-src folder and click on the Finish button.
Eclipse will immediately import your code inside your java project.

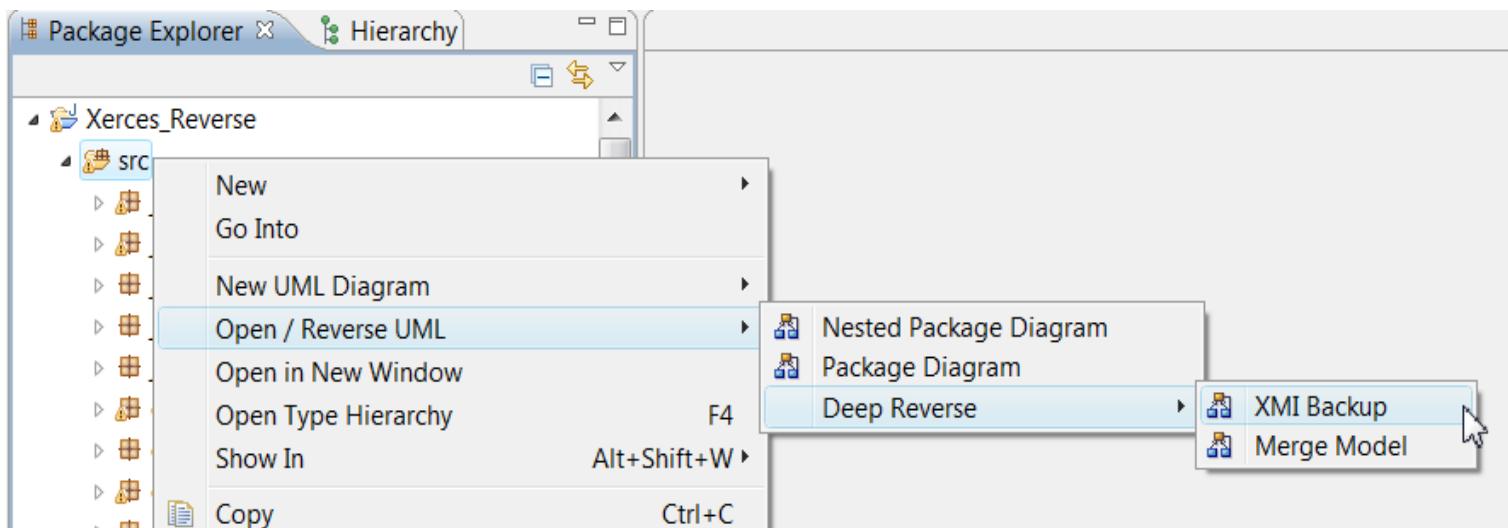


You can see the newly imported project in the package explorer.

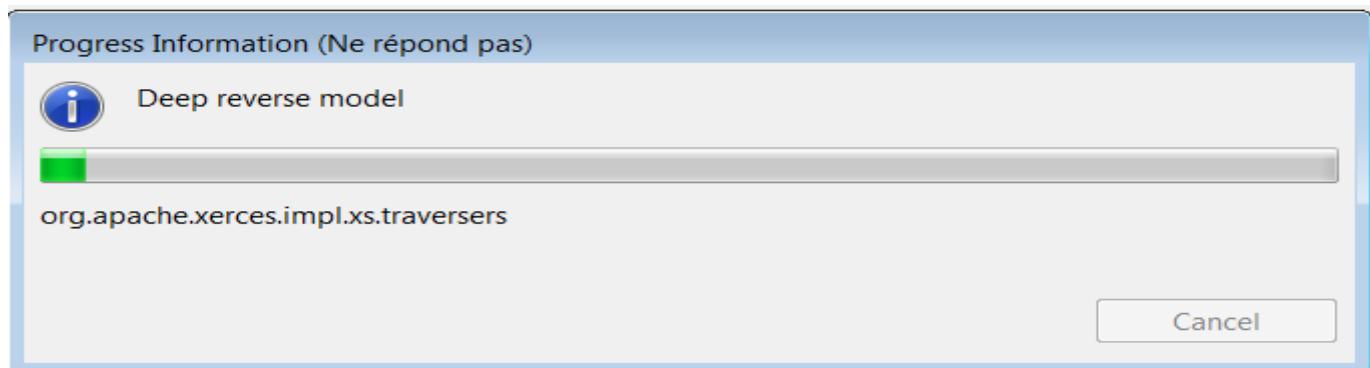


2. Save all project information inside your model

To save your entire structural project (*e.g. classifiers and connections but not business rules*) select in the Package Explorer **src > Open / Reverse UML > Deep Reverse > XMI Backup**.

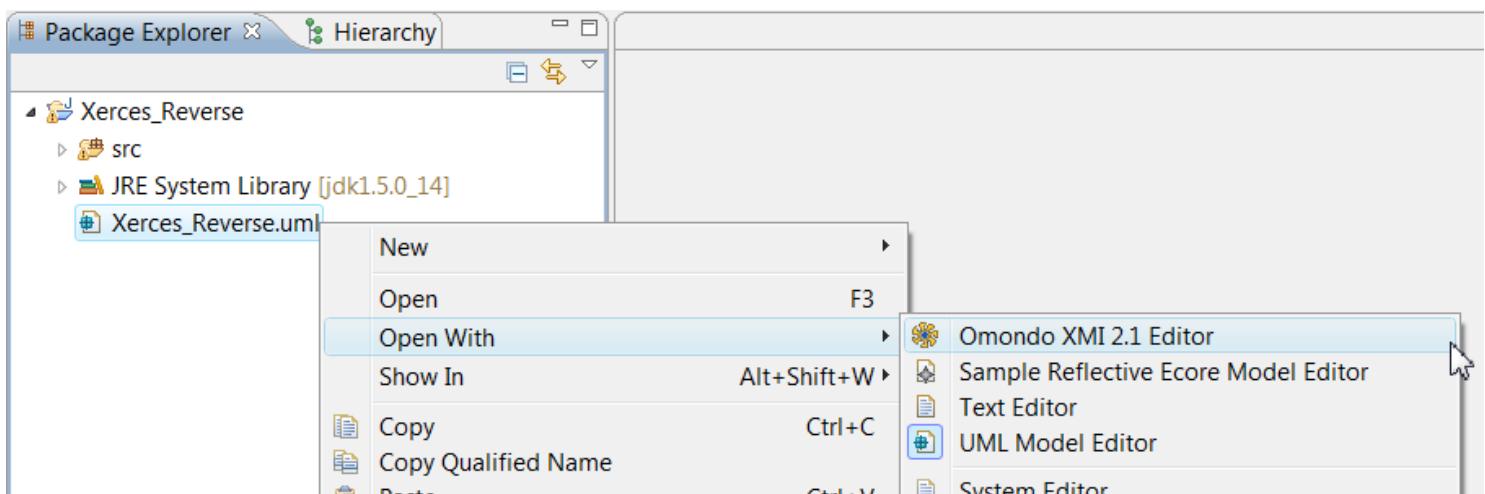


This heavy refactoring process will take few minutes to complete.

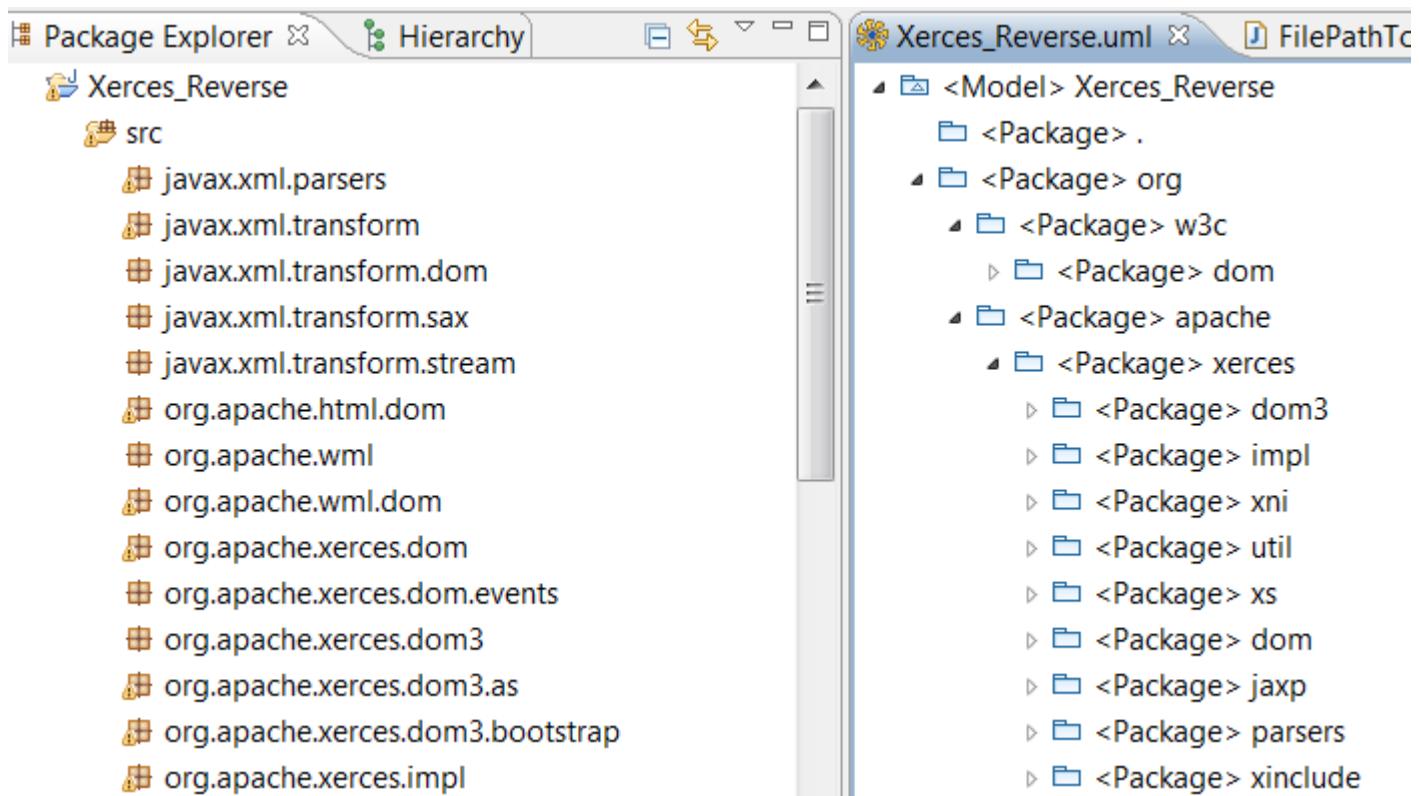


At the end of the process the Model has been created including all Java information.

To visualize the model click on the **Xerces_Reverse.uml** file > **Open With > Omundo XMI 2.1 Editor**.

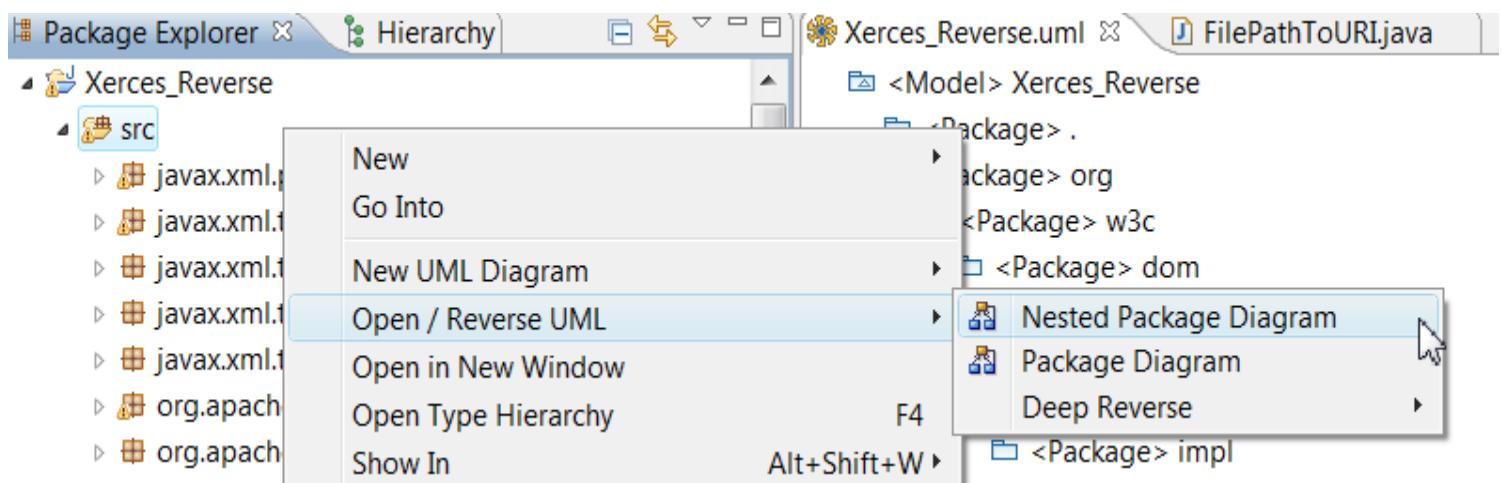


Here is the newly created model.

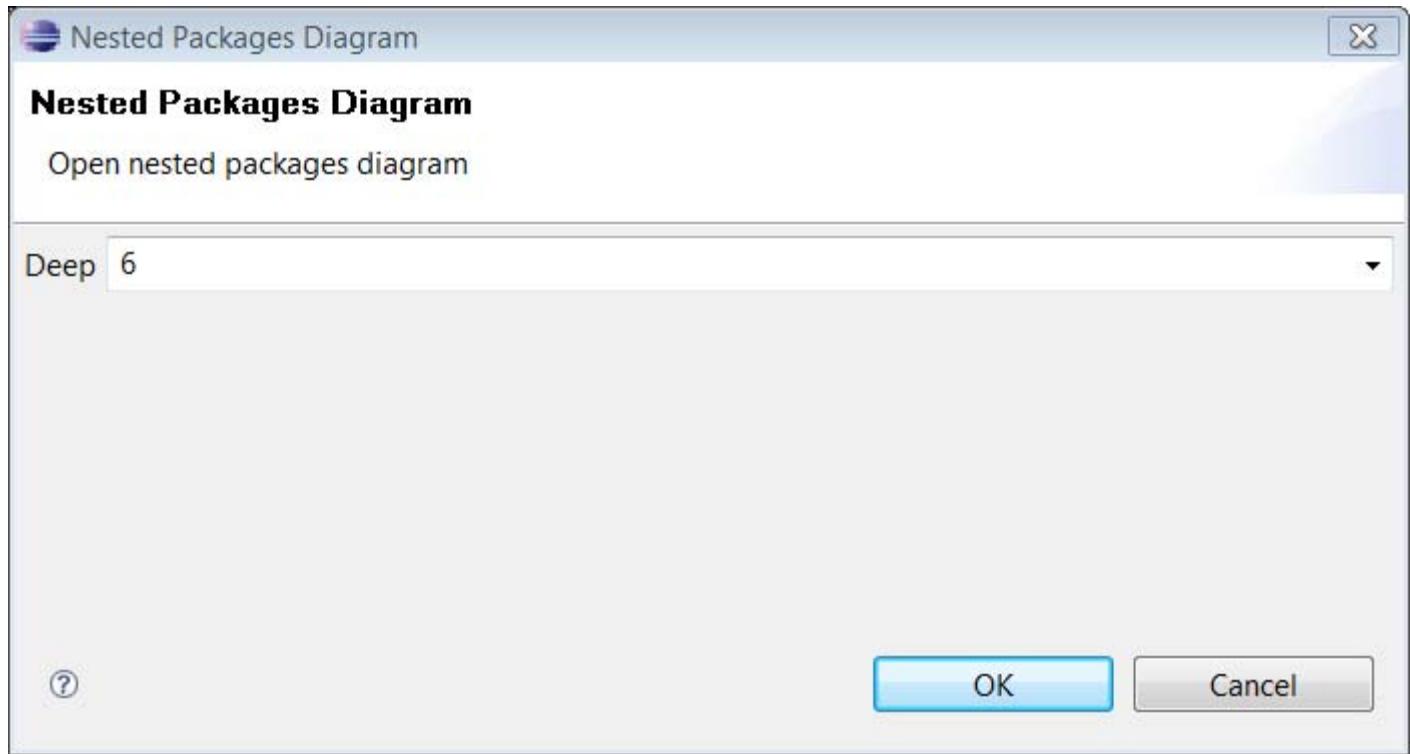


3. Nested Package View of your project.

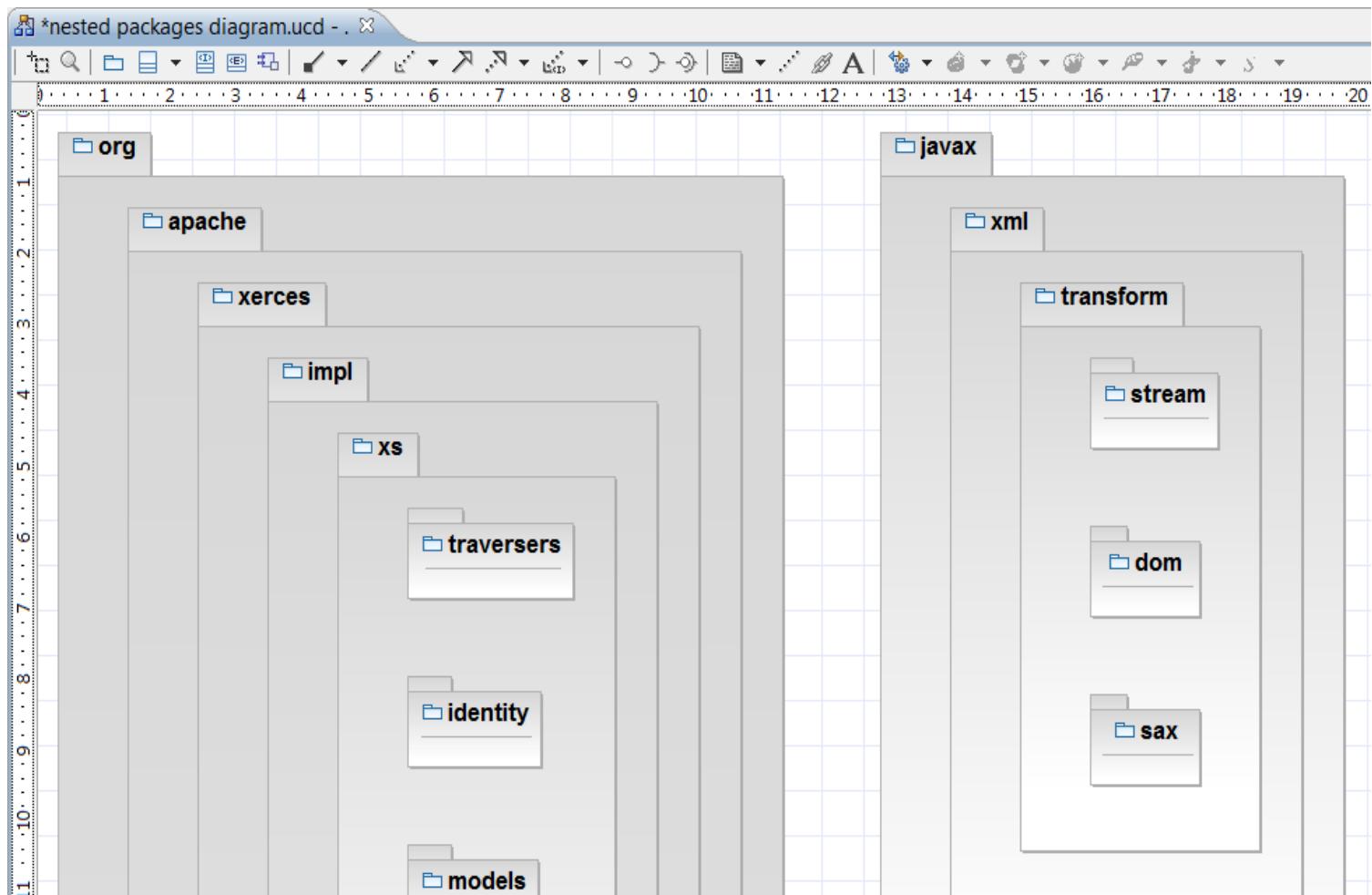
To get a nested package diagram view of your project you need to click on the **src** > **Open / Reverse UML** > **Nested Package Diagram**.



To select the level of depth click on the top down arrow or write a number in the Deep field.



Here is the nested package giving a structural view of the Xerces project.



You can see more information on reverse options at: http://www.ejb3.org/jar_file_reverse/jar_file_reverse.html

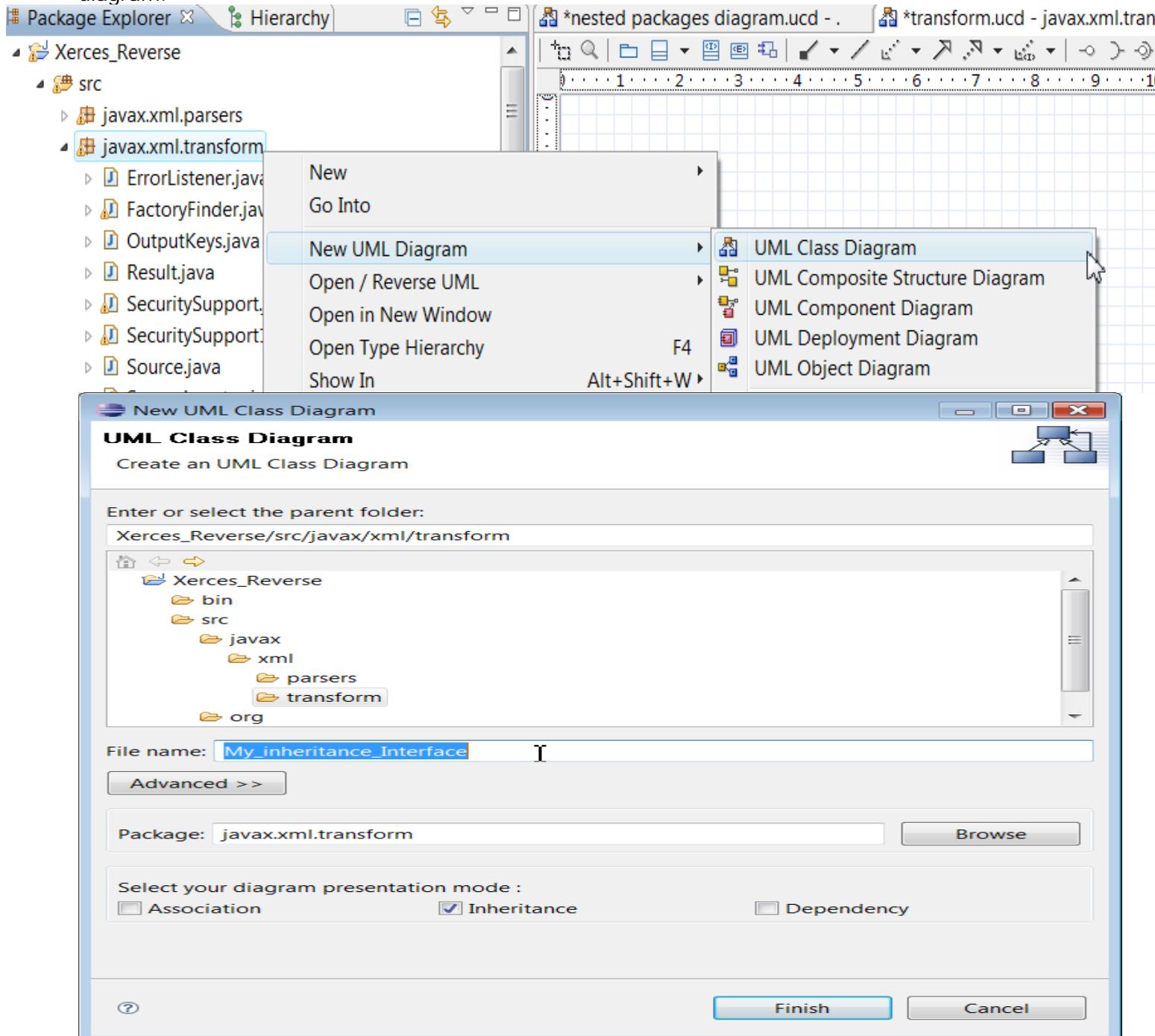
4. Reverse of a package.

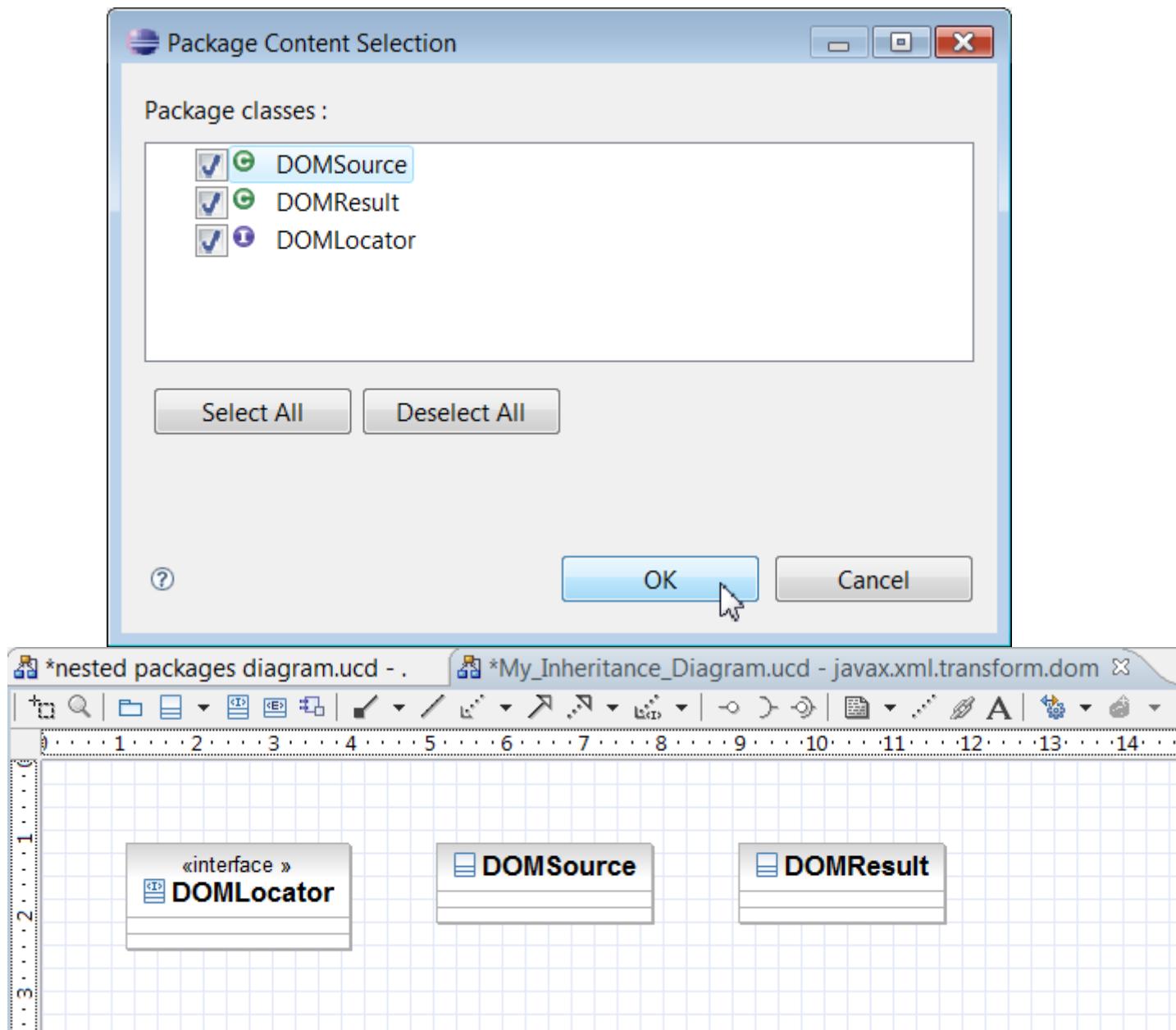
You can reverse multiple packages classifiers by reversing a package and adding classifiers from both the [XMI Editor](#) or the Package Explorer. To reverse a package you can either double click on a package inside a package diagram or select a package in the package explorer. You can decide two main options:

- either creates a class diagram with a name that you manually enter.

Click for example on `javax.xml.transform.dom` package > **New UML Diagram > UML Class**

Diagram. The `My_inheritance_Interface` class diagram has been created. Drag and drop from the Package Explorer or Select in the Package Content Selection Classifiers to be inserted inside your diagram.

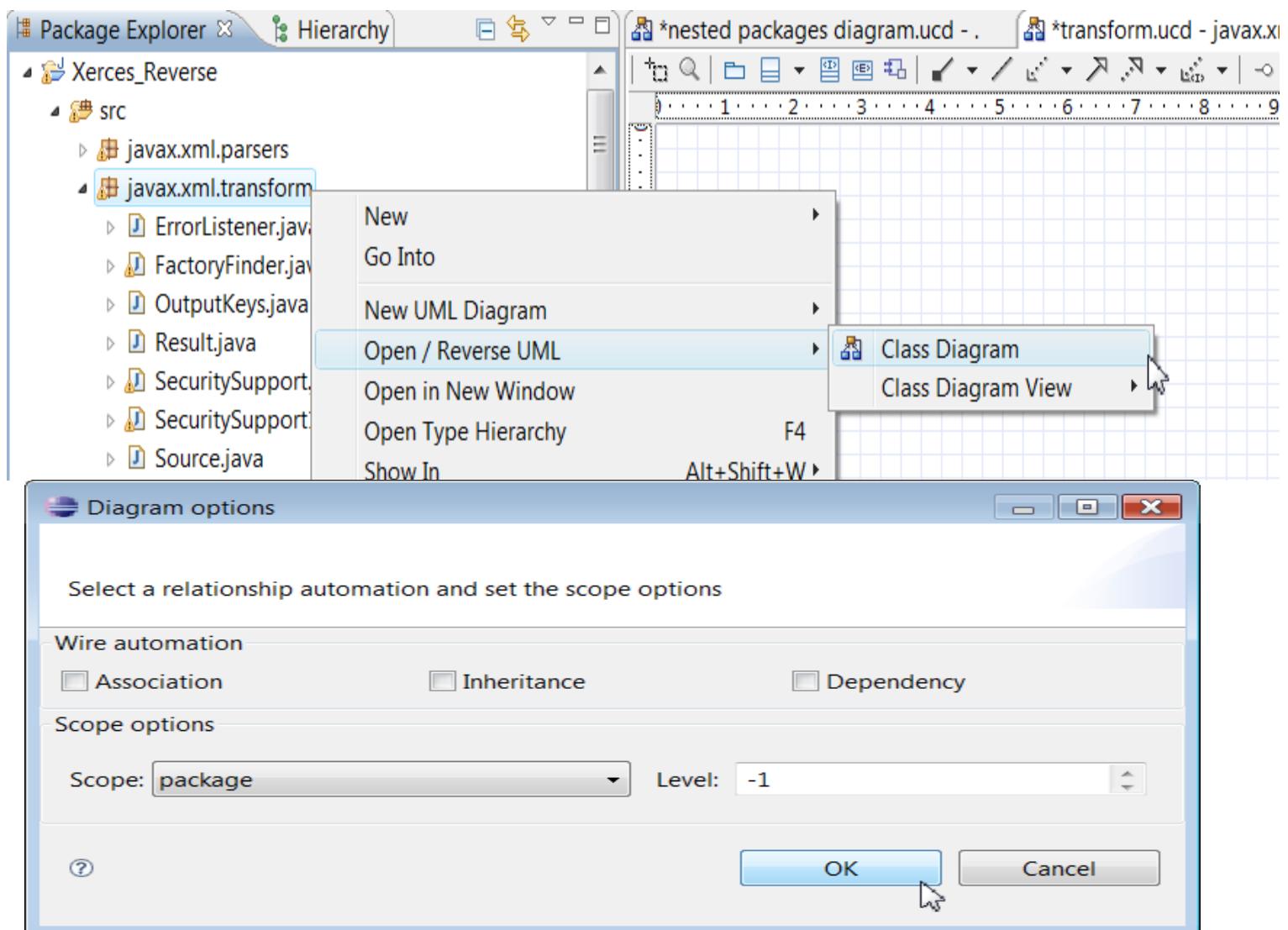




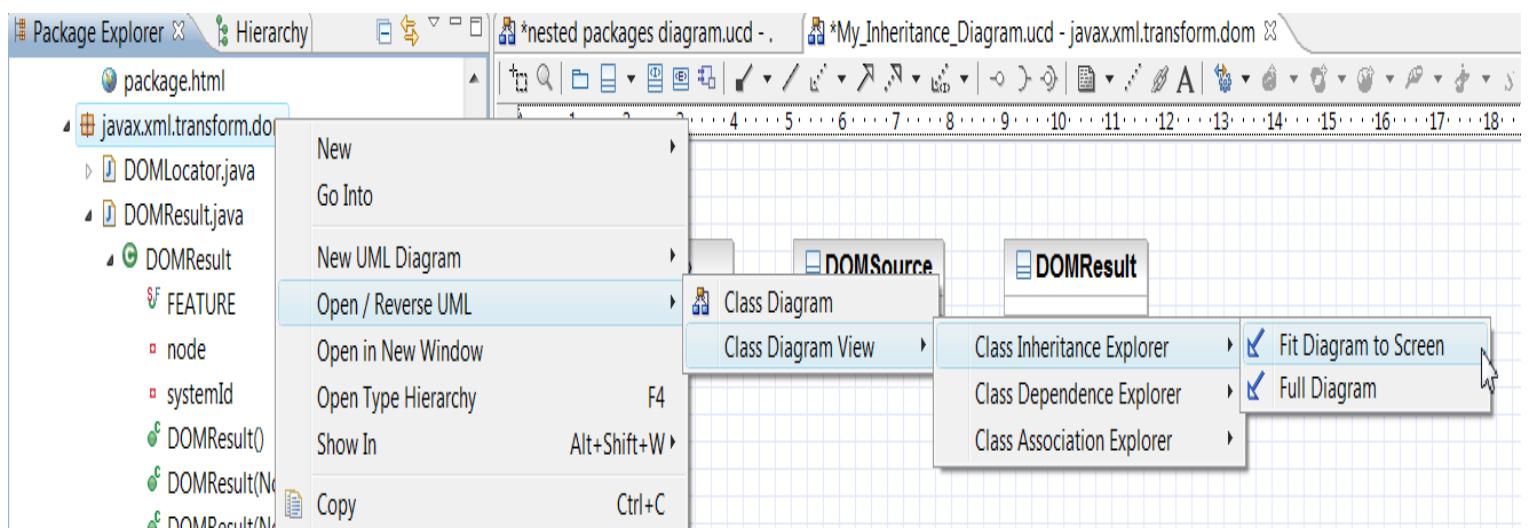
- Or use a given name reverse class diagram.

The following options are then available:

- Open/Reverse UML
 - **Class Diagram** allows to only see classifiers if you don't select any option. This is a nice feature if you want to immediately get a diagram and then navigate manually creating your connector links. The advantage of this option is that the reverse is very fast.

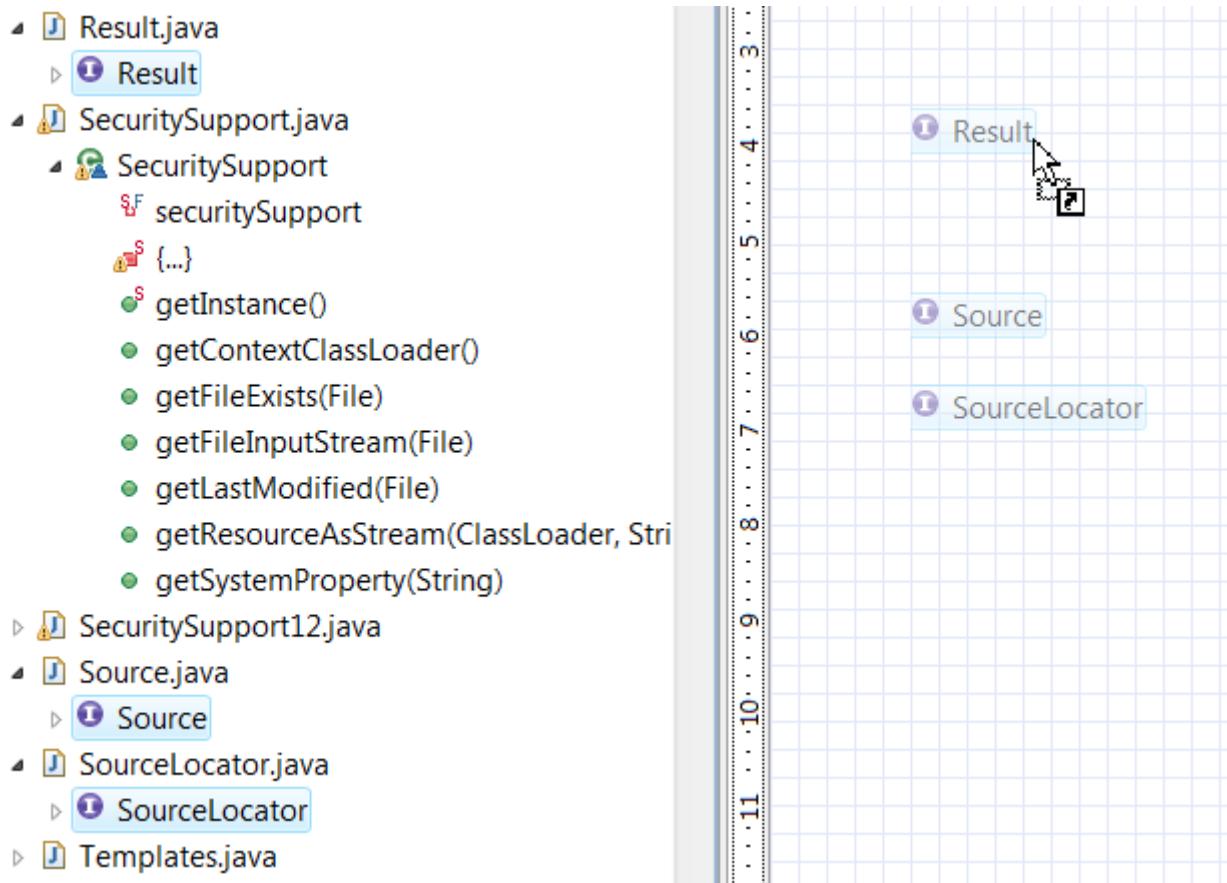


- **Class Diagram > Class Inheritance Explorer > Fit Diagram to Screen or Full Diagram.** This option allows to immediately getting a view of your diagram. You can select to see compartments (e.g. Full Diagram) or not (e.g. Fit Diagram to Screen) in order to have a smaller diagram.



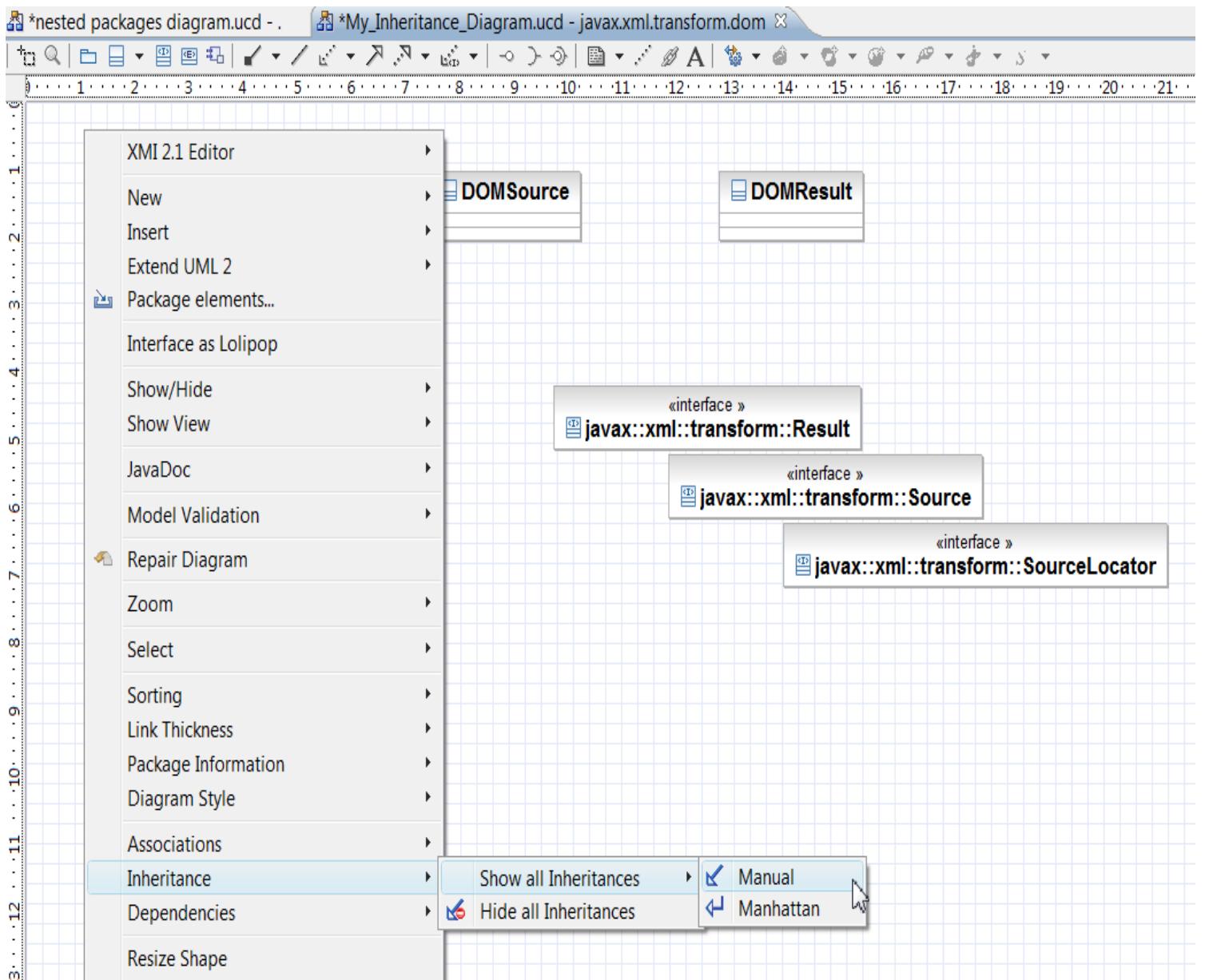
5. See Classes and Interfaces coming from different package inheritances.

You can select a group of classifiers and drag and drop them inside an existing diagram. Select a group of classifiers with the mouse in the Package Explorer and drop them inside the diagram.

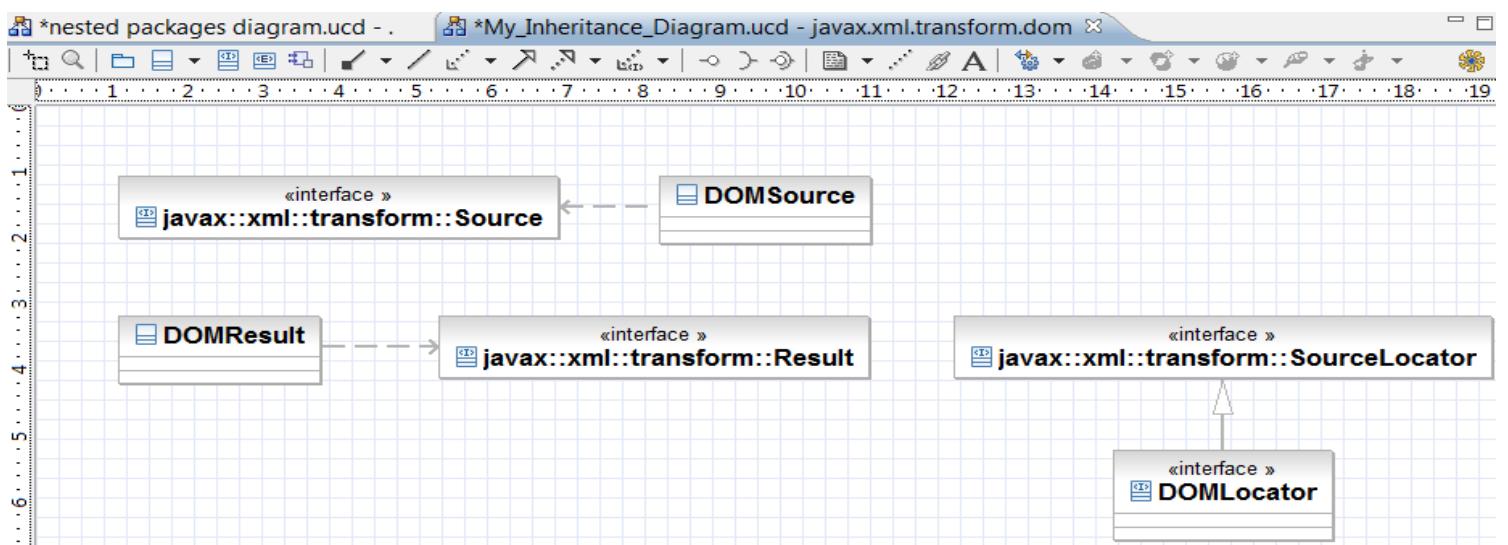


The new interfaces will immediately be added in your diagram.

Now click on the class diagram contextual menu > **Inheritance** > **Show all Inheritance** > **Manual**.



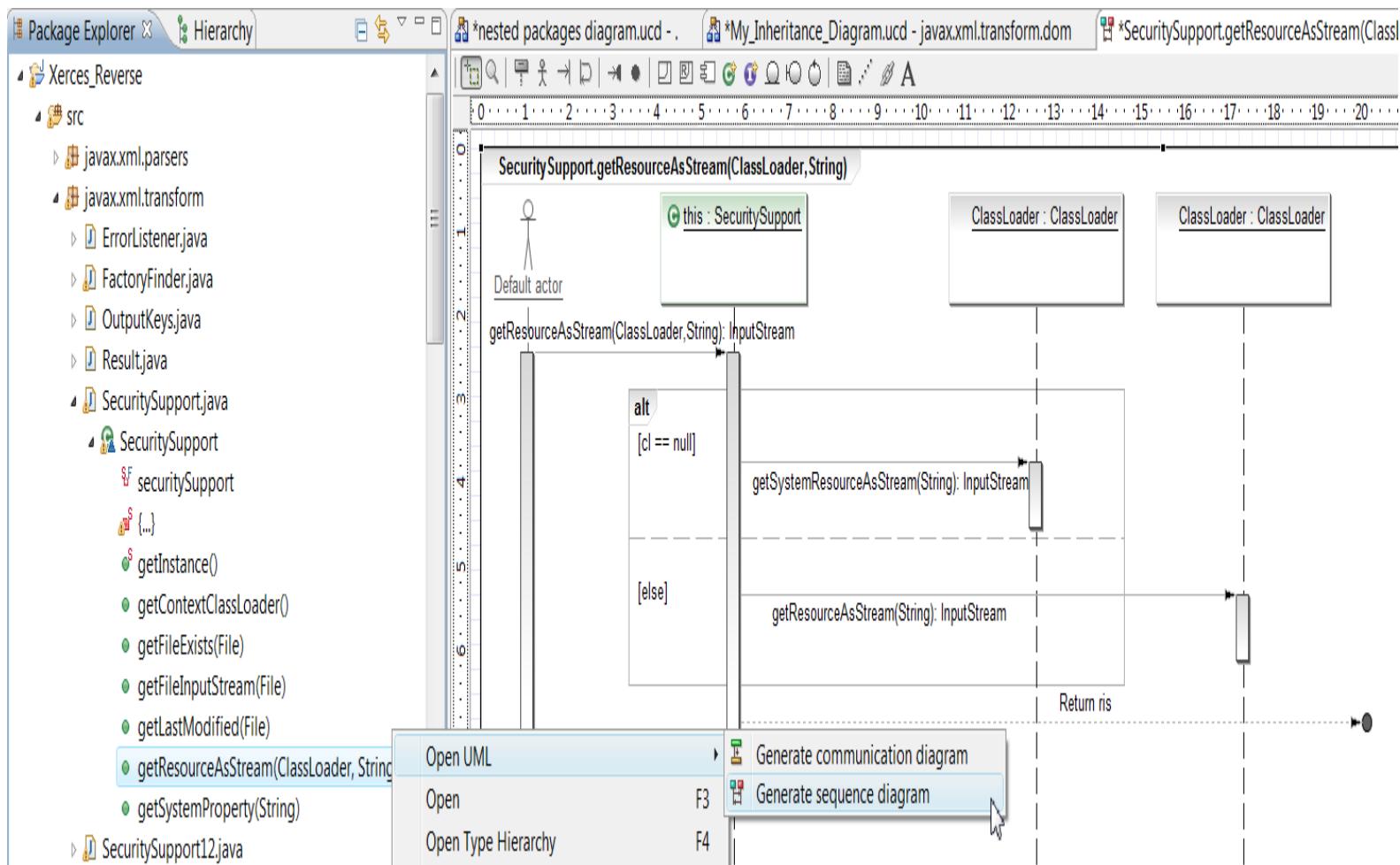
Here is the diagram you get after using the class diagram contextual menu > **Arrange Diagram** > **Arrange All**.



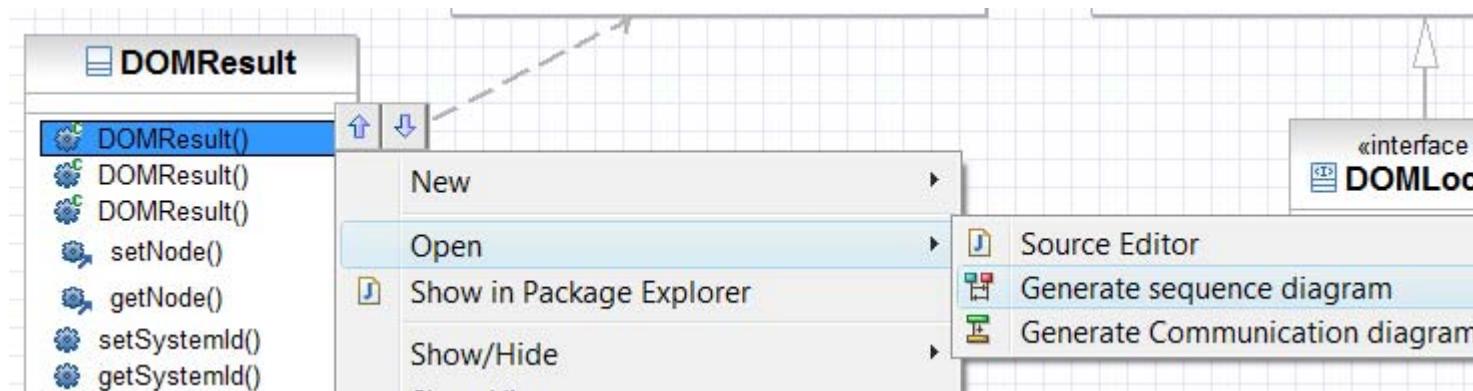
6. Reverse Method and map this information into the model

You can reverse a method by selecting a method in:

- the Package Explorer



- inside the operation compartment of a class

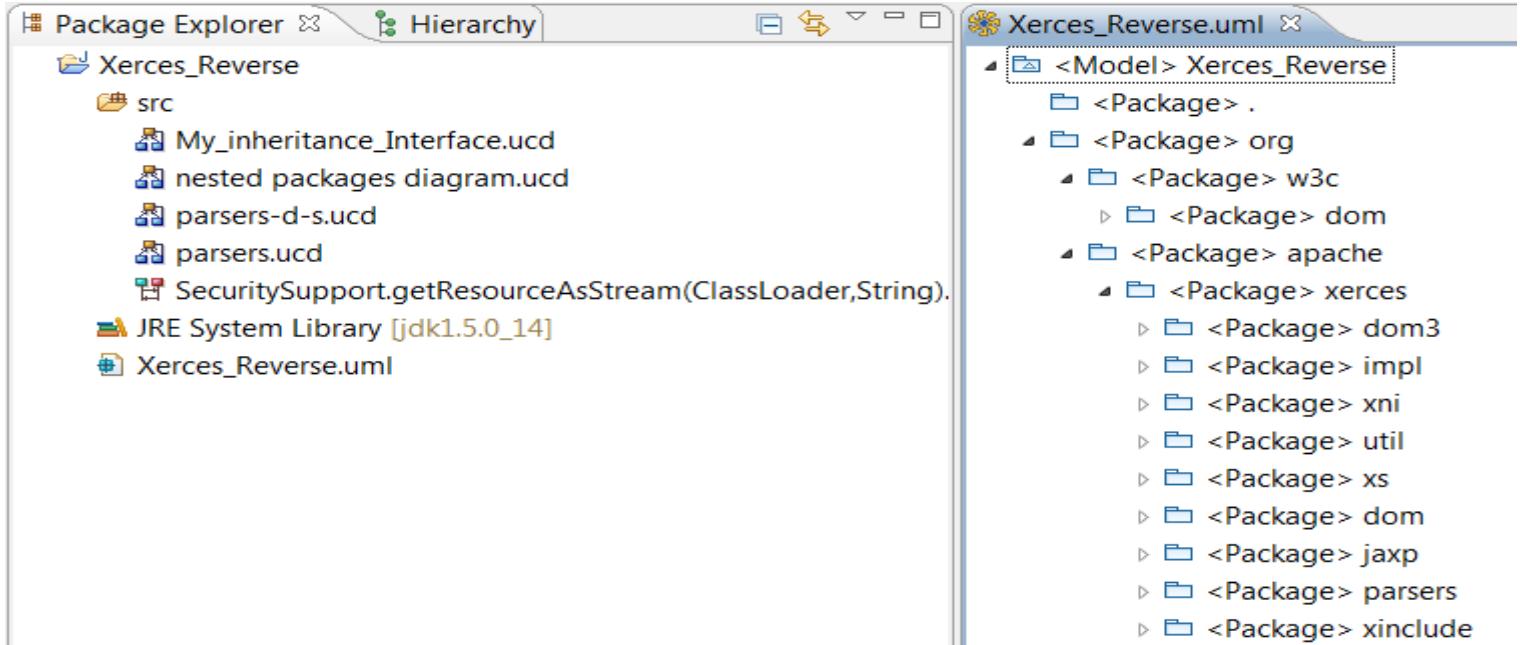


7. Erase your java code manually and only use the model to navigate

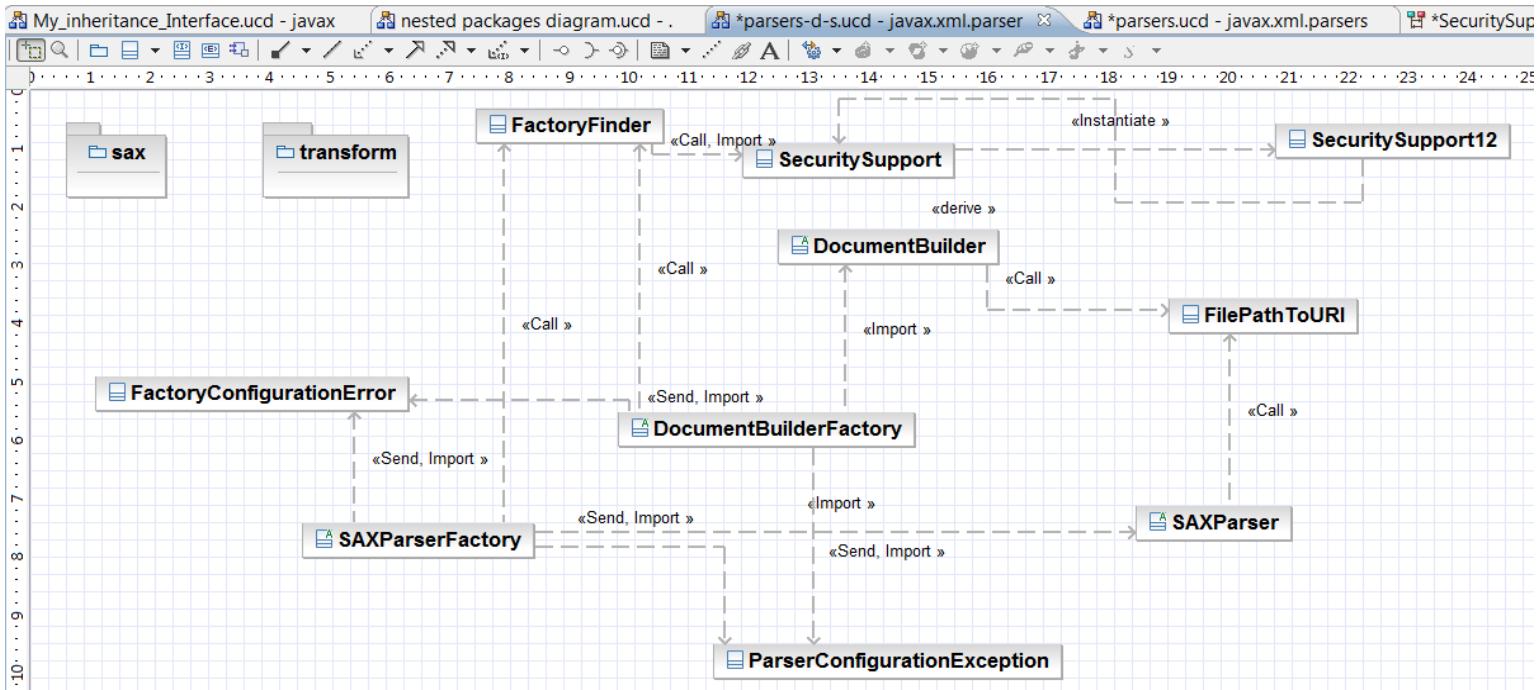
Once the XMI Backup has been selected you can delete you all java code and still be able to navigate inside your model.

For example let erase all Xerces code from the package explorer and paste and copy your existing diagram at the src root.

You will have the following Package Explorer



You can now open any diagram previously created if you double click in the Package Explorer.



You can also create new diagram from the model and don't anymore need the java code.

For more information see our tutorial at: http://www.forum-omondo.com/documentation_eclipseuml_2008/eclipseuml2008_dynamic_navigation.html#4. Navigate inside the Xerces

Java Reverse Engineering

Reverse engineering is a process to rebuild UML model from language implementation. In Java, some information can be retrieved easily by parsing Java source code or by decoding class binary file structure, such as:

- Java type class/interface
- Inheritance
- Attributes

Others need a complex code semantic analysis, such as:

- Getter/setter methods
- Cardinality of association
- Element type in a collection
- Inverse association resolution
- Qualified association

Precisely, it is necessary to analyze method implementation content to capture them. EclipseUML Personal/Professional/Studio Edition provides a very powerful reverse engineering engine to meet these requirements based on the byte-code analysis. Additionally, it provides following features:

- Model attribute detection
- Alive dependence detection between classes and packages
- Customizable dependence definition

EclipseUML Personal/Professional/Studio Edition uses the reverse engineering engine in two ways:

- UML model reverse engineering
- Dependency detection

Getter/setter method recognition

UML Reverse engineering detects automatically the getter/setter method of an attribute. It takes into account the prefix and suffix preferences of [JDT](#). The name without prefix and suffix will be used as model name, named as property.

Concrete methods

If methods have the implementations, the UML reverse engineering analyzes the code to make sure that the getter method returns this attribute value and the setter changes this attribute value.

For example, with the prefix setting of "f", in the following Java class that contains an attribute name = fCompany, the property name will be "company"

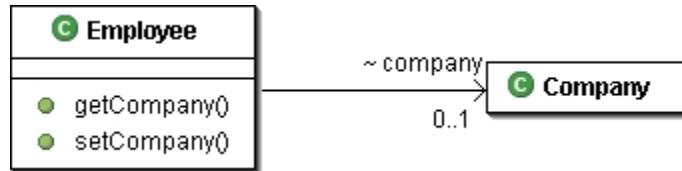
```
public class Employee
{
```

```

private Company fCompany;
public Company getCompany( )
{
    return fCompany;
}
public void setCompany(Company company)
{
    fCompany = company;
}
public class Company
{
    ...
}

```

After the reverse engineering process, we find UML role name = company, instead of fCompany.



```

public class Employee
{
    /**
     * @uml property=company associationEnd={multiplicity={(0 1)} ordering=ordered
     * elementType=company.Company}
     */
    private Company fCompany;
    /**
     * @uml property=company
     */
    public Company getCompany( )
    {
        return fCompany;
    }

    /**
     * @uml property=company
     */
    public void setCompany(Company company)
    {
        fCompany = company;
    }
}

```

Abstract methods

In case of the abstract method: class abstract method or interface method, there is no code to analyze. So the only solution is to analyze the method name and signature according to JDT definition. The setter method is ignored if a getter method is missing in the class/interface.

Cardinality detection

EclipseUML Reverse Engineering Engine is capable to detect two groups of associations:

- Cardinality 0..1 or 1, which correspond to simple reference
- Cardinality 0...*, which correspond to a container reference

To classify an attribute type to one of the two groups; the key issue is container detection. In Java, there are two categories of containers: `java.util.Collection` and `java.util.Map`. EclipseUML reverse engineering implements a mechanism to recognize not only the interface `java.util.Collection` and `java.util.Map`, but also their subtypes or implementation classes such as `java.util.List`, `java.util.Vector`, `java.util.Hashtable`.

Cardinality 0...1 or 1

For a simple reference, if the attribute is always initialised, the cardinality will be 1. Otherwise, it will be 0..1.

Examples of cardinality 0..1

Before Reverse Engineering	After Reverse Engineering
<pre>public class Employee extends Person { private Company company; public Company() { } }</pre>	<pre>public class Employee extends Person { /** * @uml.property = company * associationEnd={multiplicity={(0 1)}} */ private Company company; public Company() { } }</pre>

Example of cardinality 0..1

Before Reverse Engineering	After Reverse Engineering
<pre>public class Employee extends Person { private Company company; public Company(Company company) { this.company = company; } }</pre>	<pre>public class Employee extends Person { /** * @uml.property = company * @uml.associationEnd = {multiplicity={(1 1)}} */ private Company company; public Company(Company company) { this.company = company; } }</pre>

Cardinality 0...*

A container has always the cardinality 0..*.

Example of collection

Before Reverse Engineering	After Reverse Engineering
----------------------------	---------------------------

<pre> public class Company { private Collection employees = new ArrayList(); ... } </pre>	<pre> public class Company { /** * @uml.property = employees * associationEnd={multiplicity={(0 -1)} * inverse={company:model.Employee}} */ private Collection employees = new ArrayList(); ... } </pre>
---	---

Collection element type detection

EclipseUML Reverse engineering engine captures the element type in a collection by analysing content accessing code. For example, the argument of collection method add() may indicate the element type:

```

public class Company
{
    private List employees = new ArrayList();

    public Company(List employees)
    {
        this.employees = employees;
    }

    public void addEmployee(Employee employee)
    {
        employees.add(employee);
    }

    public Collection getEmployees()
    {

```

```

        return employees;
    }

}

```

The idea is very simple. But the real use cases are much more complex. The same method may have different implementations.

Using local variable	<pre> public void addEmployee(Employee employee) { Collection localVariable = employees; localVariable.add(employee); } </pre>
Call getter	<pre> public void addEmployee(Employee employee) { getEmployees().add(employee); } </pre>
Using getter and local variable	<pre> public void addEmployee(Employee employee) { Collection localVariable = getEmployees(); LocalVariable.add(employee); } </pre>

Of course, these codes can be used outside of the class Company.

According to the access type of this attribute, we distinguish the analyze mechanism in two categories:

- Getter category
- Setter category

The first one allows the forward analyze, the analyzer can follow the execution order. The typical example is the iteration method (see below). It is much easier than [Setter category](#), which needs backward analyze mechanism.

1. Getter category

Each accessing method may have a specific way to capture the element type. So we classify all relevant methods as following groups according to the model capture mechanism:

- Add, Remove, Index and Test group

- Get group
- Iteration group

1.1 Add, Remove, Index and Test group

This group includes following methods:

Class	Method
java.util.Collection	boolean add(Object object)
java.util.Vector	void addElement(Object object)
java.util.Vector	void add(int index, Object object)
java.util.Vector	Object set(int index, Object object)
java.util.Vector	Object elementSet (Object object, int index)
java.util.Vector	void setElementAt (Object object, int index)
java.util.Vector	void insertElementAt (Object object, int index)

Class	Method
java.util.Collection	boolean remove(Object object)
java.util.Vector	boolean removeElement(Object object)
java.util.Vector	void removeElement (Object object)

Class	Method
java.util.List	int indexOf (Object object)
java.util.List	int lastIndexOf(Object object)
java.util.Vector	int lastIndexOf (Object object, int index)
java.util.Collection	boolean contains (Object object)

The capture mechanism of this group is simplest one comparing others. We just need identify the method call and capture argument type. For example:

```
public void removeEmployee(Employee employee)

{
    Collection localVariable = employees;
    localVariable.remove(employee);
}
```

or

```
public int getEmployeeIndex(Employee employee)
{
    return employees.indexOf(employee);
}
```

1.2 Get group

This group consists of following methods:

Class	Method
java.util.List	Object get(int index)
java.util.Vector	Object elementAt(int index)
java.util.Vector	Object firstElement()
java.util.Vector	Object lastElement()

This group is a little bit difficult than previous one since we need analyze the next statements to capture the element type in type casting or the operator `instanceof`. For example,

```
public Employee getEmployeeAt(int index)
{
    return (Employee) employees.get(index);
}
```

or

```
public boolean hasEmployeeAt(int index)
{
    Object element = employees.get(index);
    return (element instanceof Employee);
}
```

1.3 Iteration group

This group consists of following methods

Class	Method
java.util.Collection	Iterator iterator ()
java.util.List	Iterator listIterator ()
java.util.List	Iterator listIterator (int index)
java.util.Vector	Enumeration elements ()

This group is more difficult than previous one again since it is necessary to analyze the following cast and `instance of` statements only after element retrieve call such as `next()` or `nextElement()`. For example,

```
Iterator iterator = company.getEmployees().iterator();
while (iterator.hasNext())
{
    Employee employee = (Employee) iterator.next();
}
```

```

or
for (Iterator iterator = company.getEmployees().iterator(); iterator.hasNext())
{
    Employee employee = (Employee) iterator.next();
}

```

All casting and operator **instance of** must be ignored without calling `next()`. Otherwise, the result will be wrong:

```

Iterator iterator = company.getEmployees().iterator();
Address address = (Address) getAddress();
while (iterator.hasNext())
{
    Employee employee = (Employee) iterator.next();
}

```

2. Setter category

When the code assigns a filled container to this attribute, for example,

```

List employees = new ArrayList();

employees.add(new Employee());

Company company = new Company (employees);

```

or

```

List employees = new ArrayList();

employees.add(new Employee());

company.setEmployees(employees);

```

it is necessary to perform the semantic analyze following inverse execution order. EclipseUML Personal/Professional/Studio edition implements this sophistic mechanism. It is used only when the first category calls are missing.

Inverse association resolution

EclipseUML Reverse engineering uses following rules to resolve inverse associations:

1. If Class A has an association "b" of type B and Class B hasn't any association of type A, there isn't inverse association.
2. If Class A has an association "b" of type B and Class B has only one inverse association "a" of type A, we set up the two associations as inverse associations.
3. If Class A has an association "b" of type B and Class B has more than one association of type A (for example, "a1" and "a2"), we try to resolve this conflict by analysing the accessing of the two couple of attributes: "b" and "a1", "b" and "a2".
 - a. If there is only one method that accesses one couple of attributes directly or via getter/setter, this couple of attributes will be considered as inverse associations.
 - b. If there is more than one method that accesses both couples of attributes, we use collected statistic to resolve this conflict.

For example:

```
public class Project
{
    private HashMap teams = new HashMap ();
    public void addTeamMember(String name, Employee member) {
        Collection team = (Collection) teams.get(name);
        if (team == null) {
            team = new ArrayList();
            teams.put(name, team);
        }
        team.add(member);
        member.setProject(this);
    }
}
```

The attribute accesses are in bold. So the associations implemented by the attribute `teams` and `project` are inverse associations.

Qualified association detection

EclipseUML Reverse engineering engine can recognize the `java.util.Map` and its implementation classes such as `java.util.HashMap` and `java.util.Hashtable`. It can capture not only the key and value type, but also the element type in the associated value if it is a collection.

In the following illustration, we use the Company and Project class as example:

```
public class Company
{
    private Hashtable projects = new Hashtable ();
    public Company(Hashtable projects)
    {
        this.projects = projects;
    }
    public Hashtable getProjects()
    {
        return projects;
    }
    public void setProjects (Hashtable projects)
    {
        this.projects = projects;
    }
}
```

According the accessing type of this attribute, we distinguee the analyze mechanism in two categories:

1. Getter category
2. Setter category

The first one allows the forward analyze, the analyzer can follow the execution order. The typical example is the iteration method (see below). It is much easy than Setter category, which needs backward analyze mechanism.

1. Getter category

Each accessing method may have a specific way to capture the element type. So we classify all relevant methods as following groups according to the model capture mechanism:

- Put, Remove, Index and Test group
- Get group
- Iteration group

1.1 Put, Remove and Test group

This group includes following methods:

Class	Method	Recognition
java.util.Map	<code>Object put(Object key, Object value)</code>	<code>Key and value</code>

Class	Method	Recognition
java.util.Map	<code>Object remove(Object object)</code>	<code>Key</code>

Class	Method	Recognition
java.util.Map	<code>boolean containsKey (Object object)</code>	<code>Key</code>
java.util.Map	<code>boolean containsValue (Object object)</code>	<code>Value</code>
java.util.Hashtable	<code>boolean contains (Object object)</code>	<code>Value</code>

The capture mechanism of this group is simplest one comparing others. We just need identify the method call and capture argument type. For example:

```
public void putProject(String name, Project project)
{
    projects.put(name, project);
}
```

1.2 Get group

This group consists of following methods:

Class	Method	Recognition
java.util.Map	<code>Object get(Object object)</code>	<code>Key and value</code>

This group is a little bit difficult than previous one since we need analyze the next statements to capture the element type in type casting or the operator `instance of`. For example,

```
public Project getProjectAt(String name)
```

```

{
    return (Project) projectsMap.get(name);
}

```

1.3 Iteration group

This group consists of following methods

Class	Method	Recognition
java.util.Map	Iterator values ()	Value
java.util.Map	java.util.Set keySet() followed by iterator()	Key
java.util.Map	Collection values() followed by iterator()	Value
java.util.Hashtable	Enumeration keys ()	Key
java.util.Hashtable	Enumeration elements ()	Value

This group is more difficult than previous one again since it is necessary to analyze the following cast and **instance of** statements only after element retrieve call such as `next()` or `next element()`. For example,

```

Iterator iterator = company.getProjects().values().iterator();
while (iterator.hasNext())
{
    Project project = (Project) iterator.next();
}

```

or

```

for (Iterator iterator = company.getProjects().keySet().iterator(); iterator.hasNext();
{
    String name = (String) iterator.next();
}

```

All casting and operator **instance of** must be ignored without calling `next()`. Otherwise, the result will be wrong:

```

Iterator iterator = company.getProjects().values().iterator();
Address address = (Address) getAddress();

while (iterator.hasNext())

{
    Project project = (Project) iterator.next();
}

```

If the value is a collection, the element type mechanism will be used for deep analyze. Here is an example of the association teams between Employee and Project:

```
public class Project

{
    private HashMap teams = new HashMap ();

    public void addTeamMember(String name, Employee member) {

        Collection team = (Collection) teams.get(name);

        if (team == null) {

            team = new ArrayList();

            teams.put(name, team);

        }

        team.add(member);

        member.setProject(this);

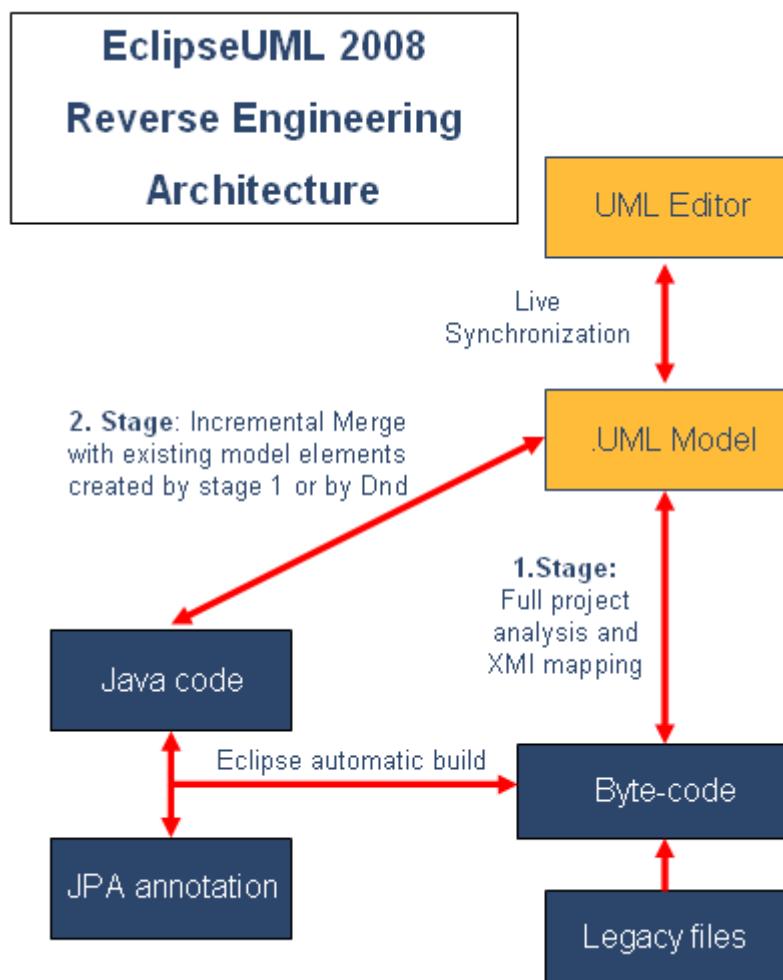
    }
}
```

Reverse Engineering Architecture

UML model reverse engineering is a complex processes which first detects all information from the code and then map it immediately to the UML editor and the UML Model.
It works at three simultaneous levels and analyzes all projects:

- byte code
- .class files
- .java files

The architecture is using the following workflow:



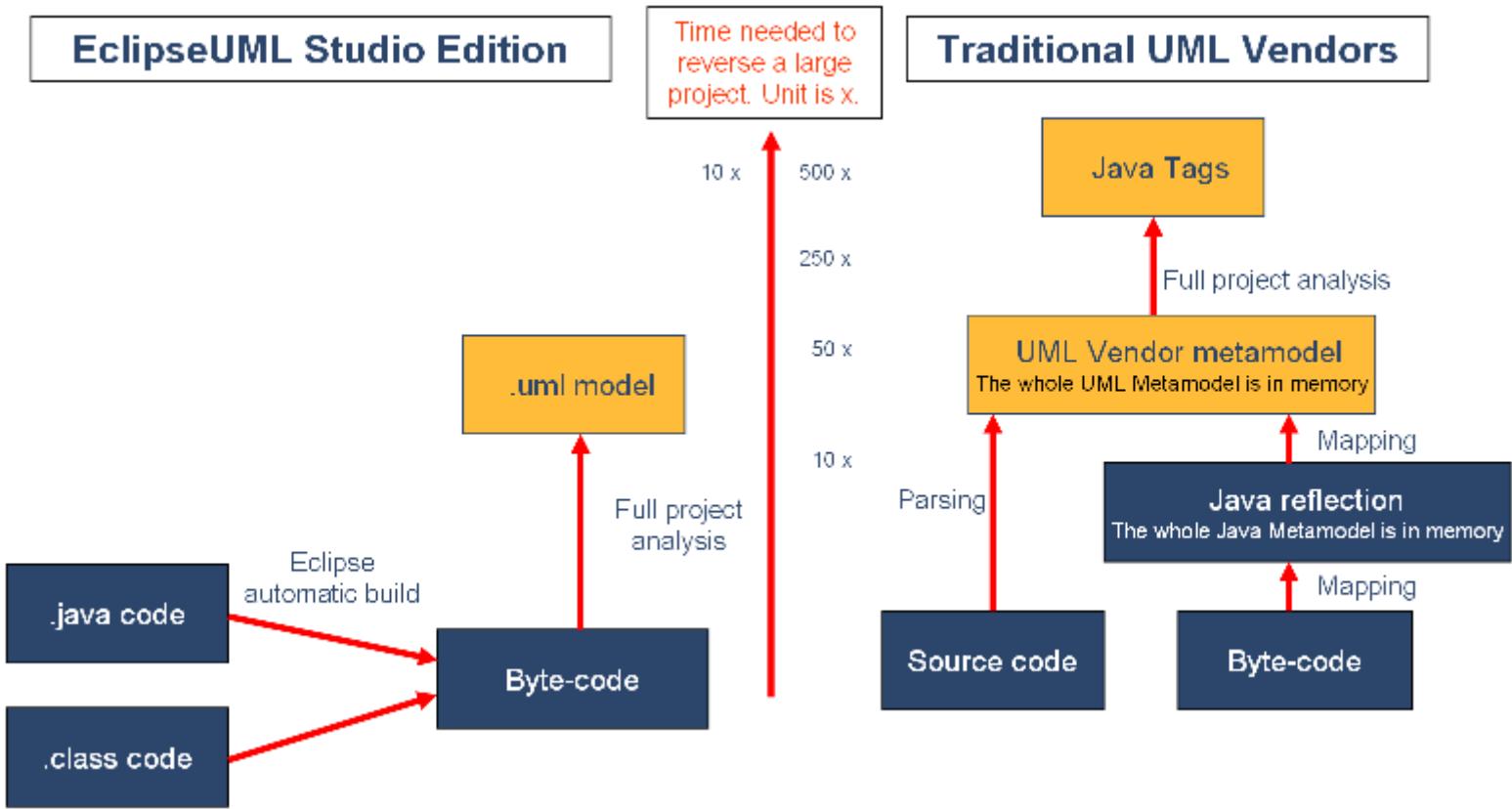
EclipseUML will detect at the end of the analyze stage the following information:

- nested packages
- classifiers structure (e.g. attributes, methods, inner classes etc....)
- inheritances
- associations
- dependencies
- JPA annotations

Working directly at byte-code level and mapping all information to .uml model file allows incredible speed and flexibility.

It is three times faster than writing tags in the code and 20 times faster than using a transition model such GMF.

This is a research study of the time needed to reverse large java projects.



Reverse Engineering Preferences

The reverse engineering is using three concepts:

- XMI backup will map all java information to .uml model. [More information ..](#)
- Dynamic navigation will only map java classifiers which are displayed inside diagrams. [More information..](#)
- Merge Model will merge new code with existing model. It keeps existing information safe as well as updating classifiers. [More information..](#)

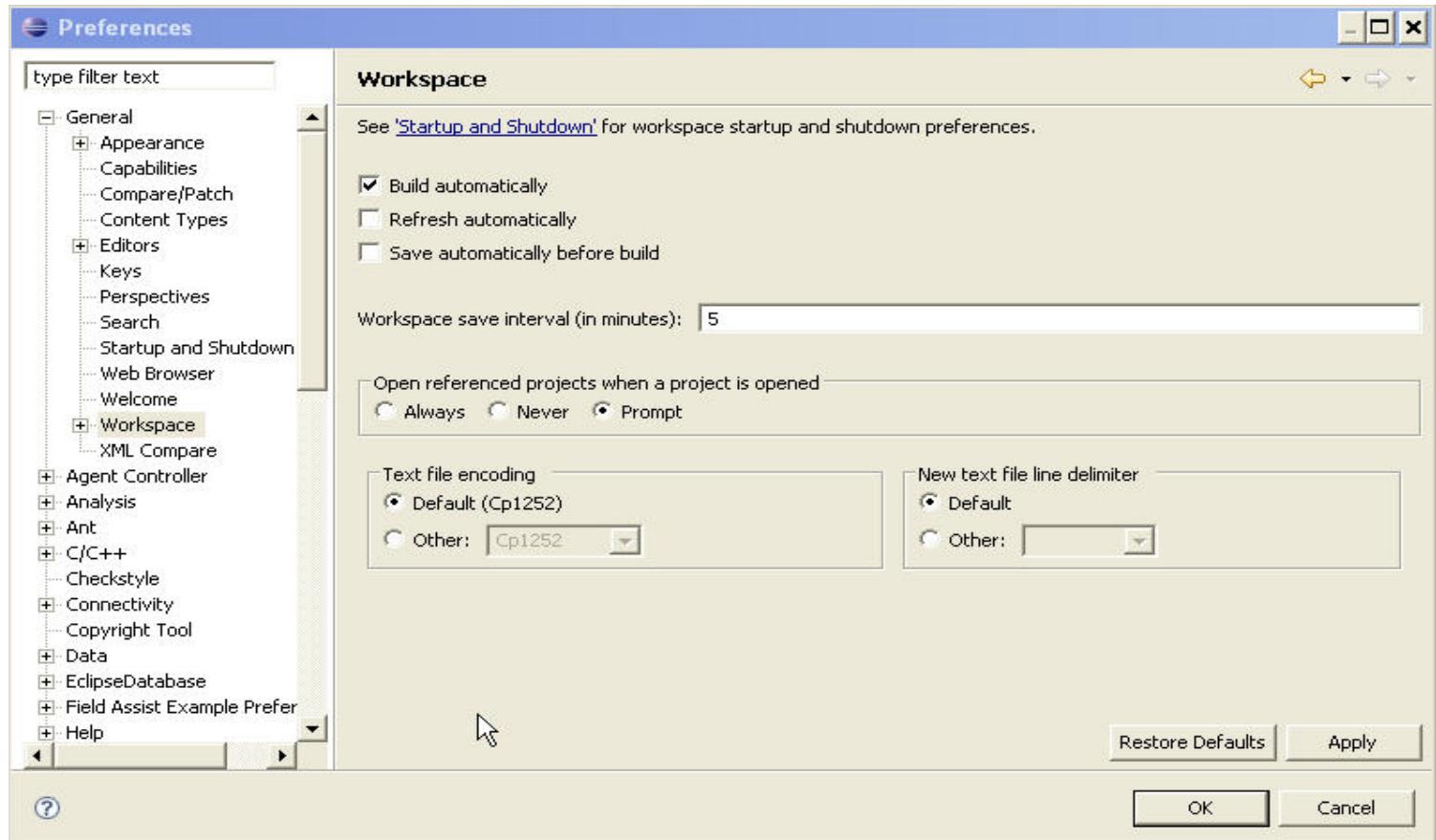
Preferences

In this section we will learn how to set up EclipseUML preferences. The following areas are covered:

- [Workbench Preferences](#)
- [JDT Preferences](#)
- [Setting Preferences](#)
- [Global EclipseUML Preferences](#)
- [Class Diagram](#)
 - [Association](#)
 - [Class](#)
 - [Colors](#)
 - [Dependency](#)
 - [Inheritance](#)
 - [Package](#)
- [PSM Code Generation](#)
- [Show/Hide](#)
- [Documentation generation](#)
- [Sequence Diagram](#)
- How EclipseUML's Preferences are working (2min30s): [Flash demo](#) / [.exe file](#)
- Using Class Diagram Preferences (2min): [Flash demo](#) / [.exe file](#)

Eclipse Preference

The workbench build option is used in the Workspace by the Java Development Toolkits (JDT) to compile the Java source after each modification.



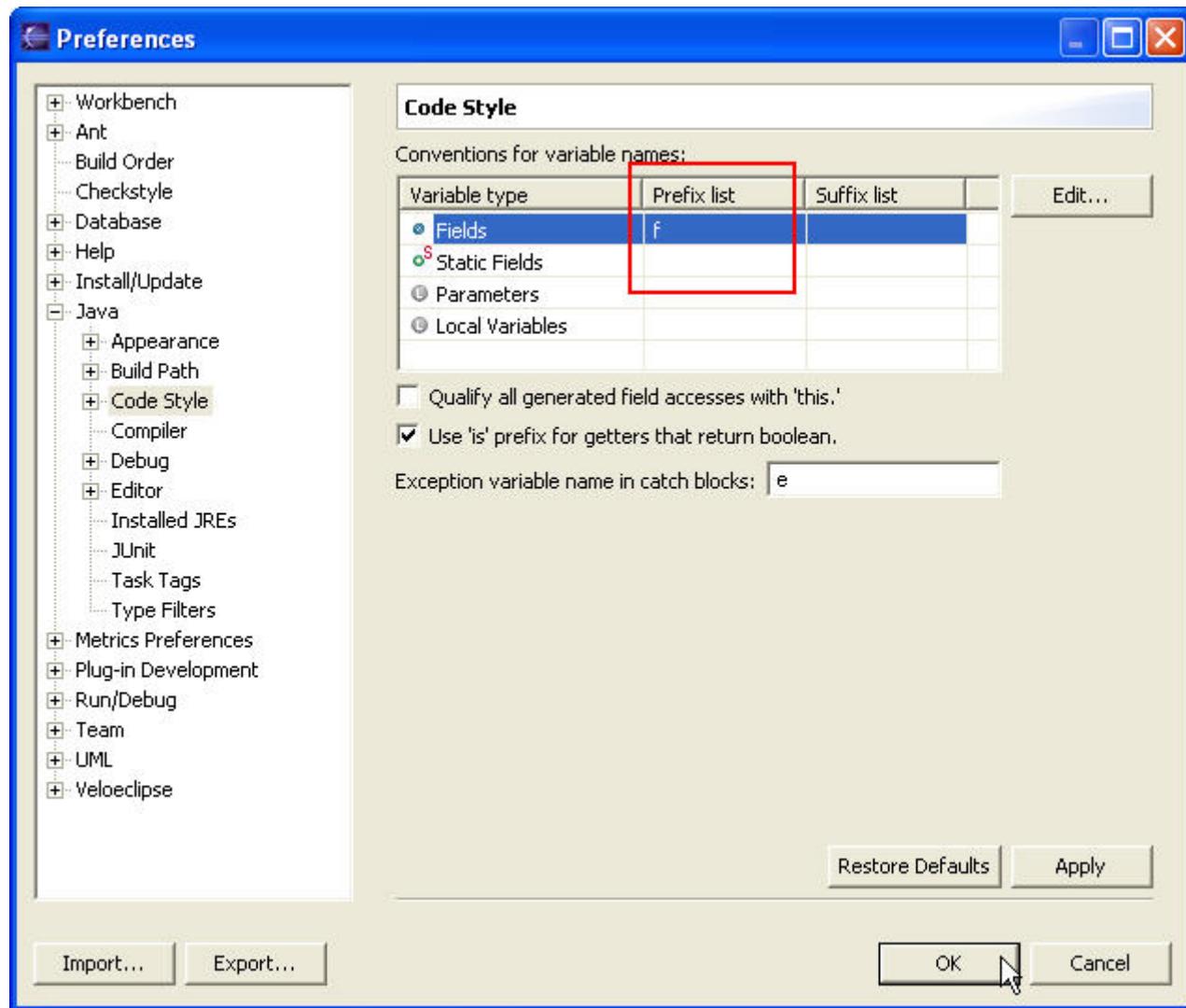
EclipseUML dependency detection and reverse engineering engine depend indirectly on this option. If this option is disable, both mechanisms may not work correctly.

JDT Preference

EclipseUML is tightly integrated in JDT. Therefore, JDT Java code generation options are used directly by two components in EclipseUML:

- [Reverse engineering](#)
- [Code generation](#)

In this example, f character will prefix any new field name.

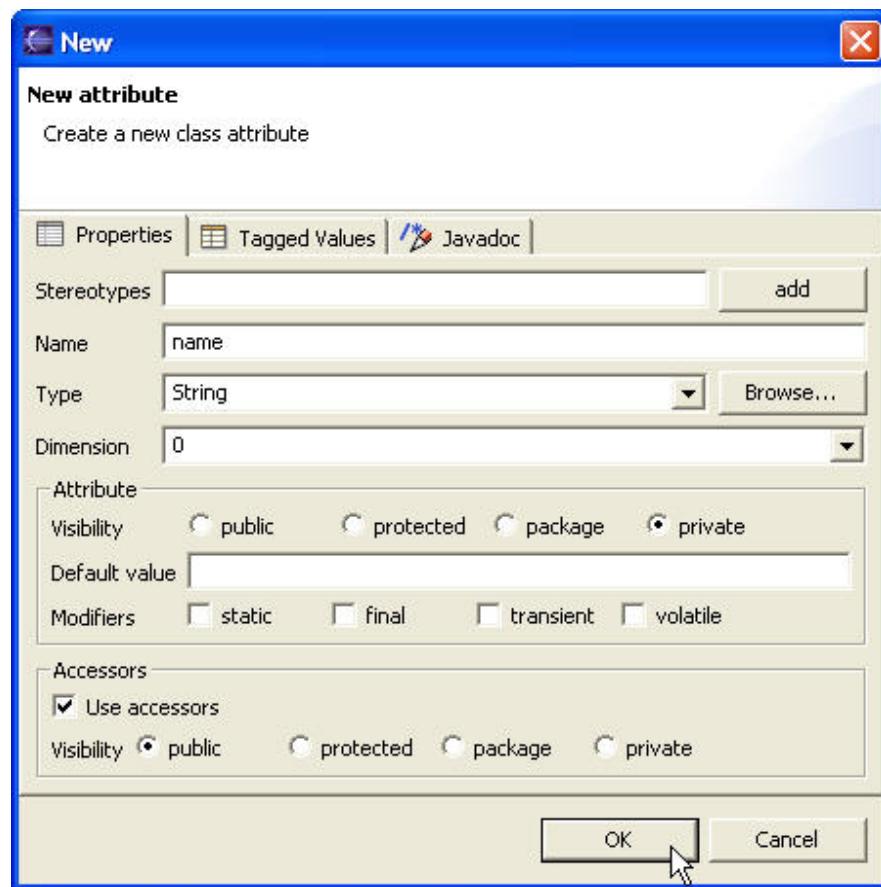


Reverse engineering

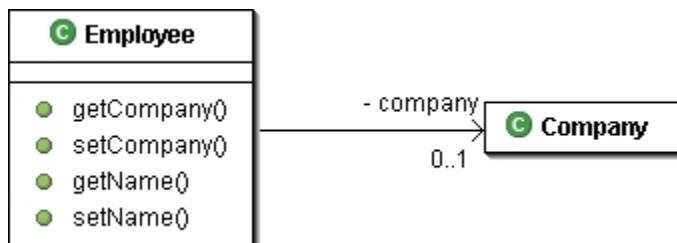
See the detail example in [Getter/Setter method recognition](#) in UML Model Reverse Engineering

Java Code generation

EclipseUML uses this option to generate Java code for attributes. When you create an attribute via diagram, EclipseUML code generator will concatenate the first prefix/suffix in this option to produce the Java attribute name. Using our previous example, if we add a new attribute String name:



we will get a Java attribute **fName** instead.



```
public class Employee
{
    /**
     * @uml.property =company associationEnd={multiplicity={(0 1)}
     ordering=ordered elementType=
     * company.Company}
     */
    private Company fCompany;
```

```

    /**
     * @uml property=company
     */
    public Company getCompany()
    {
        return fCompany;
    }

    /**
     * @uml property=company
     */
    public void setCompany(Company company)
    {
        fCompany = company;
    }

    /**
     *
     * @uml property=name
     */
    private String fName;

    /**
     *
     * @uml property=name
     */
    public String getName()
    {
        return fName;
    }

    /**
     *
     * @uml property=name
     */
    public void setName(String fName)
    {
        this.fName = fName;
    }
}

```

Setting Preferences

The Omondo mechanism of preferences is working at three different levels: global/diagram/element.

1. Eclipse preferences:
This includes workbench settings and JDT settings.
(first level)
2. Diagram preferences:
(second level)
3. Element Preferences:
(third level)

Depending on its nature, an element can keep its preferences.

Every element of a diagram or diagrams keeps their own preferences.

Changing global preferences will not impact your existing diagram's or element's preferences.

Different preferences can be introduced using:

1. The OK button will only change new diagrams or elements and will not change your existing diagrams.
2. The Apply button will dynamically change all existing diagrams' preferences.
3. Restore Default will automatically select the preference of the higher level. For example, clicking on Restore Default in the element preferences (third level); will automatically select the diagram preferences (second level). Clicking on Restore Default in the diagram preferences will select global preferences (first level).

Global EclipseUML Preferences

The **global Eclipse preferences** are used for every project. It is the highest level of preferences you can specify.

Three options are available:

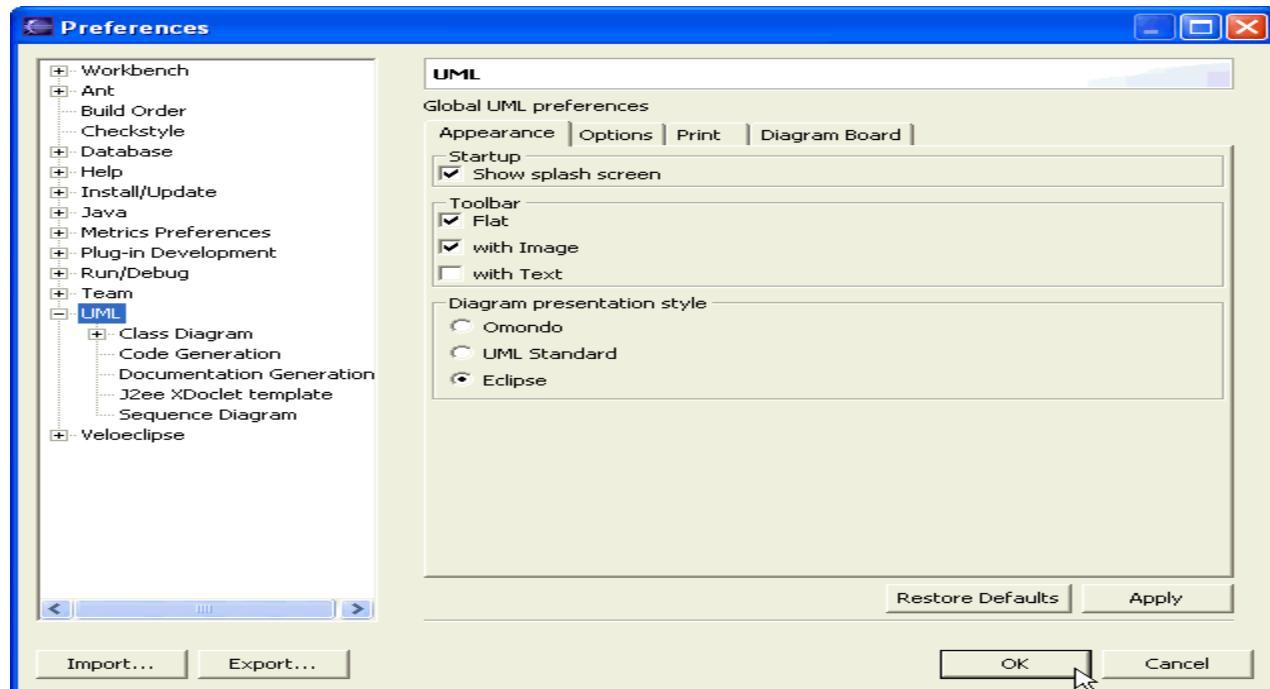
1. [Appearance](#)
2. [Option](#)
3. [Print](#)
4. [Diagram Board](#)

1. Appearance

Appearance preferences represent the general EclipseUML presentation when you first start modeling.

Three options are available:

- Startup
- Toolbar
- Diagram presentation style



Startup

This preference allows you to active the Omondo splash screen or not.

Toolbar

This preference allows you to choose the toolbar appearance.

Multiple possibilities are provided :

- Flat (the toolbar buttons are flat, with no relief)
- with Image (icons are available)
- with Text (text is printed)

By default, *flat* and *with Image* are selected, text is only visible on tooltips.

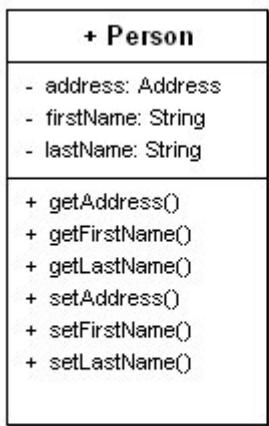
Diagram presentation style

This property allows you to choose a style for your UML diagram's appearance. Three possibilities are available :

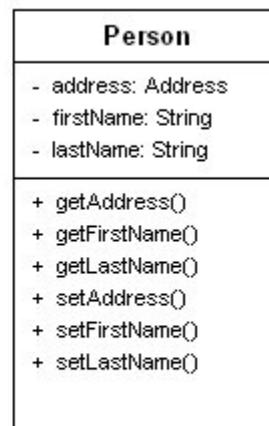
- Omondo (this look and feel, designed by Omondo, will evolve)
- UML Standard (the standard look and feel, just as usually in UML diagrams)
- Eclipse (The Eclipse look and feel, using the same colors and icons)

By default, the presentation style is set to *Eclipse*.

Here is an example of each style on a single class:



Omondo



Standard



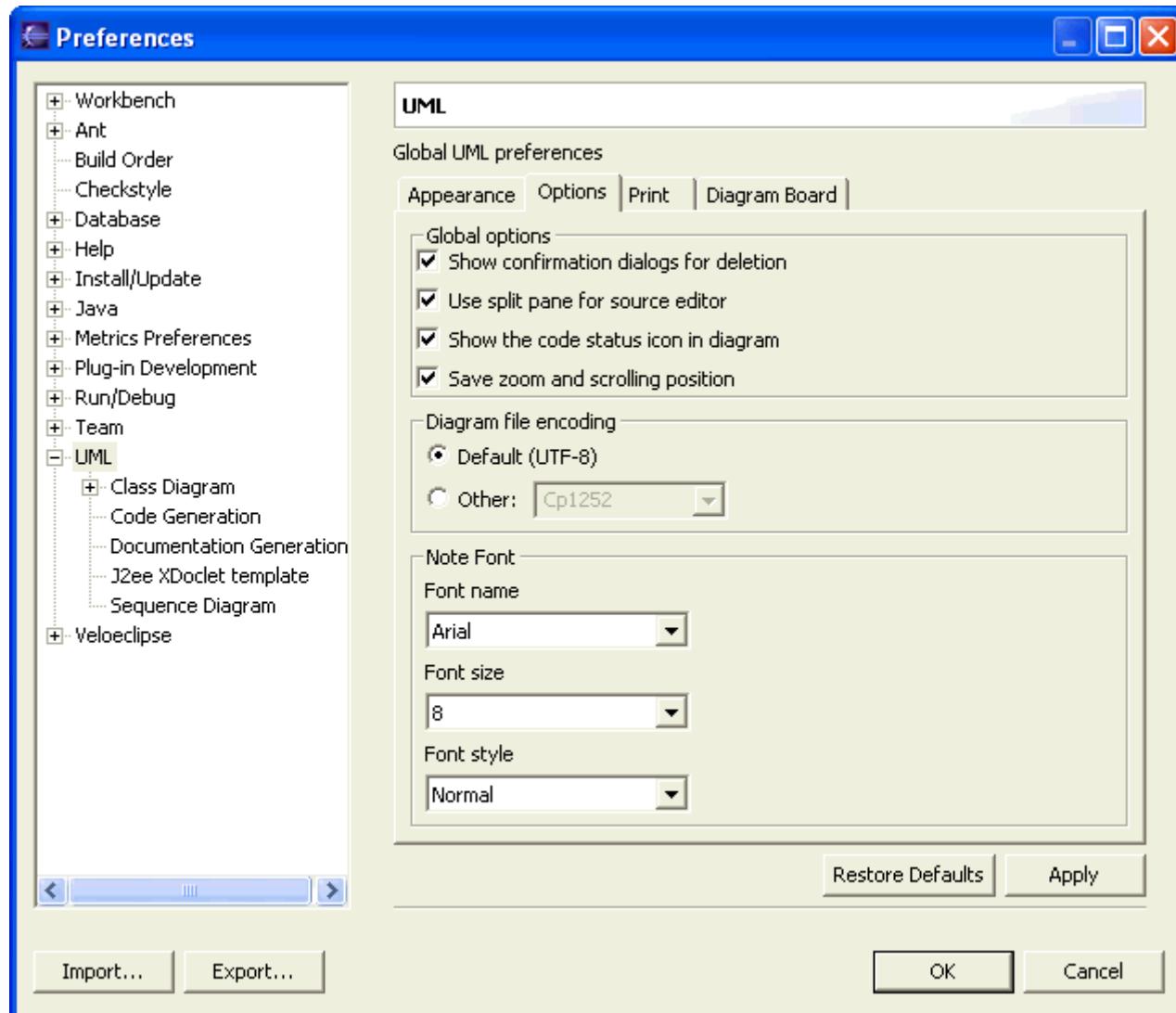
Eclipse

2. Options

Options Preferences represent the general EclipseUML customization.

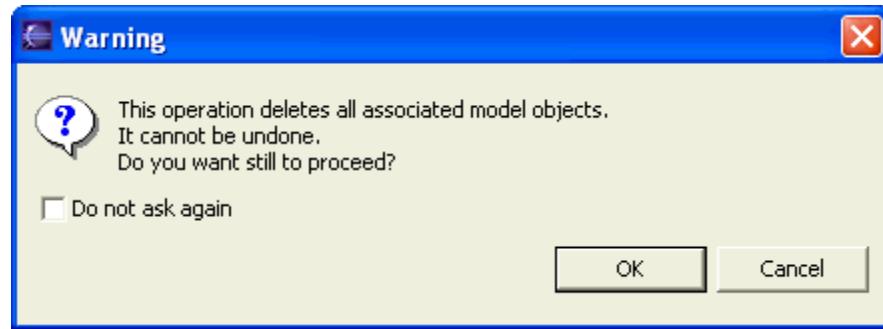
Four options are available:

- Global options
- Diagram file encoding
- Note Font



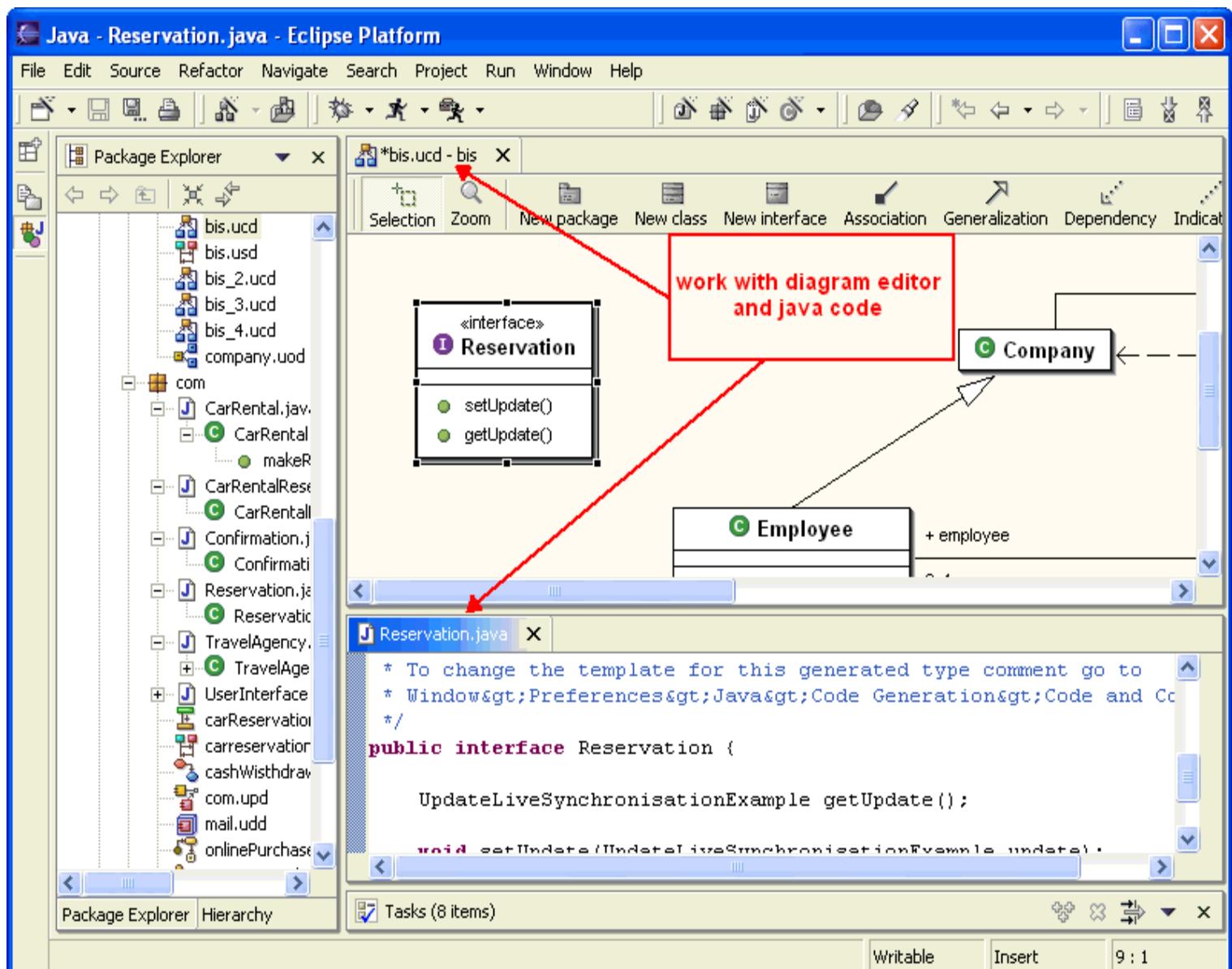
Global options

Show confirmation dialogs for deletion allows you to delete any element, with or without any warning. If you select this option then the warning will not appear when deleting an element. In order to reactivate this warning window, you must select the global preferences and unselect "Do not ask again".

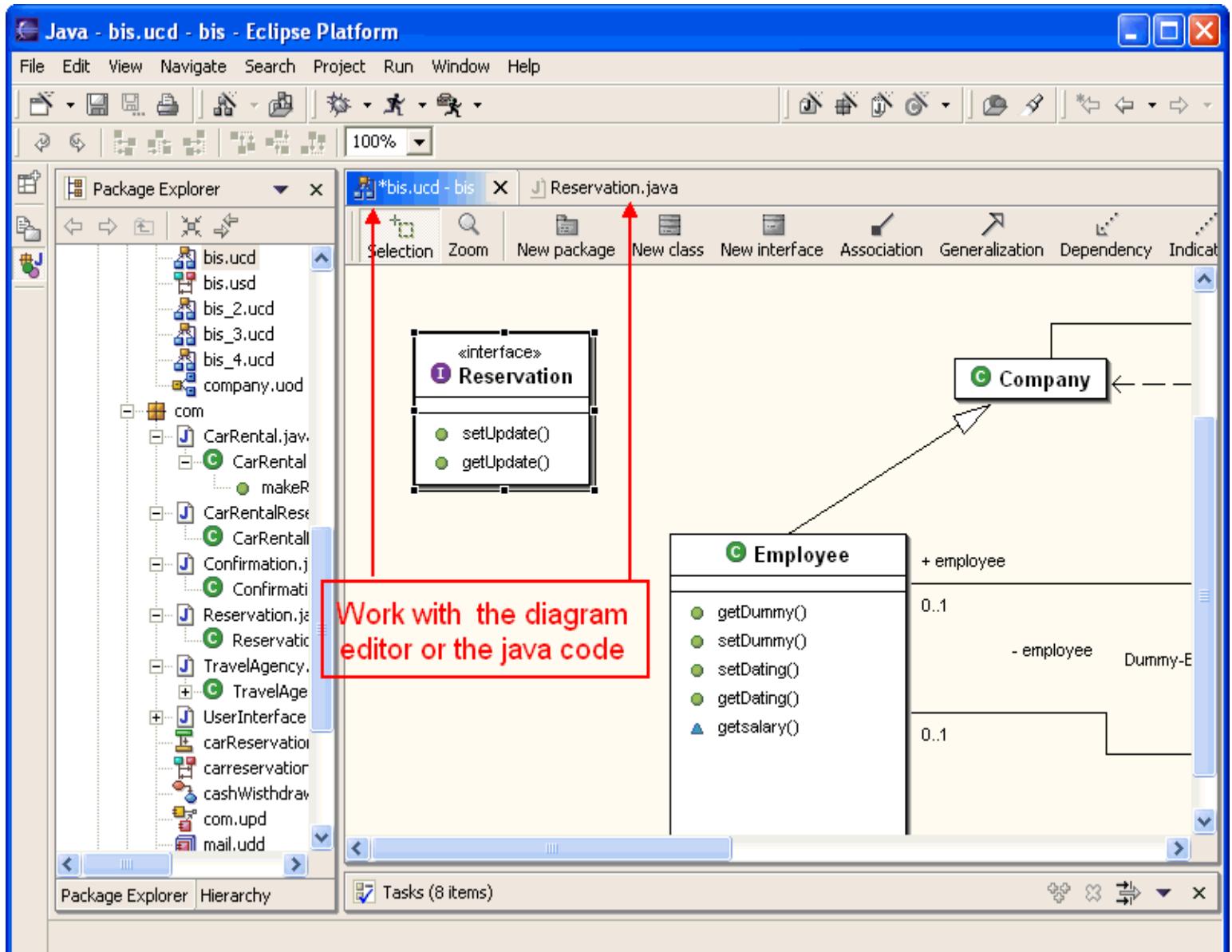


Use split pane for source editor allows you to have two different and simultaneous views of each editor or to keep both diagram and source editors in the same view.

If you select this option, you will be able to work with the diagram editor and the Java code in the same view.

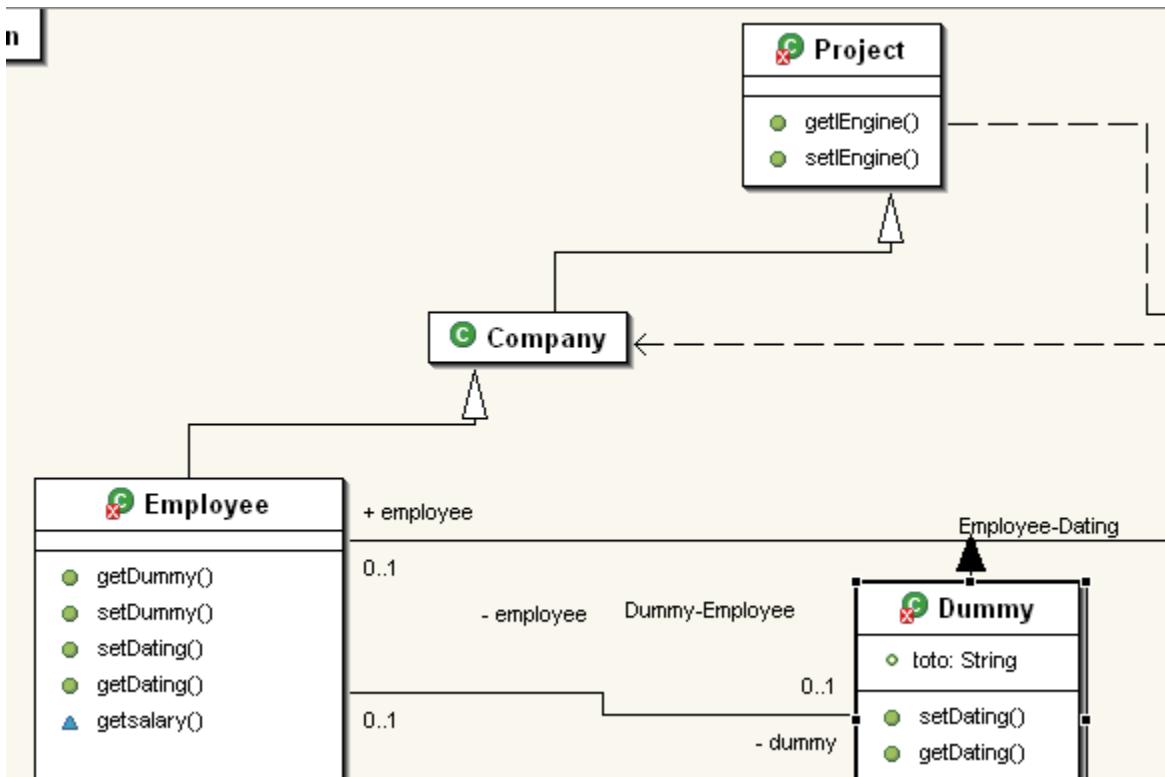


If you do not select this option, you will not be able to work with the diagram editor and the Java code in the same view.

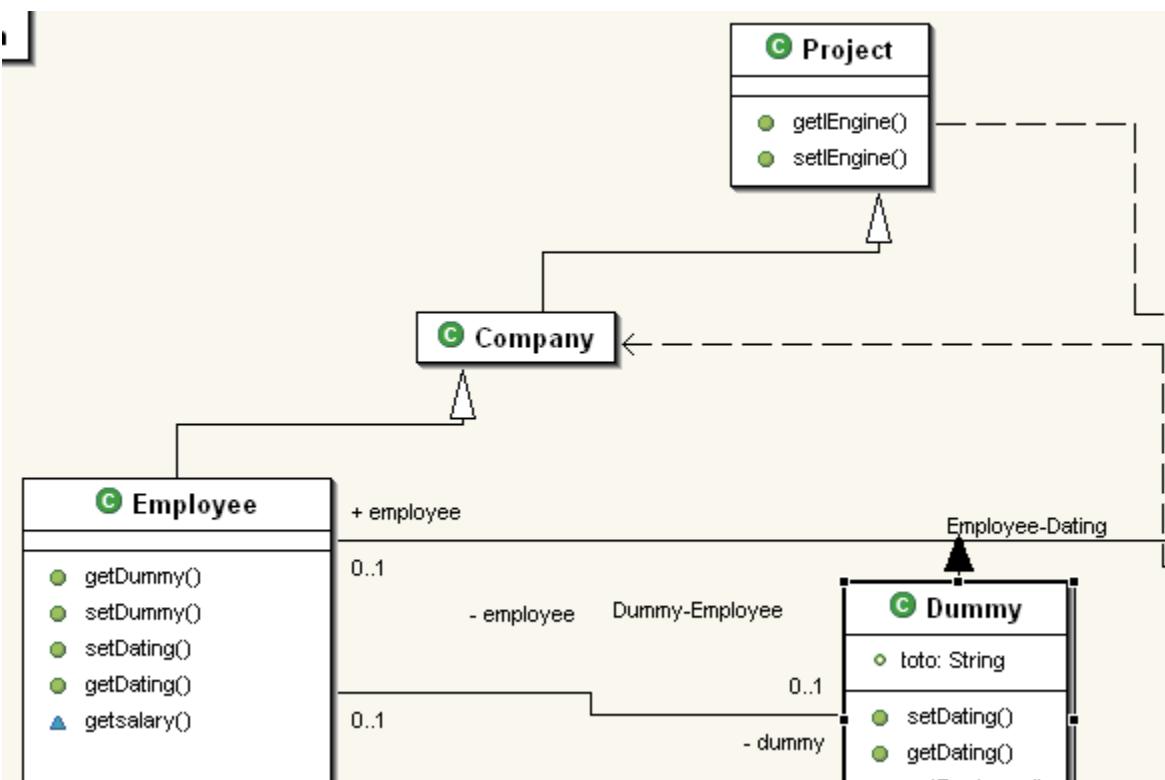


Show the code status icon in diagram is useful if using Eclipse diagram presentation style.

If you select show the code status, every error will be visible in the diagram editor:



If you do not select show the code status, errors will not be visible in the diagram editor:



Save zoom and scrolling position allows you to reopen a diagram as it was before it was closed. This is important in order to start your work exactly where you finished it before.

Encoding

The defined encoding is used when diagrams are saved.

Default (UFT-8) is recommended.

When using special characters in diagrams (including notes), it is important to save the diagram with an appropriate encoding. This is especially true when using Japanese, Chinese, or Korean languages.

Note Font

An appropriate font must be selected for exporting images including special characters.

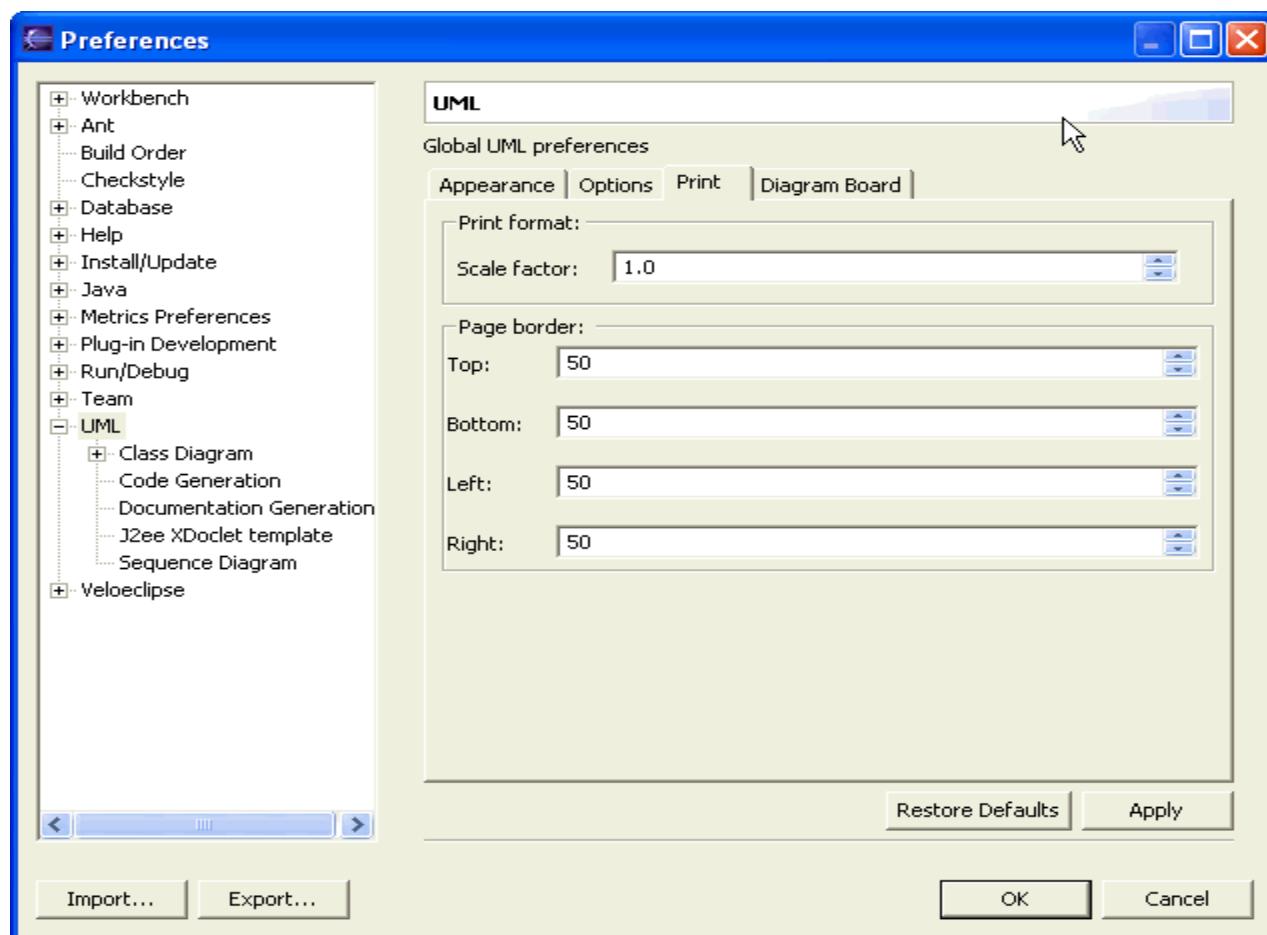
This is especially true when using Japanese, Chinese or Korean languages as the default font (Arial 12) doesn't support such characters.

3. Print

Options are proposed to configure the [diagram printing](#).

Two options are available:

- Print format
- Page Border



Print format represents the scale factor. The scale factor 1.0 is equivalent to a 100% presentation and 2.0 is equivalent to 200%. It is useful to visualize the overview before printing.

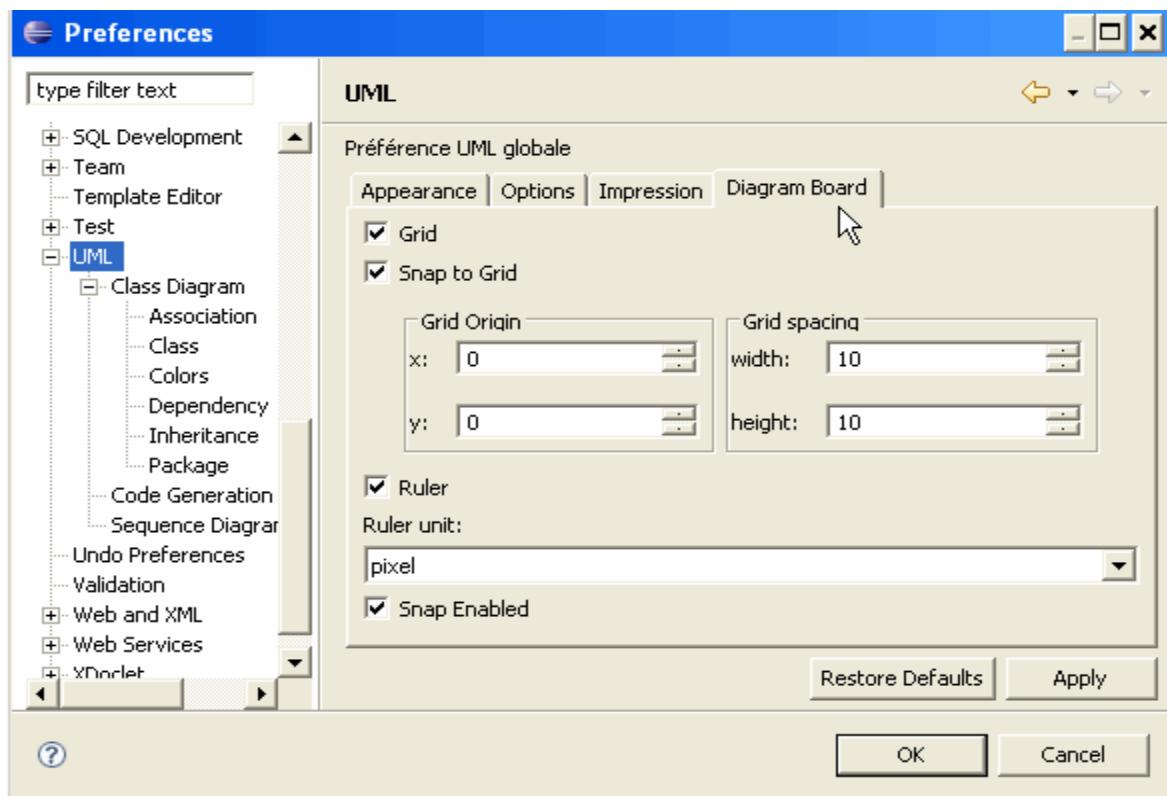
Page border represents the border customization of your diagram editor image and its export to your printer.

4. Diagram Board

Options are proposed to configure the diagram grid and ruler.

Six options are available:

- Grid
 - You can select to see or not to see the grid in any new diagram created after this selection)
- Snap to Grid
 - You can move element on the grid or not
- Grid spacing
 - Each element moves is depending on spacing
- Ruler visibility
 - You can see the ruler or not
- Ruler unit
 - You can select the unit
- Snap Enabled
 - You can align element with the blue line which appears when two elements are at the same time:
 - align top
 - align middle
 - align down



Class Diagram Preferences

In this section, you will understand how to configure the class diagrams through the Eclipse preferences. The UML preferences can be selected from the menu bar: **Window > Preferences > UML > Class Diagram**.

The class diagram preferences are set in the UML preferences sub menu.

- [Association](#)
- [Class](#)
- [Colors](#)
- [Dependency](#)
- [Inheritance](#)
- [Package](#)

If you select one of the following options, then each new class diagram will be created with the selected options

- From the Source area, the format after code insertion could be selected
 - Format after code insertion
- From the Wire Automation area, the java code reverse engineering can be customize in order to show:
 - Association
 - Inheritance
 - Dependency
- From Show/Hide Compartment, the different UML compartment can be shown or not depending on selecting:
 - Attribute Compartment
 - Operation Compartment
 - Literal Compartment
- From Reverse area, the reverse engineering option can be customize:
 - Ask for classifier selection when reversing a package
 - Show options dialog for each diagram navigation
 - Detect attribute properties
- From the scope options on class selection area, can be defined the range of the reverse engineering:
 - Scope
 - Level (reverse any element at 1 level for example. Please don't select over 3 for large projects in order to avoid to reverse few thousand elements)
- From the Dynamic file creation area, can be defined the class diagram save option in the package explorer
 - Do not create diagram file before saving action

Preferences

- + Connectivity
- Copyright Tool
- + Data
- + EclipseDatabase
- + Field Assist Example Preference
- + Help
- + Install/Update
- + Internet
- + Java
- + Model Validation
- + Plug-in Development
- + Profiling and Logging
- Readme Example
- + Report Design
- + Run/Debug
- + Server
- + SQL Development
- + Team
- Template Editor
- + Test
- UML
 - + Class Diagram
 - Association
 - Class
 - Colors
 - Dependency
 - Inheritance
 - Package
 - Code Generation
 - Sequence Diagram
 - Undo Preferences
 - Validation
- + Web and XML
- + Web Services
- + XDoclet

Class Diagram

General settings for Class Diagram reverse engineering

Source

Format after code insertion

Wire automation

Association

Inheritance

Dependency

Show/Hide Compartment

Attribute Compartment

Operation Compartment

Literal Compartment

Reverse

Ask for classifier selection when reversing a package

Show options dialog for each diagram navigation

Detect attribute properties

Scope options on class selection

Scope: project

Level: 1

Scope options on package selection

Scope: package

Level: -1

Dynamic file creation

Do not create diagram file before saving action

Buttons:

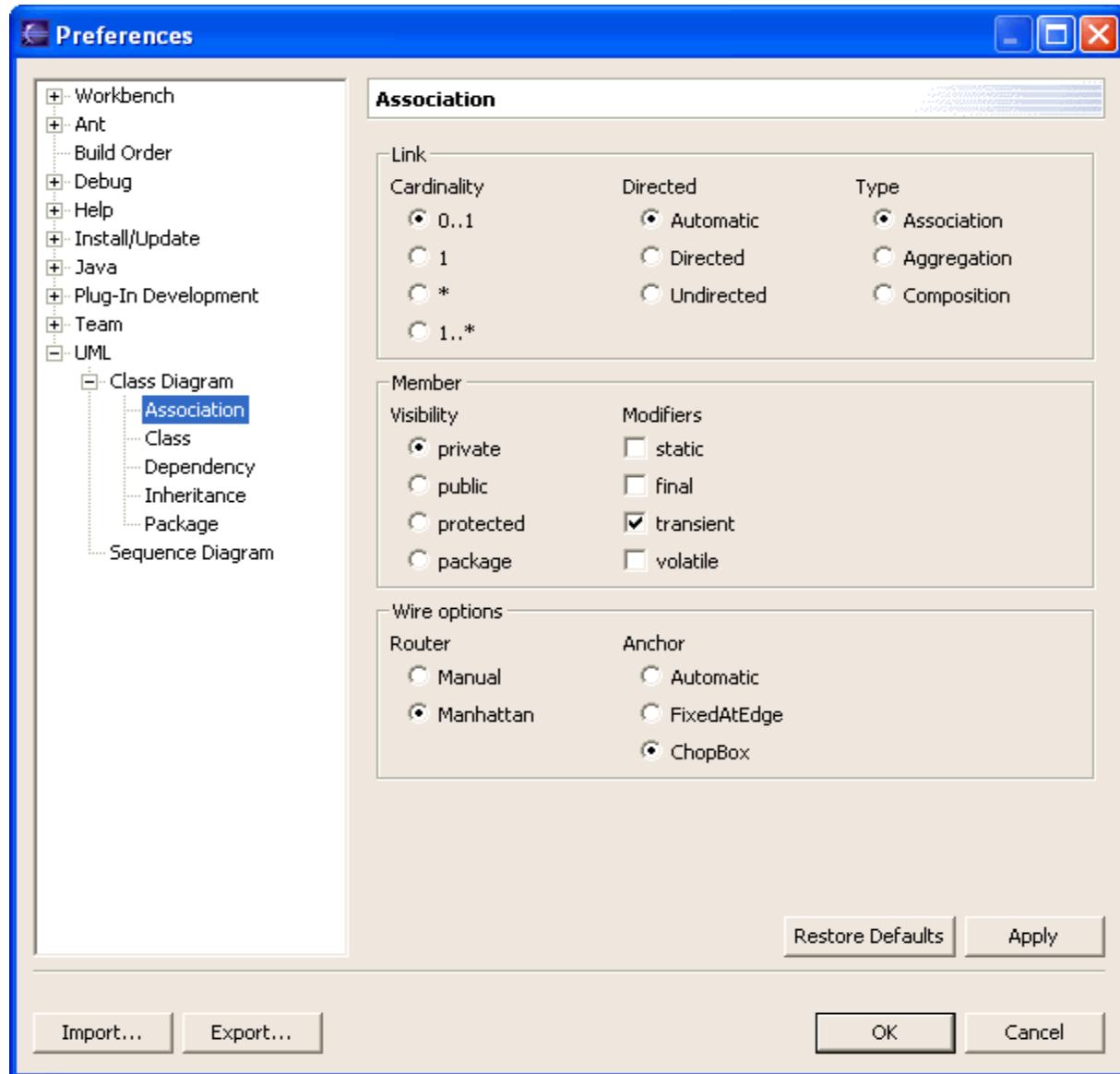
Restore Defaults Apply

OK Cancel

?

Associations

Associations can be customized from the preferences (UML > Class Diagram > Association).

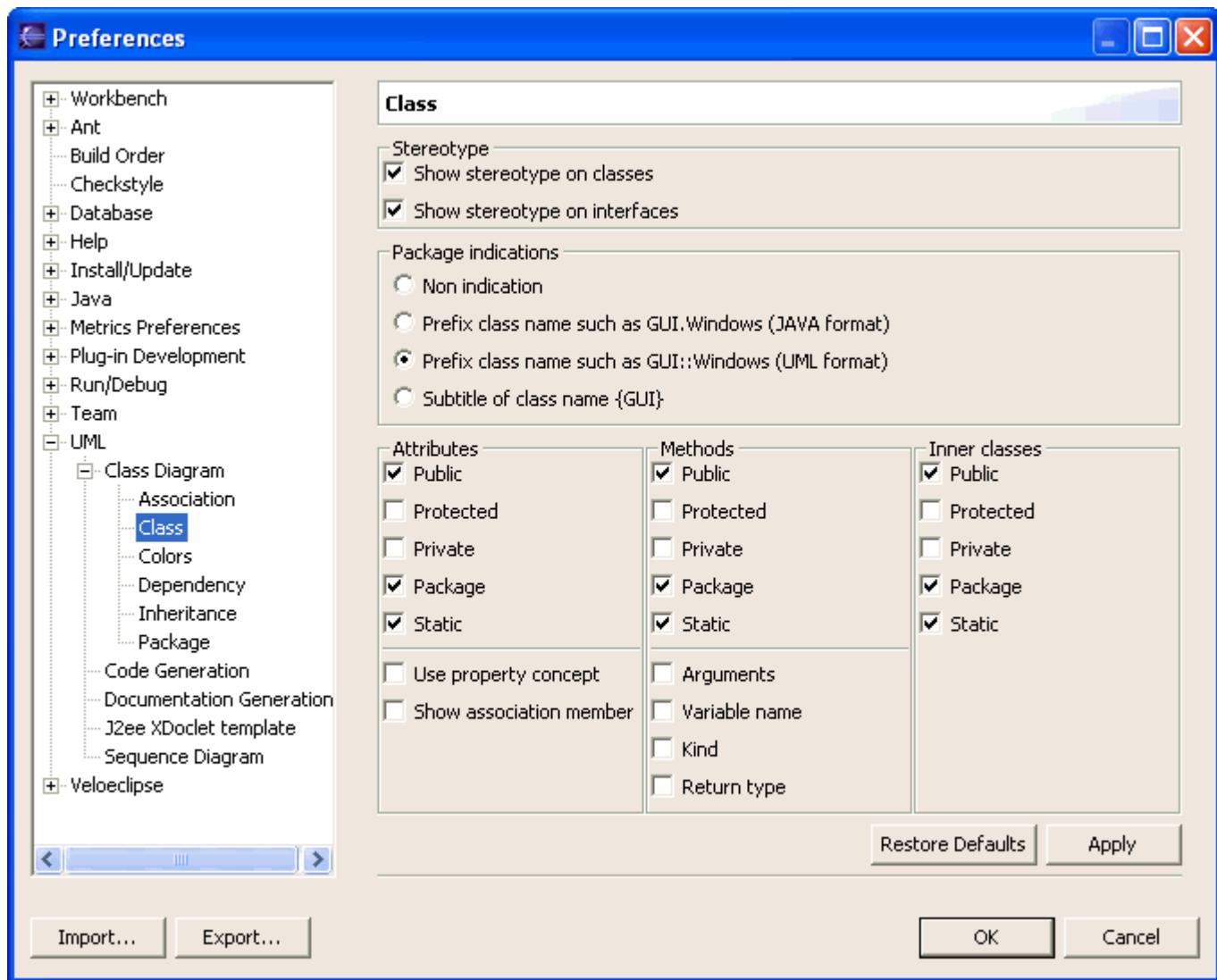


Three options are available:

- 1. From the Link area, the cardinality, the direction and the type of the association can be defined.
 - Cardinality
 - Direction
 - Type
- 2. From the member area, the visibility and the modifiers of the new members created as association ends can be defined.
- 3. From the wire area, [the router and the anchor](#) of the association can be defined.
 - The router link can be manual, or designed automatically using the Manhattan option
 - Anchor option can be automatically fixed with Chop box or manually fixed using FixedAtEdge.

Class

Classes and Interfaces can be customized from the preferences (UML > Class Diagram > Class).



Five major options are available:

1. Stereotype

This preference allows you to choose:

- Show stereotype on classes
- Show stereotype on interfaces

2. Package indications

Usually, classes and interfaces of a class diagram come from the same package. But sometimes they can come from different packages, and indication about the package from which classes or interfaces come from might be useful. This preference allows you to set how package information should be shown in the class diagram. This only applies to new classes or interfaces, any modification will not impact existing classes.

Four ways of package indications are available:

- No indication (only the class name is displayed)
- In a package view
- Prefix class name such as GUI:Windows (Classes from a different package are prefixed with their package name)
- Subtitle of class name {GUI} (Classes from a different package are subtitled with their package name)

3. Attributes

New functions:

Use property Concept: By default, the [property concept](#) is activated, so the attribute and its accessors are displayed as a single property. ([See getting started example.](#))

Show association member: By default, the show association member is not activated, ([See getting started example.](#))

This preference allows you to choose which attributes will be visible in your UML view. It is only applied to new classes; any modification will not impact existing classes.

Multiple possibilities are provided :

- Public (The public attributes will be visible).
- Protected (The protected attributes will be visible)
- Private (The private attributes will be visible).
- Package (The package attributes will be visible. An attribute without visibility modifier has the package visibility).
- Static (The static attributes will be visible if their visibilities are visible).
- Use property concept (The attributes and their accessors will be displayed as properties)
- Show association member (The attributes which are association ends will be displayed)

More detail is then provided for each class in the diagram through the [view selector](#).

4. Methods

This preference allows you to choose which methods will be visible in your UML view. It is only applied to new classes; any modification will not impact existing classes.

Multiple possibilities are provided :

- Public (The public methods will be visible).
- Protected (The protected methods will be visible)
- Private (The private methods will be visible).
- Package (The package methods will be visible. A method without visibility modifier has the package visibility).
- Static (The static methods will be visible if their visibilities are visible).
- Arguments (The argument types are visible).
- Variable name (The argument names are visible).
- Kind (The argument kinds are visible).
- Return type (The return type is visible).

More detail is then provided for each class in the diagram through the [view selector](#)

5. Inner classes

This preference allows you to choose which inner classes will be visible in your UML view. It is only applied to new classes; any modification will not impact existing classes.

Multiple possibilities are provided :

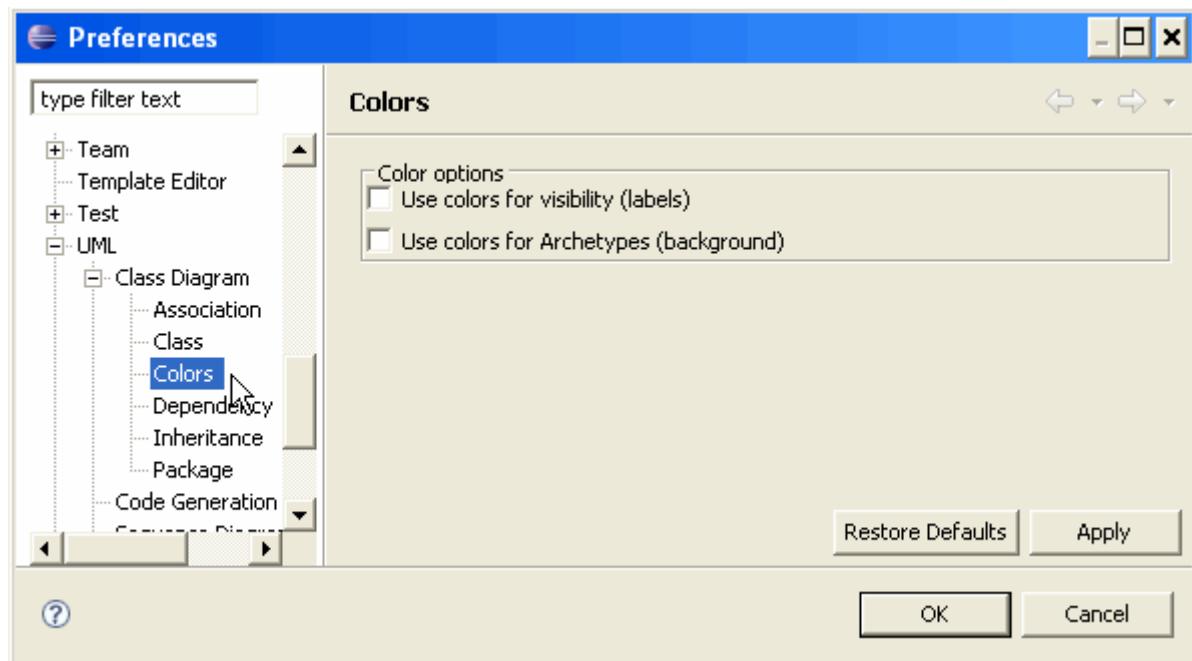
- Public (The public inner classes will be visible).
- Protected (The protected inner classes will be visible)
- Private (The private inner classes will be visible).
- Package (The package inner classes will be visible. An inner class without visibility modifier has the package visibility).
- Static (The static inner classes will be visible if their visibilities are visible).

More detail is then provided for each class in the diagram through the [view selector](#).

Colors

Colors can be customized inside your class diagram.

From the menu bar select **Window > Preferences > UML > Class Diagram > Colors**.



Three options are available:

1. Use colors for visibility

This preference allows you to choose the color of the text inside your diagram.

Eclipse presentation.



Eclipse presentation if Use of colors for visibility(labels) is selected in the menu.



2. Use colors for Package/Class/ Interfaces (borders).

This preference allows you to choose the color of borders inside your diagram.

Eclipse presentation.



Eclipse presentation if Use of colors for borders is selected in the menu.

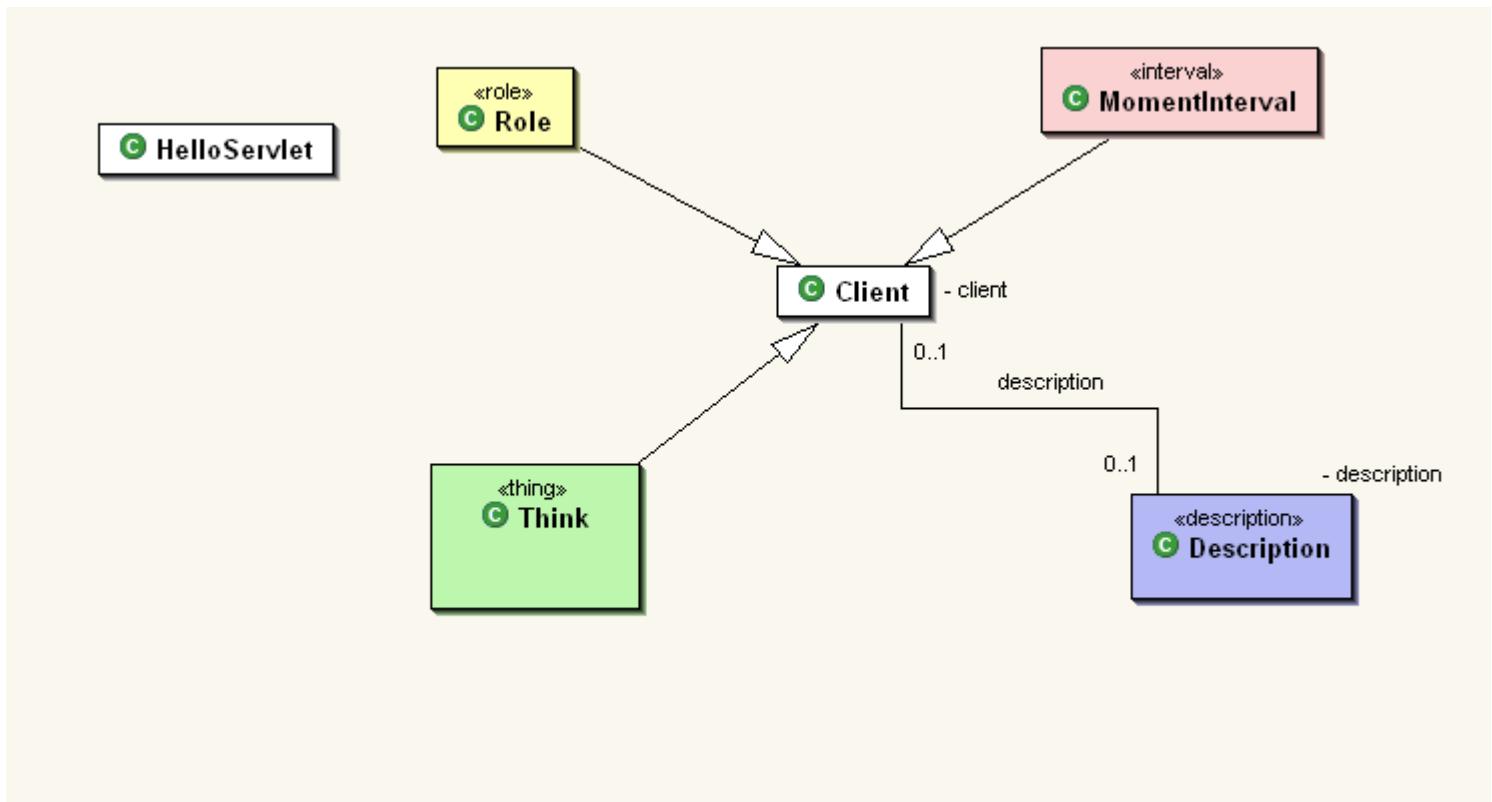


3. Use colors for Archetypes

This preference allows you to put colors depending on Archetypes in your Class Diagram Editor.

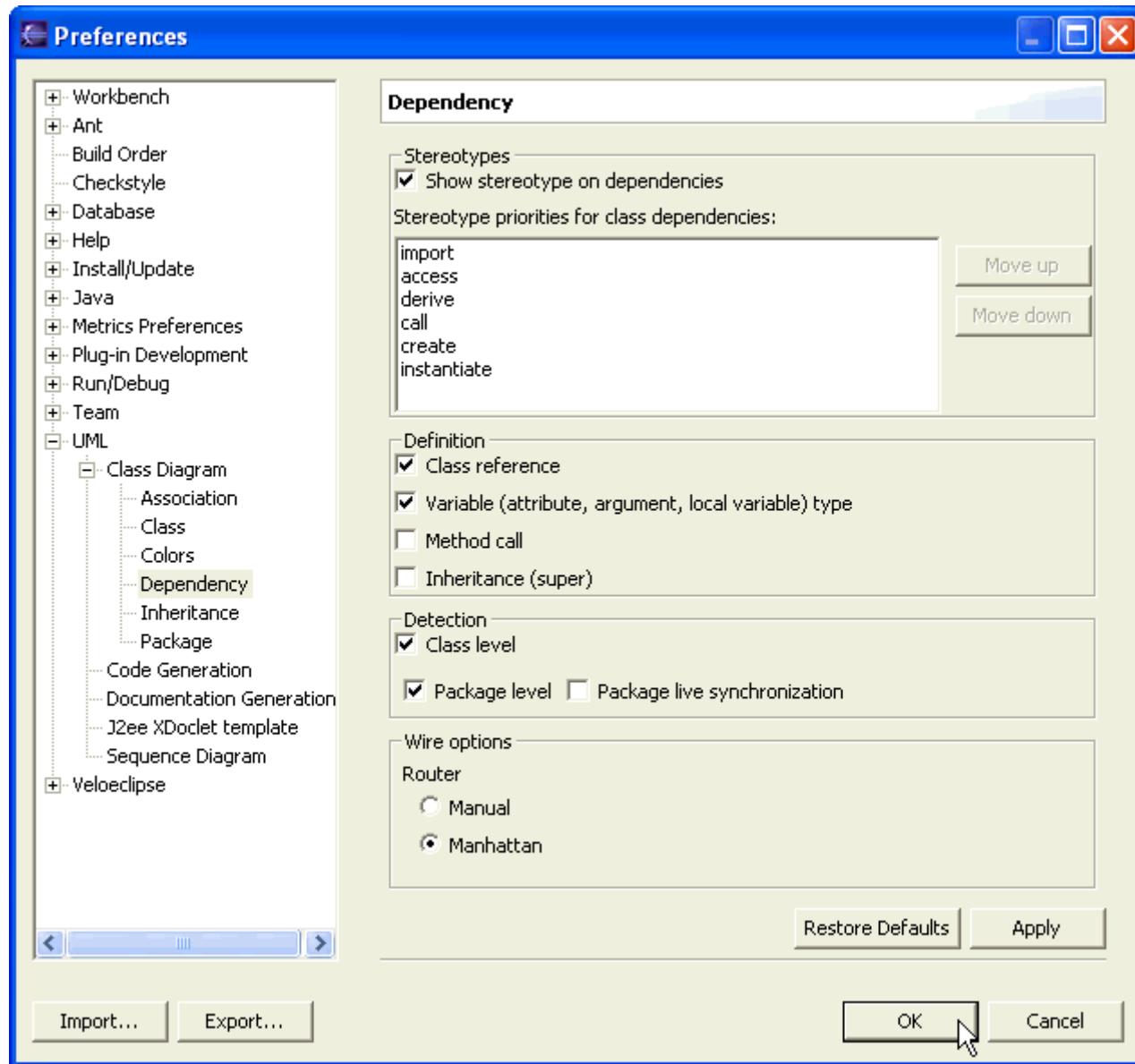
For possibilities are provided :

- Yellow: the value of the stereotype is *role*.
- Pink: the value of the stereotype is *moment or interval*.
- Green: the value of the stereotype is *thing*.
- Blue: the value of the stereotype is *description*.



Dependency

Dependencies can be customized from the preferences (UML > Class Diagram > Dependency).



Different options are available:

1. The Stereotype area:

Either **show or not show the stereotype** on dependencies

Select **stereotype priorities** for class dependencies

EclipseUML will just show one stereotype on a class dependency, even if a class dependency could have more than one stereotype. We will therefore have to select a priority order within the six kinds of stereotype. This priority order will decide which stereotype will be shown.
The six kinds of stereotype are (considering two elements A and B):

- **import**: B is imported by A

- **access**: A uses a B attribute
- **derive**: A derives from B
- **call**: A uses a B method
- **create**: A is a B factory
- **instantiate**: A instantiates B

2. Dependency definition area

This preference allows you to define the dependency relationships:

Options	Stereotypes	Description
Class reference	«access»	Access directly class attributes and call static methods.
	«instantiate»	Create an instance
Variable type	«access»	All variable types used in a class/interface such as attribute type, method argument type, method return type and local variable type.
Method call	«call»	Invoke an instance method
Inheritance (super)	«derive»	Specification or implementation of one type

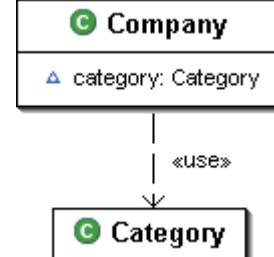
This definition is used to detect the dependency relationships once a Java class is modified.

Class reference

Direct class attribute access

```
public class Category
{
    static public Category software
        = new Category();
}

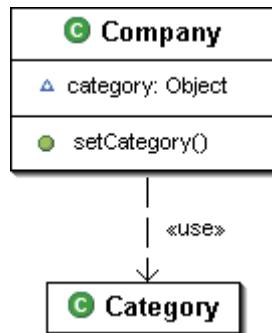
public class Company {
    Object category = Category.software;
}
```



Instance creation

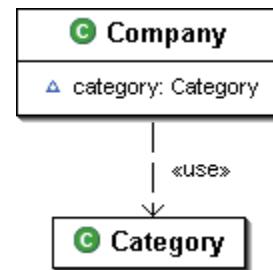
```
public class Category
{
}

public class Company {
    Object category = new Category();
}
```



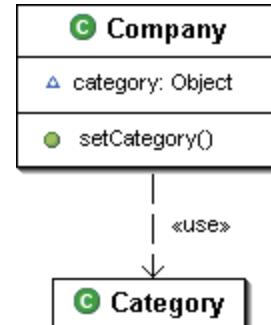
Variable type

```
public class Category {  
}  
  
public class Company {  
    Category category;  
}
```



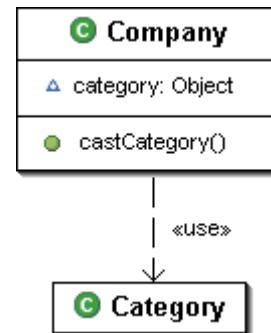
Method argument type

```
public class Category {  
}  
  
static public Category software  
    = new Category();  
}  
  
public class Company {  
    Object category;  
    public void setCategory(Category  
category) {  
        this.category = category;  
    }  
}
```



Method return type

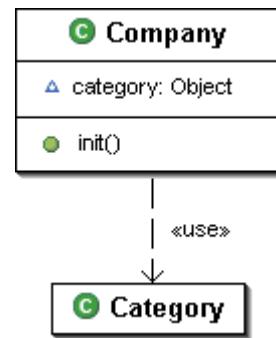
```
public class Category {  
}  
  
public class Company {  
    Object category;  
    public Category castCategory() {  
        return (Category) category;  
    }  
}
```



Local variable type

```
public class Category
{
}

public class Company {
    Object category;
    public void init() {
        Category defaultCategory = null;
        this.category = defaultCategory;
    }
}
```



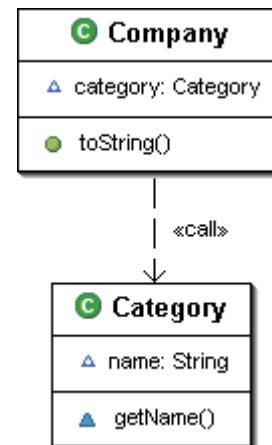
Method Call

The dependency based on the method call is setup between the caller and the method declaration class.

```
public class Category
{
    String name;

    String getName() {
        return name;
    }
}

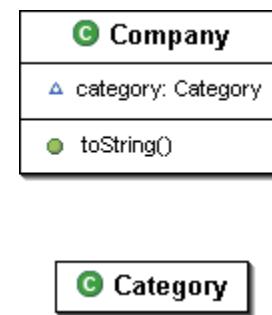
public class Company {
    Category category;
    public String toString() {
        return category.getName();
    }
}
```



The following example doesn't fit this condition since the method `toString()` isn't defined the class `Category`.

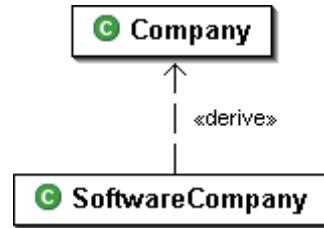
```
public class Category {
}

public class Company {
    Category category;
    public String toString() {
        return category.toString();
    }
}
```



Inheritance

```
public class Company {  
}  
  
public class SoftwareCompany extends  
Company {  
}
```



3. Dependency detection area

Two levels of dependency are supported: between classes and packages. The package dependency live synchronization needs extra CPU resources.

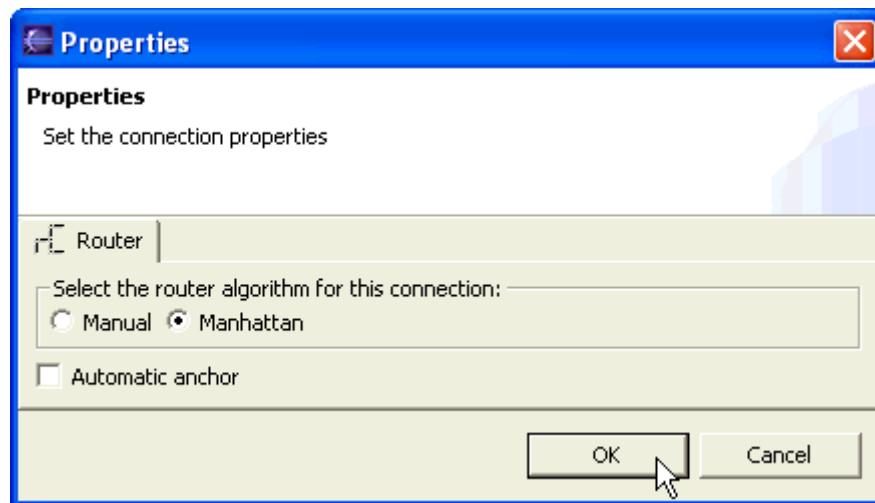
4. Wire options area

This preference allows you to choose the dependency default [router and anchor](#).

- The router link could be manual or designed automatically using the Manhattan option

Inheritance

Associations can be customized from the preferences (UML > Class Diagram > Inheritance).



Wire options

This preference allows you to choose the inheritance default [router and anchor](#).

- The router link can be manual or designed automatically using the Manhattan option
- The anchor can be automatically fixed by selecting the, "Automatic anchor" checkbox

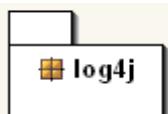
Package

The Omondo mechanism of Package preferences has two different levels: global and package. From the menu bar, Select **Window > Preferences > UML > Class Diagram > Package**

1. The Eclipse preferences concerning package are:

(first Level)

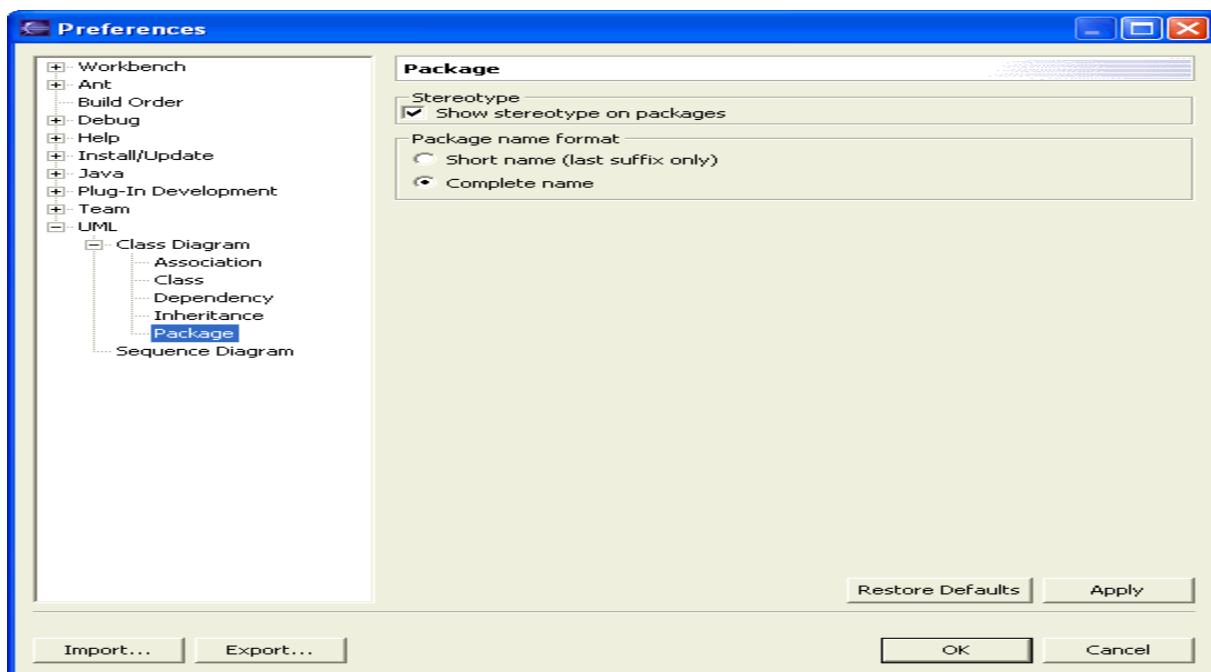
- In the **stereotype** area, you can decide to either show or not show the stereotype on packages
- In the Package name format area, you can decide either to have:
 - a short name



- or a complete name



According to the general [EclipseUML preference mechanism](#), these preferences will not change the existing diagram. Existing diagrams can be changed using the diagram editor preferences. Using the Apply button allows you to change the existing package's preferences, while clicking on the OK button will just validate the preferences for new packages. Here is what the Package preferences page looks like:



2. The class diagram preferences concerning package are:
(second Level)

- In the **stereotype** area, you can decide to either show or not show the stereotype on packages
- In the Package name format area, you can decide either to have a short or a long name.

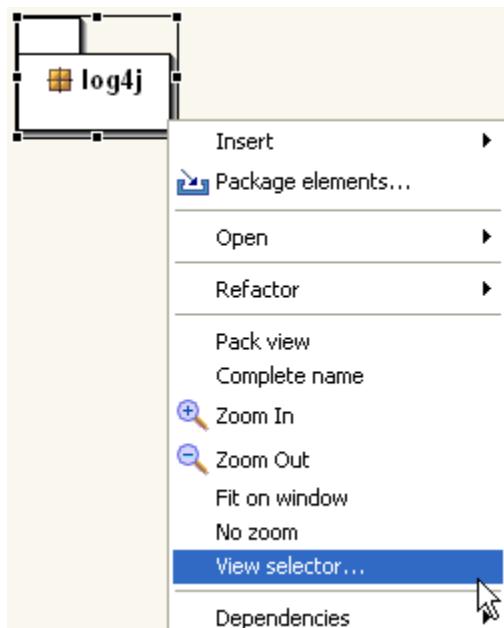
You can work at class diagram level using the diagram editor's popup menu.
Open your class diagram editor and click inside the diagram. In order to work at global diagram level, be sure not to select any element.

3. The Package preferences concerning package are:
(third Level)

- In the **stereotype** area, you can decide to either show or not show the stereotype on packages
- In the Package name format area, you can decide either to have a short or a long name.

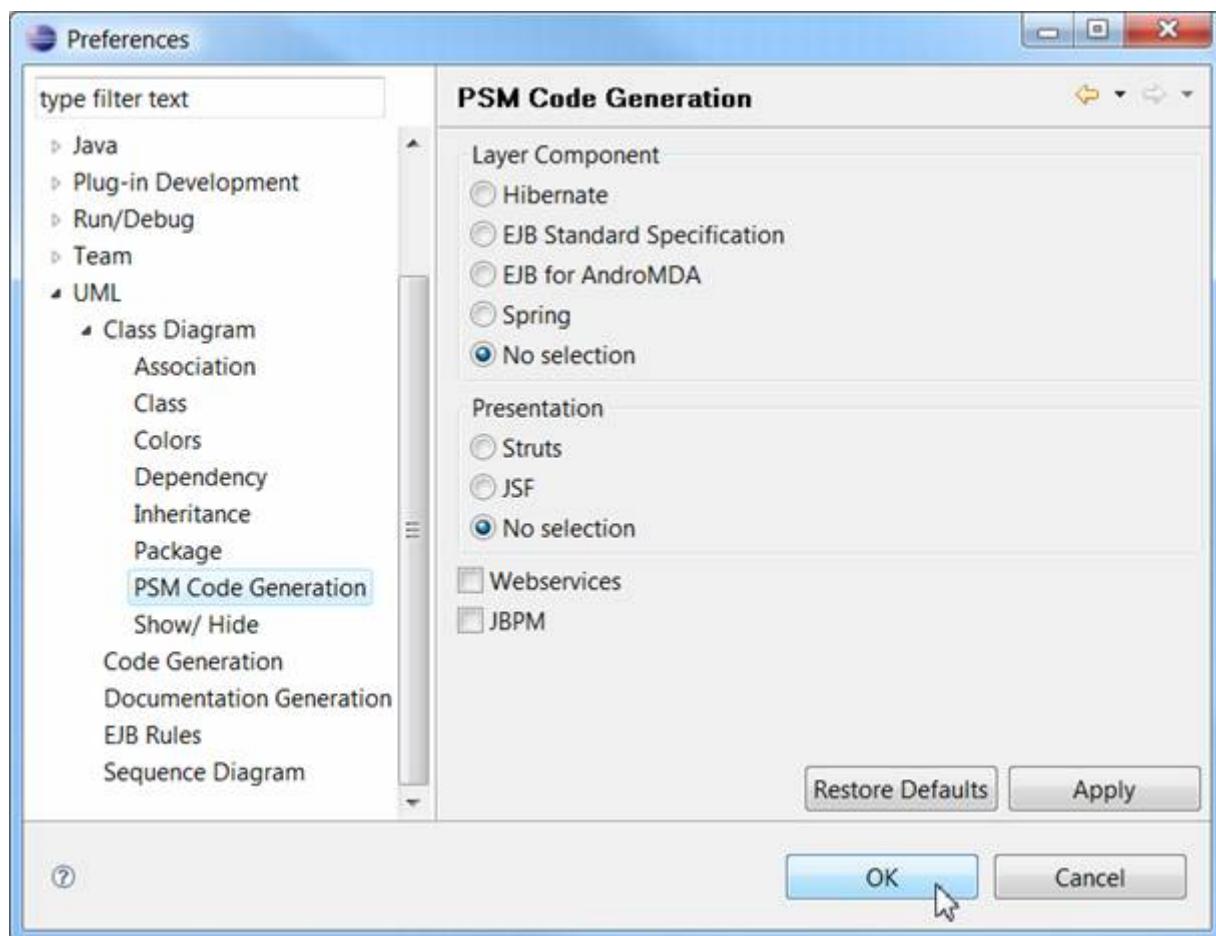
You can work at package level by selecting one package then right click to open the popup menu and finally choose View selector.

Here is the example:



PSM Code Generation

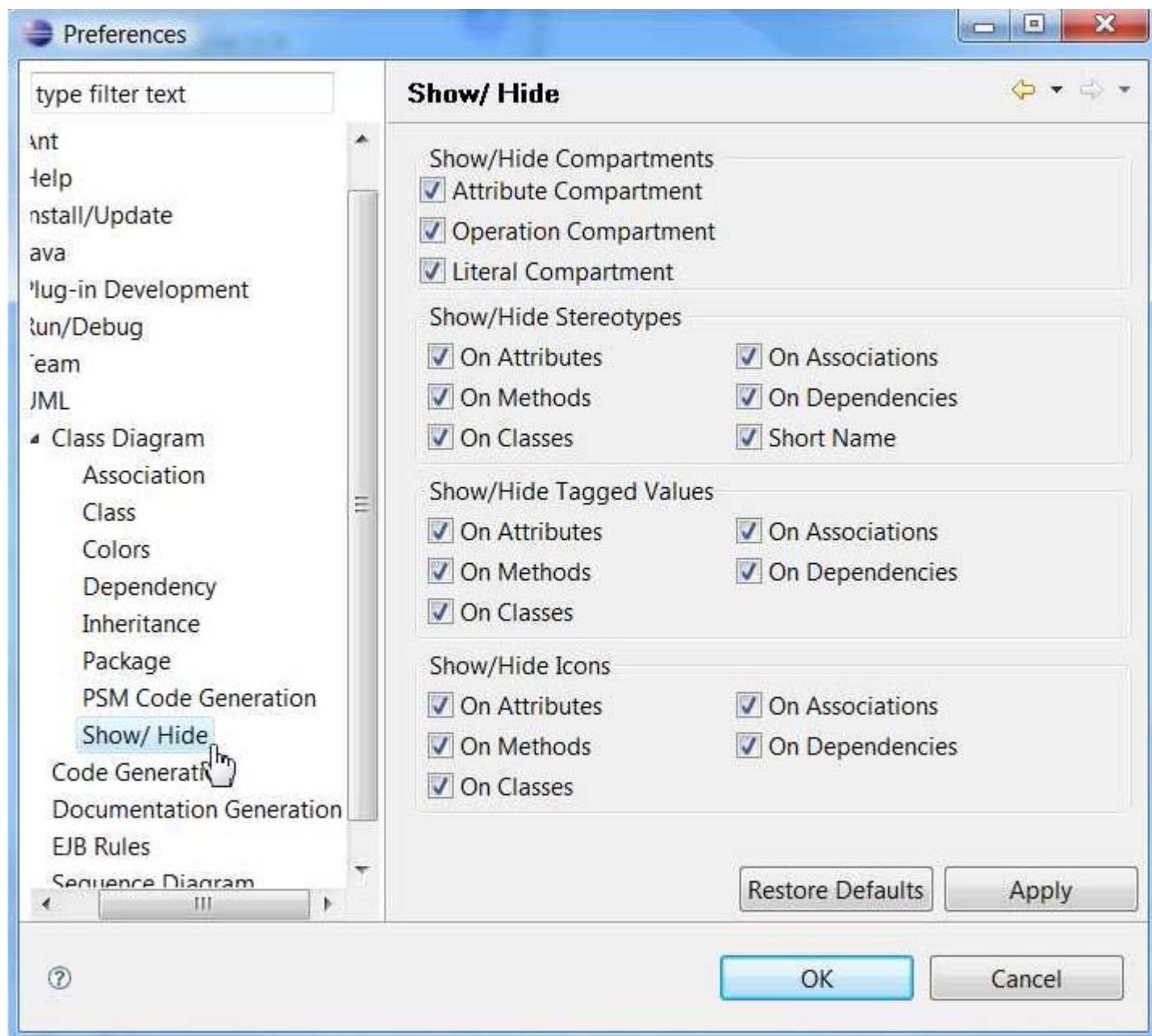
The PSM code generation preference allows you to activate or not the JEE menu in the class diagram.



Show/Hide

The Show Hide Preferences allows hiding:

- Attribute Compartment
- Operation Compartment
- Literal Compartment
- Stereotypes
- Tagged Values
- Icons

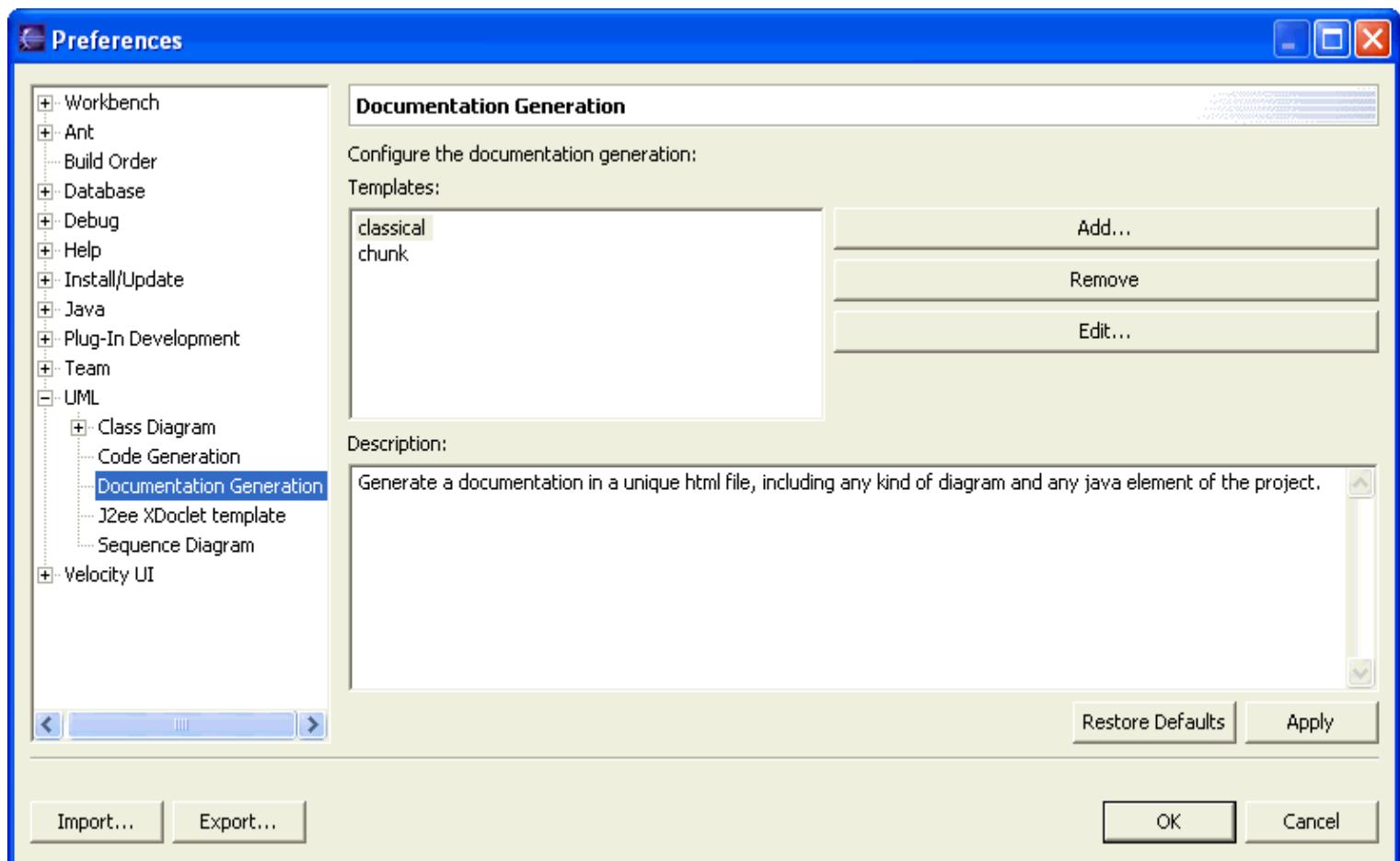


Documentation Generation

The **documentation generation preferences** are used for customizing the EclipseUML project documentation generated.

The following options are available:

- Configure the documentation generation using templates
 - Add
 - Remove
 - Edit
- Check the generation description



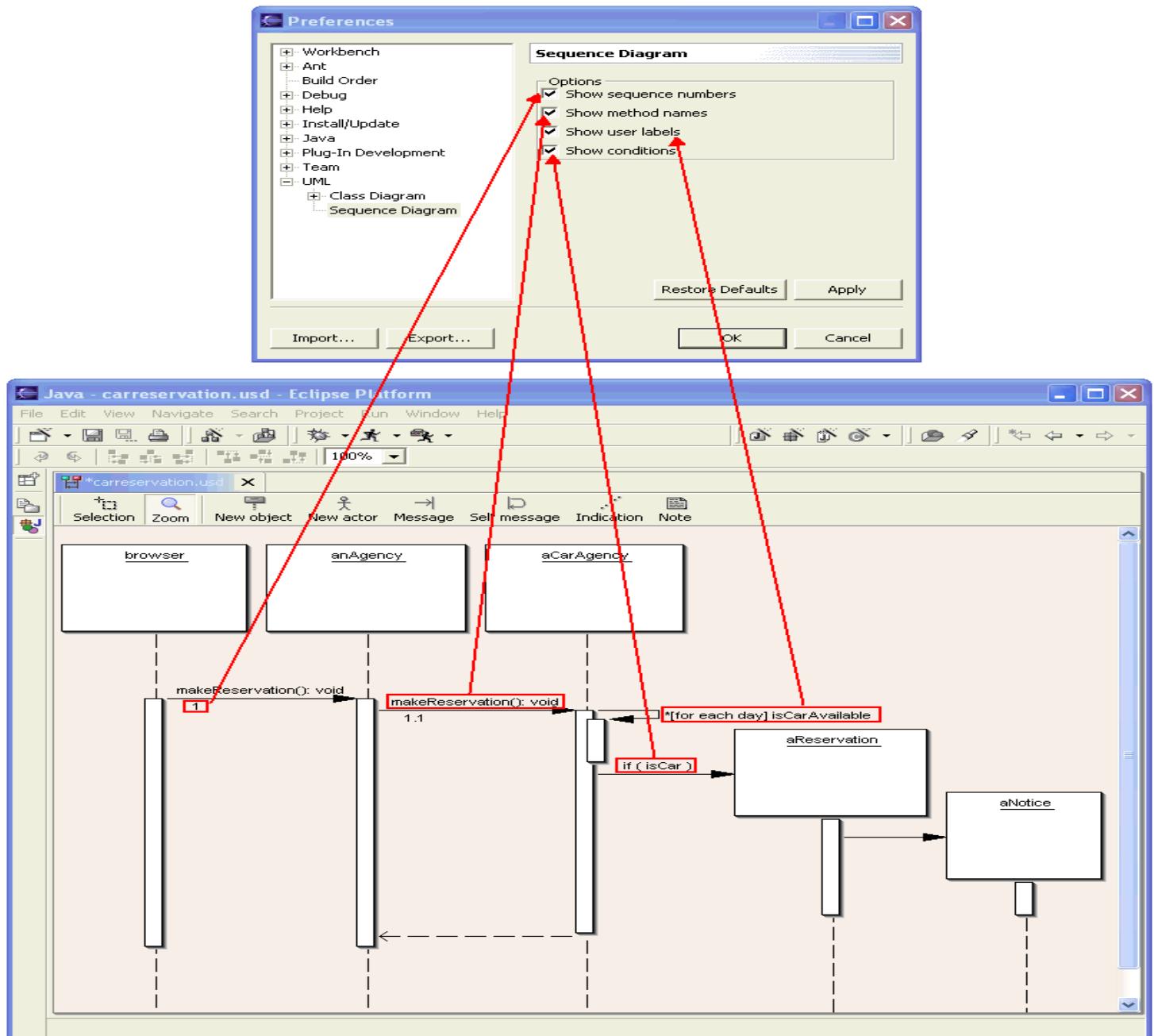
Sequence Diagram

The Sequence Diagram Preferences allow you to choose sequence diagram options. Four options are available and will help the modeler to show what is really important inside their diagram.

1. Show sequence numbers
2. Show method name
3. Show user labels
4. Show conditions

We can select each option in the Sequence Diagram window and click on Apply button in order to see the live result in the diagram editor.

The following pictures show the sequence diagram preferences options and their relation to the sequence diagram editor.



XMI 2.2 Editor

XMI 2.1 Editor

- [Activate the XMI 2.1 Editor](#)
- [Navigate in the XMI 2.1 Editor](#)
- [Java metamodel elements](#)
- [UML metamodel elements](#)
- [Multiple projects model](#)

Activate the Model Editor

To activate the UML XMI 2.1 Editor you need to:

1. Create a Java, JEE, JPA or other Eclipse project
2. Create an UML diagram (class, usecase, state, activity, component or deployment diagram)
3. Launch the UML Model Editor

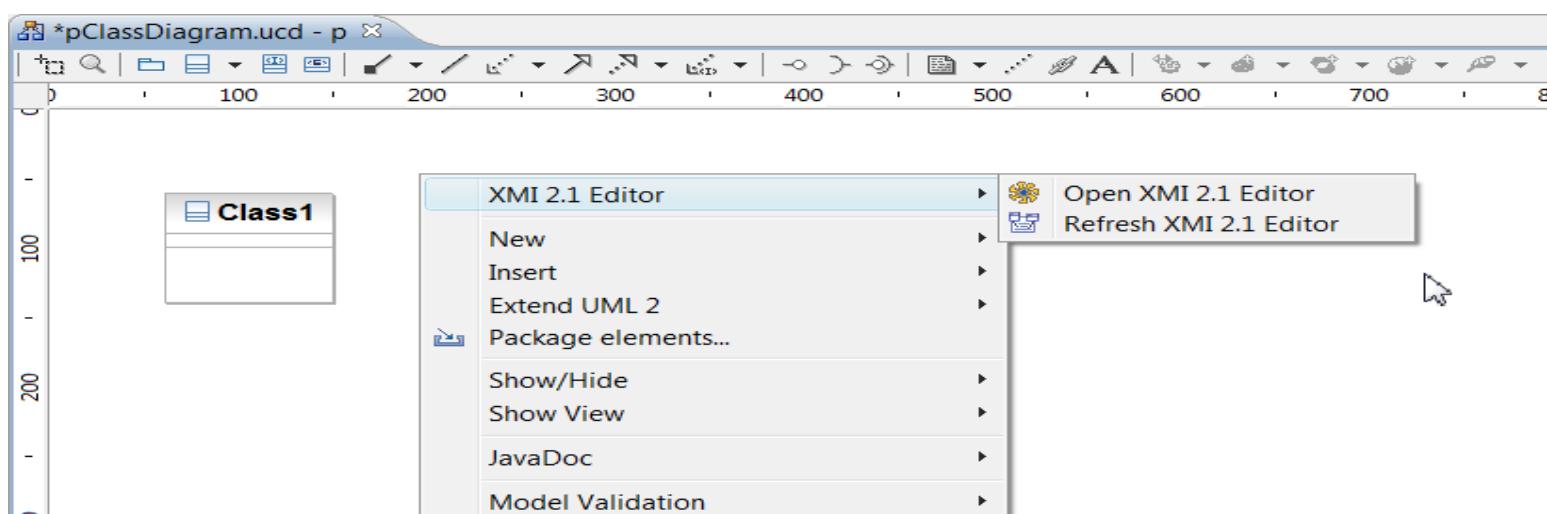
The XMI 2.1 Editor is an instance of the UML Editor and of UML 2 metamodel.

It means that to open the XMI Editor, you need to have activated the instance of the UML 2.1 metamodel inside your project.

The XMI editor is only working at project level. It is not possible to use more than one project or to divide the UML 2.1 metamodel.

Once the UML 2.1 metamodel instance is activated, you can open the XMI editor by using the diagram contextual menu.

Click on the diagram background > XMI Editor > Open XMI Editor

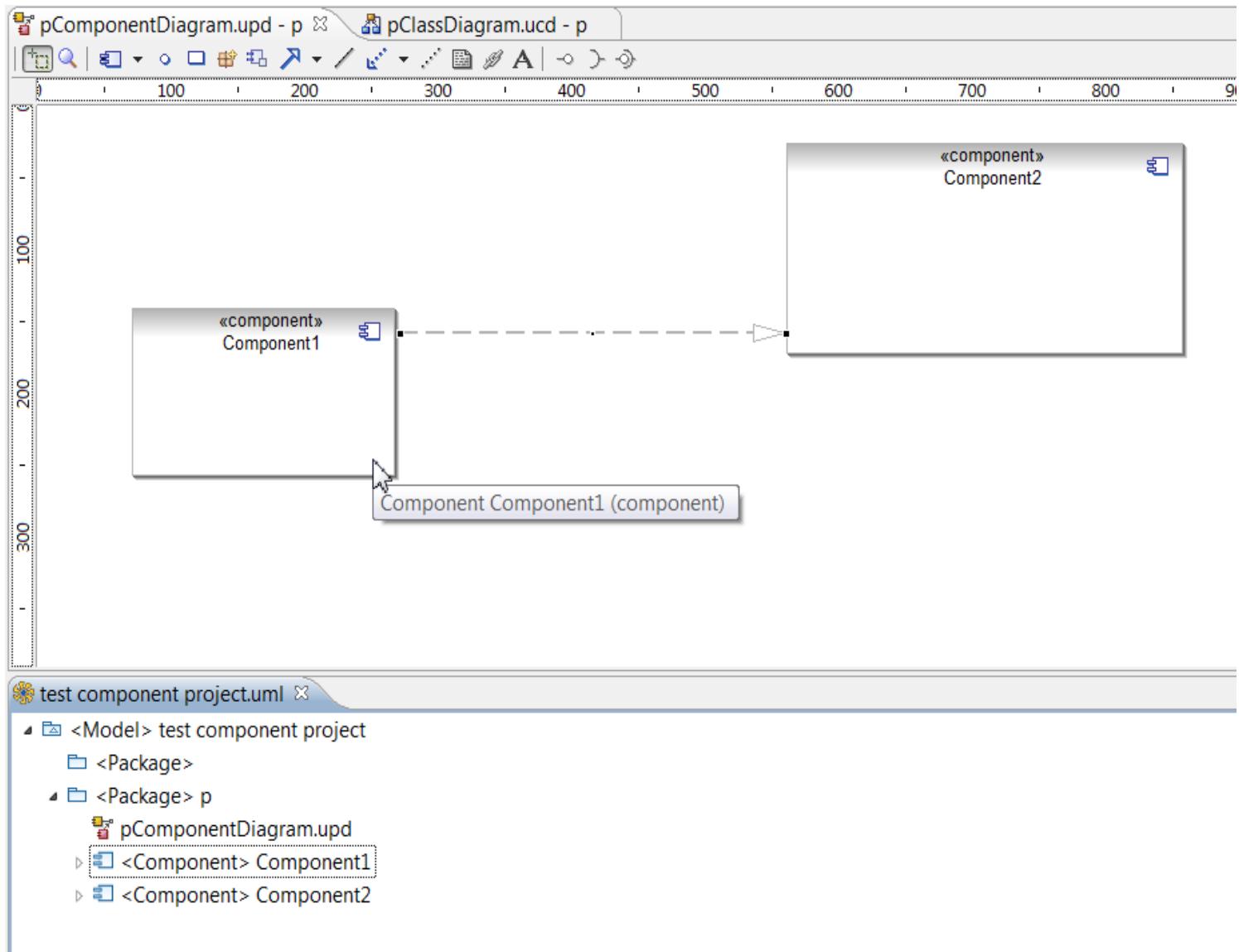


Navigate in the Model Editor

You can navigate in the XMI 2.1 Editor by selecting an UML element in one of the following diagram:

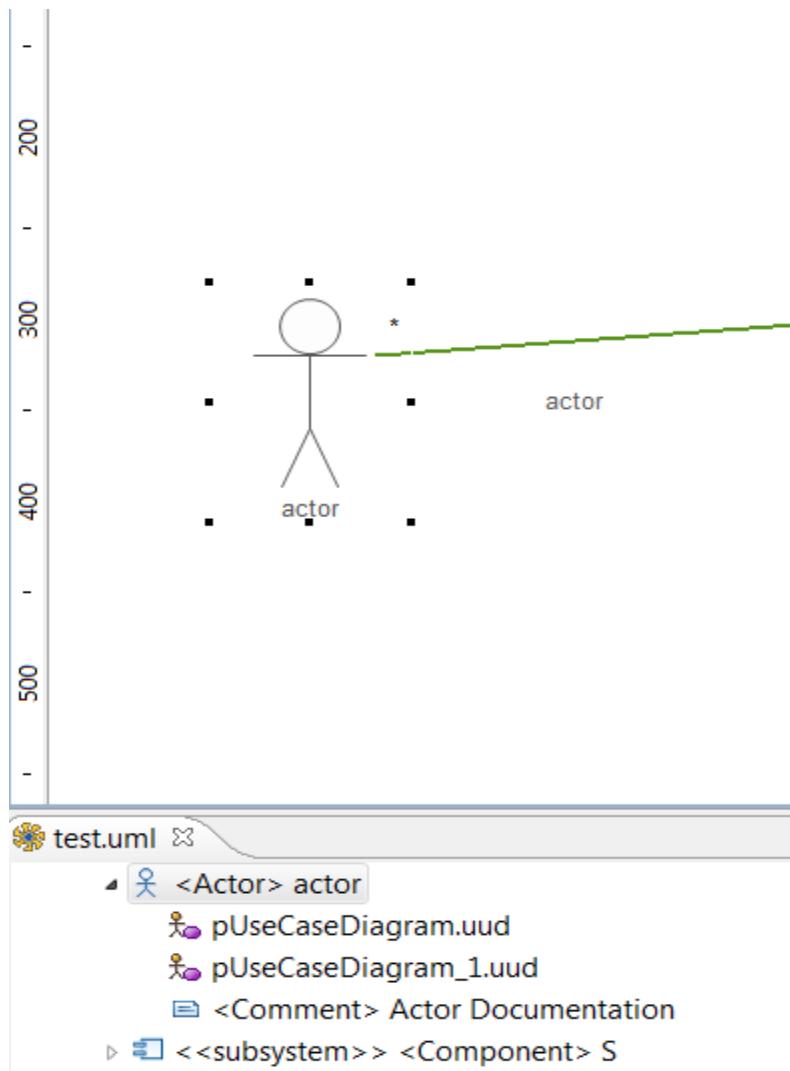
- Class Diagram
- UseCase Diagram
- StateChart Diagram
- Activity Diagram
- Component Diagram
- Deployment Diagram
- Sequence Diagram (*coming soon*)
- Composite Structure Diagram (*coming soon*)

If you click on an element in the UML Editor, then the XMI 2.1 Editor will immediately show the metamodel view of this element.



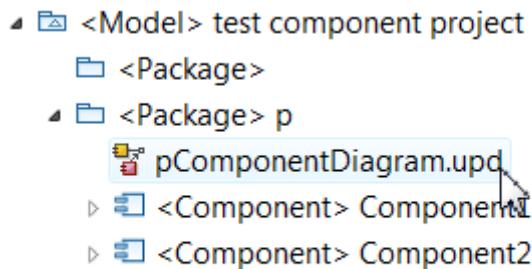
If the UML element is used in more than one diagram, then you will see the UML Editor diagram name at the root of the UML element.

In this example actor is the same UML element and is used in two different diagrams (e.g. *pUseCaseDiagram* and *pUseCaseDiagram_1*)



If you double click on the eannotation representing the UML diagram, then you will open the associated UML Editor diagram.

In this example the double click will open the component diagram (*you can notice that the icon is depending on the type of diagram*)



Java metamodel element

You can change, add or delete metamodel elements using the XMI 2.1 Editor view, the property view, the Package Explorer or the UML Editor.

In order to understand the metamodel concept you should consider two kinds of elements (e.g. Java and non Java element).

- If you use the Package Explorer or the UML Editor Menu, then all created elements in the class diagram will be Java elements.
- If you use the XMI 2.1 Editor, then all new created elements will only be UML elements (*e.g. no Java code*)

1. Java elements

1. [Change](#)
2. [Add UML inside Java Class](#)
3. [Delete](#)

In this example we have a Java Class (*e.g. Class1*) which is displayed in:

1. the Package Explorer
2. the UML Editor
3. the property view
4. the model editor

If you update the Class information in one of the view, then all views will be updated.

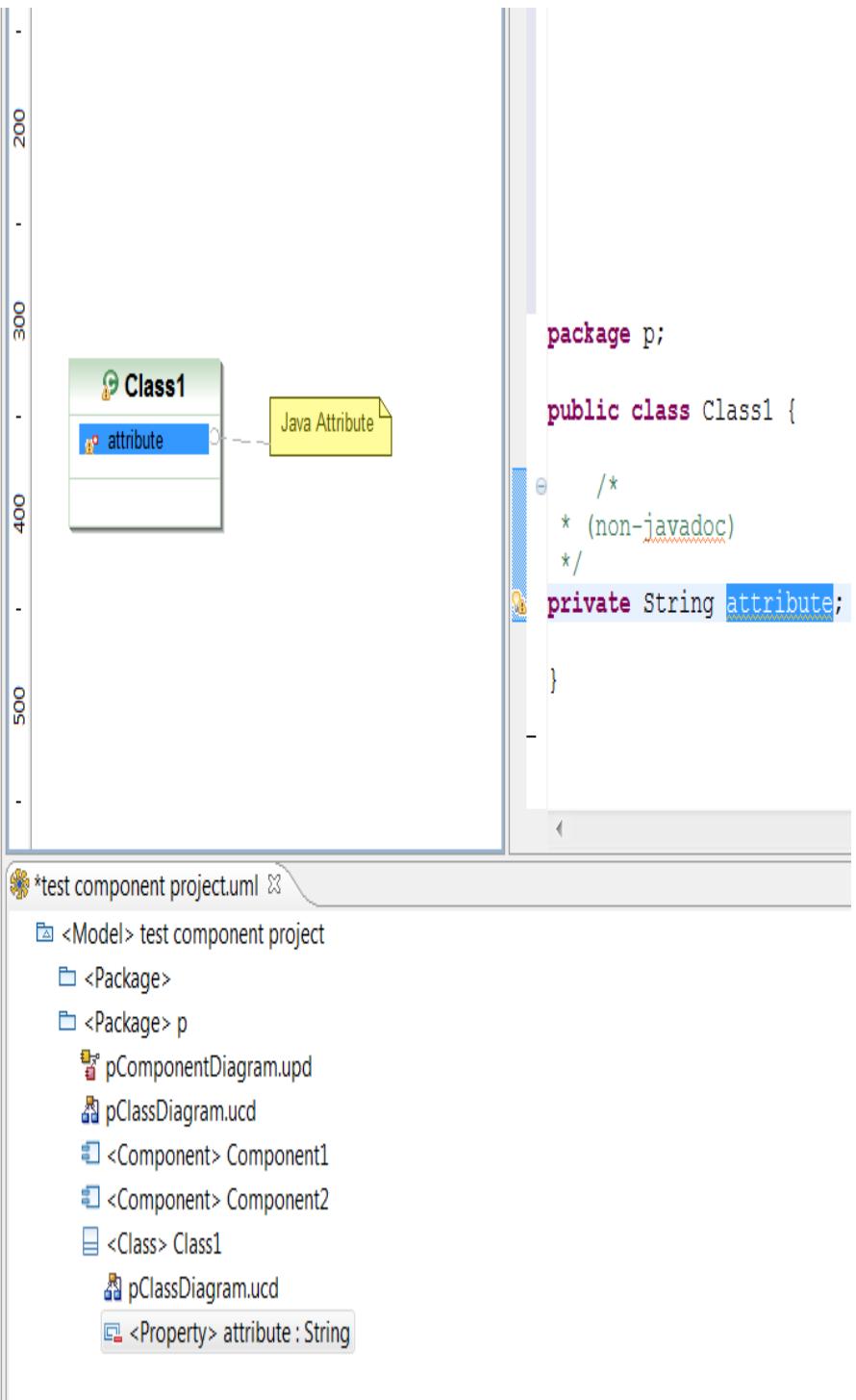
1. Change Java Element

In this example we are going to change attribute visibility directly in the Property View.

Note that the Java code is updated as well as Package Explorer, UML Editor, Property View and XMI 2.1 Editor.

If the change the visibility of the following attributes from Private to Public:

Property	Value
UML	
Aggregation	None
Association	
Class	<Class> Class1
Client Dependency	
Default	
End	
Is Derived	false
Is Derived Union	false
Is Leaf	false
Is Ordered	false
Is Read Only	false
Is Static	false
Is Unique	true
Lower	1
Name	attribute
Redefined Property	
Subsetted Property	
Template Parameter	
Type	<Class> String
Upper	1
Visibility	Private



By just changing the Visibility in the Properties view we immediately get the following:

The screenshot shows a UML modeling environment with several windows open:

- Project Explorer:** Shows files like Class1.java, Class1, attribute, pClassDiagram.ucd, JRE System Library [jdk1.5.0_14], and test component project.uml.
- Properties View:** Displays properties for a selected element (Class1). Key settings include:
 - Visibility: Public
 - UML:
 - Aggregation: None
 - Association: <Class> Class1
 - Client Dependency
 - Default
 - End
 - Is Derived: false
 - Is Derived Union: false
 - Is Leaf: false
 - Is Ordered: false
 - Is Read Only: false
 - Is Static: false
 - Is Unique: true
 - Lower: 1
 - Name: attribute
 - Type: <Class> String
 - Upper: 1
- Class Diagram:** Shows a Class1 element with an attribute named "attribute". A yellow callout box labeled "Java Attribute" points to the attribute name in the class definition.
- Java Editor:** Displays the generated Java code:

```
package p;

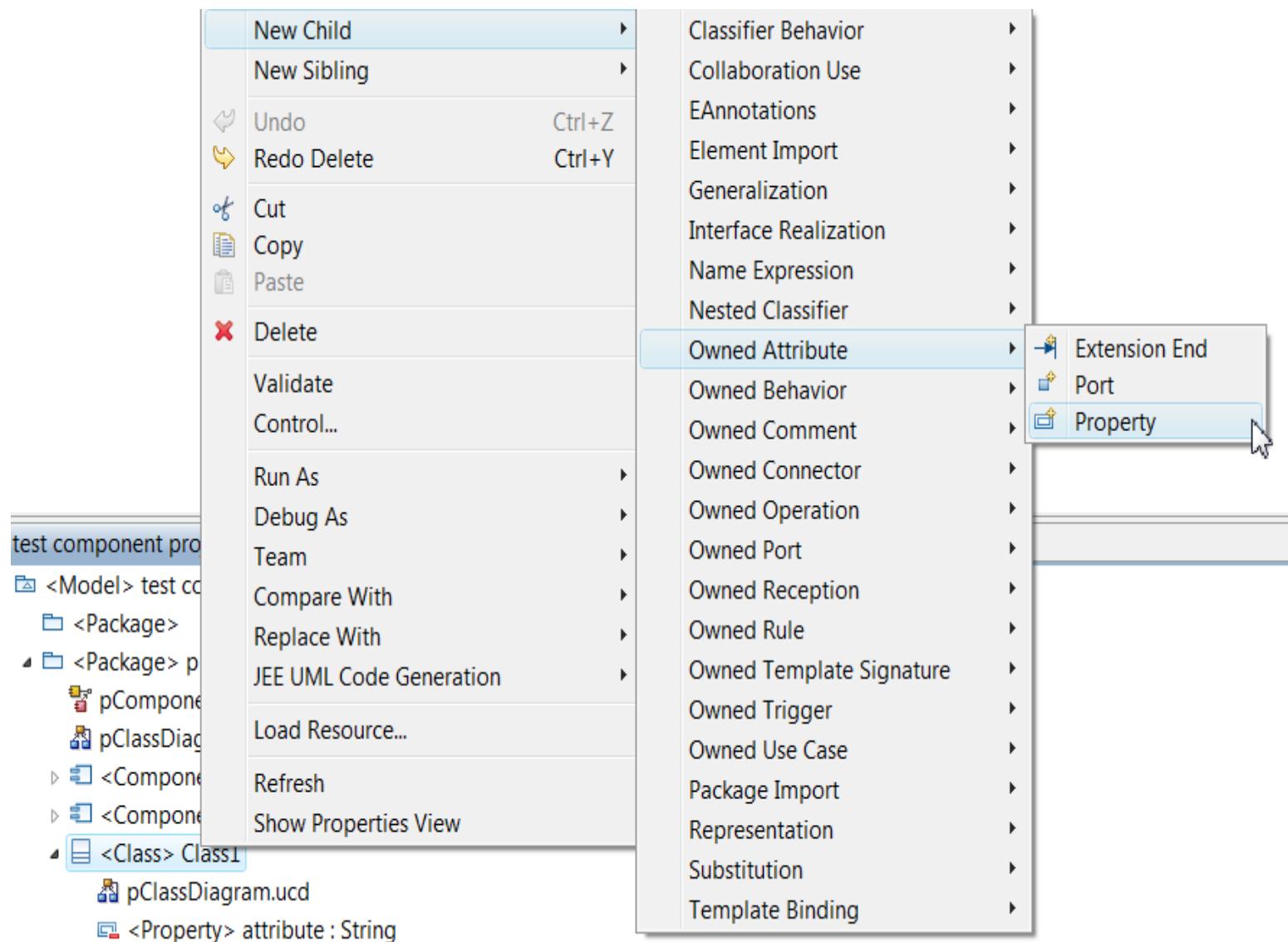
public class Class1 {

    /**
     * (non-javadoc)
     */
    public String attribute;
}
```
- Outline View:** Shows the structure of the test component project, including packages, components, and classes.

2. Add UML element to a Java Class, Interface or Enumeration

For example we are going to extend a Java Class by adding a UML property.

Select the class in the **XMI 2.1 Editor > New Child > Owned Attribute > Property**



The new UML property will be added in your XMI 2.1 Editor and in your UML Editor.

The Package Explorer will not be updated because not java code has been created.

Note that you can **add a name using the Properties View Name Field**

test component project

- src
- p
 - Class1.java
 - Class1
 - attribute
 - pClassDiagram.ucd
- JRE System Library [jdk1.5.0_14]
- test component project.uml

Properties

Property	Value
UML	
Aggregation	None
Association	
Class	<Class> Class1
Client Dependency	
Default	
End	
Is Derived	false
Is Derived Union	false
Is Leaf	false
Is Ordered	false
Is Read Only	false
Is Static	false
Is Unique	true
Lower	1
Name	UML Attribute
Redefined Property	
Subsetted Property	
Template Parameter	
Type	
Upper	1
Visibility	Public

```

classDiagram
    class Class1 {
        attribute
    }
  
```

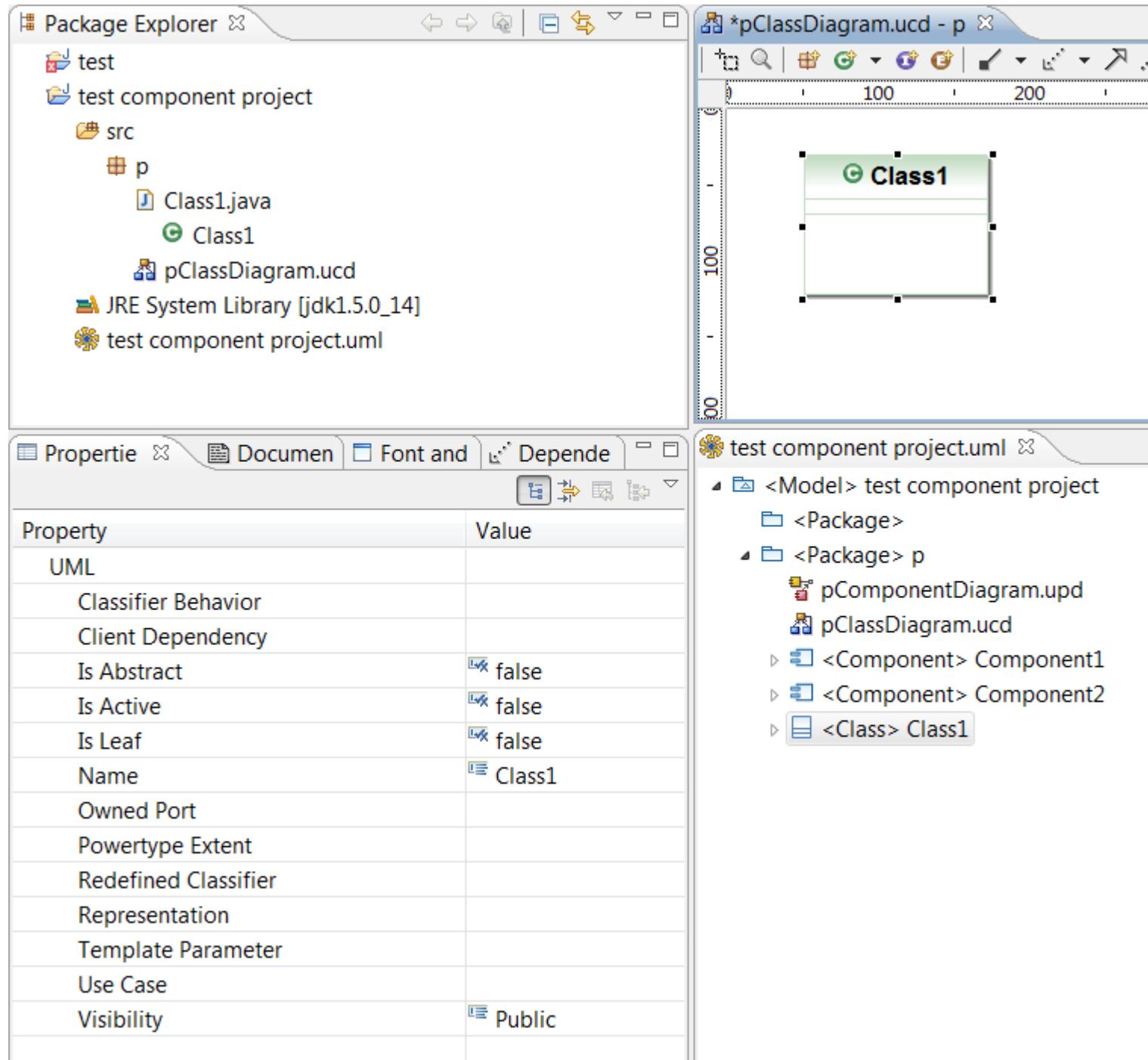
*test component project.uml

- <Model> test component project
- <Package>
- <Package> p
 - pComponentDiagram.upd
 - pClassDiagram.ucd
 - <Component> Component1
 - <Component> Component2
 - <Class> Class1
 - pClassDiagram.ucd
 - <Property> attribute : String
 - <Property> UML Attribute

3. Delete a Java element:

If you delete a Java element from:

- the XMI 2.1 Editor, then only the metamodel will be changed and not the Java Code.
- the Java Code, then the metamodel will also be updated



UML metamodel element

You can change, add or delete metamodel elements using the XMI 2.1 Editor view, the property view or the UML Editor.

In order to understand the metamodel concept you should consider two kinds of elements (e.g. Java and non Java element).

- If you use the Package Explorer or the UML Editor Menu, then all created elements in the class diagram will be Java elements.
- If you use the XMI 2.1 Editor, then all new created elements will only be UML elements (e.g. *no Java code*)

1. UML elements

1. Add a new UML element
 1. [Add a UML Element in the class diagram](#)
 2. [Add a UML Element in the other diagrams](#)
2. [Change](#)
3. [Delete](#)

1.1 Add a new UML Element in a class diagram

You can add new UML element which belongs to the class diagram category (e.g. Class, Interface and Enumeration) by just a drag and drop from the XMI 2.1 Editor.

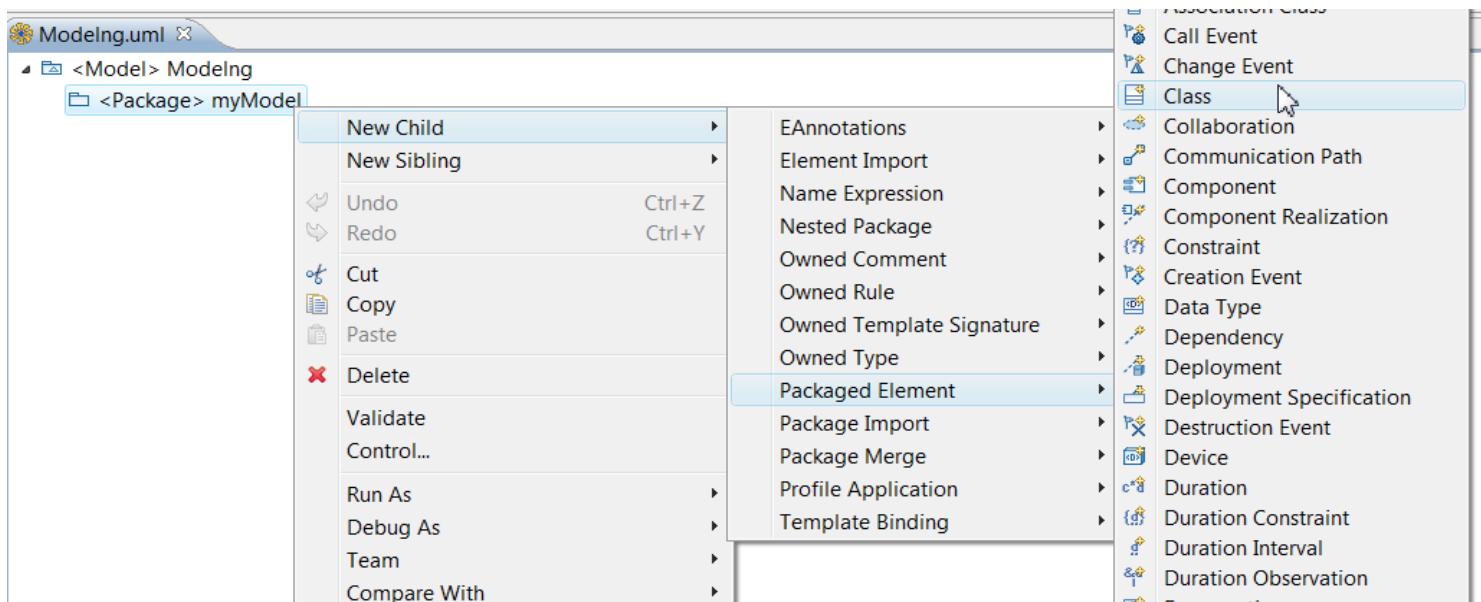
The UML element doesn't generate Java code and can be mixed with existing Java Classes in the same class diagram.

In order to add new UML elements in the class diagram you should use the drag and drop from the XMI 2.1 Editor to the UML Editor feature.

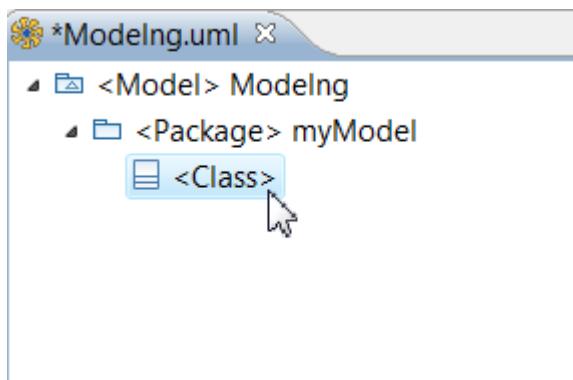
Only the XMI 2.1 Editor can generate UML classes for the class diagram (e.g. *the class diagram on the toolbar is still generating only Java elements*).

To add a new UML Class in the class diagram.

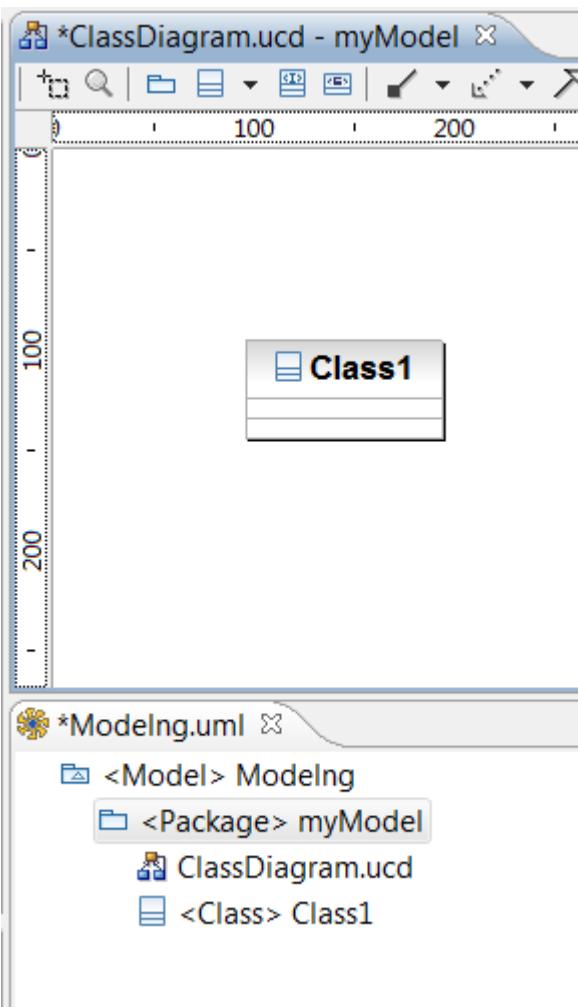
First create a new class in the model editor by selecting **New Child > Packaged Element > Class**



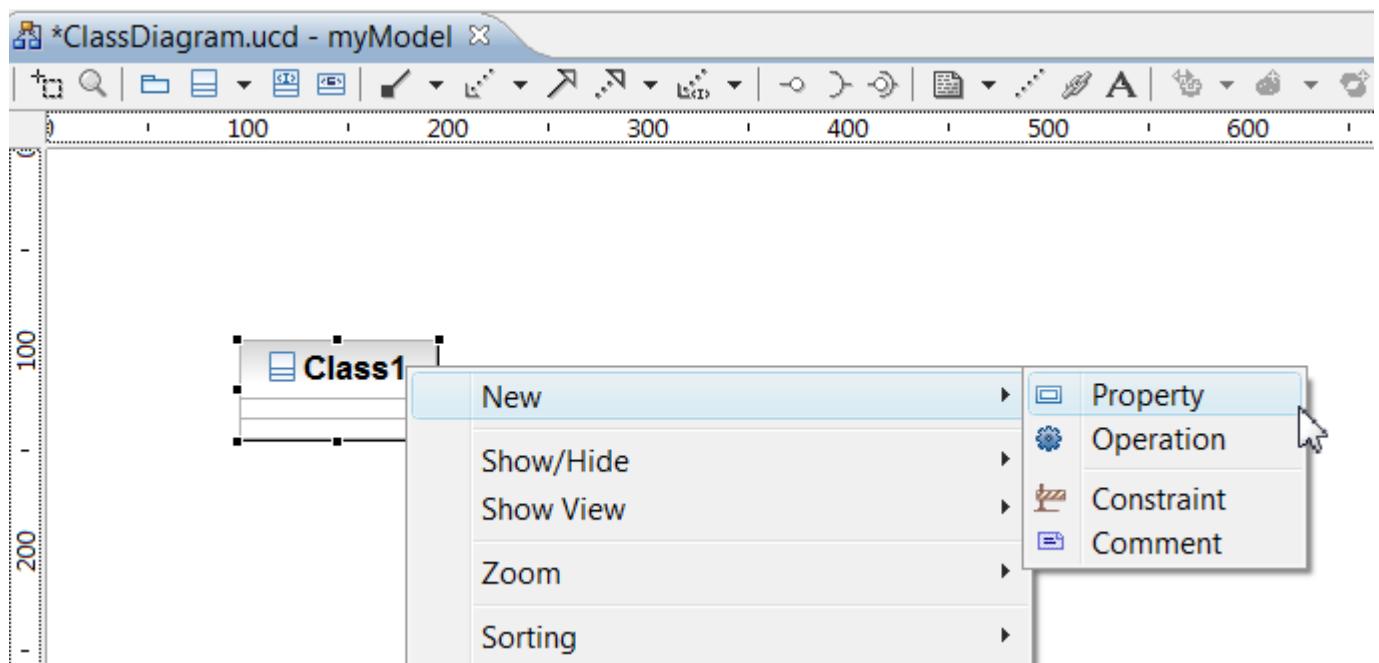
The Class has been created in the XMI 2.1 Editor



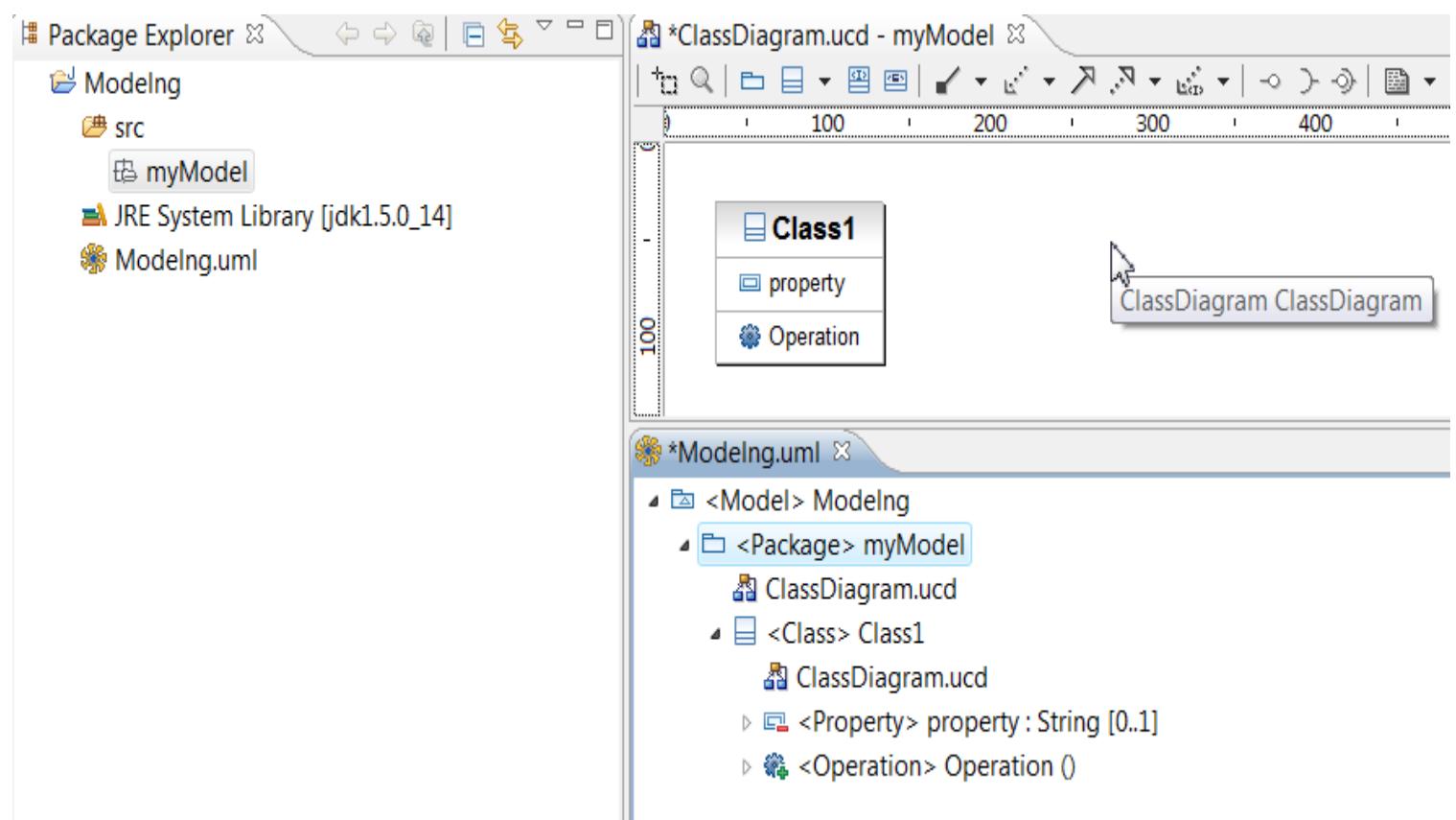
Drag and drop the Class from the XMI 2.1 Editor to the UML Editor.
You can notice that the UML Class borders are black.



You can add new UML Property or Operation to the Class.
Click on the Class > New > Property



The created Class is only a metamodel Class and doesn't have any Java code.



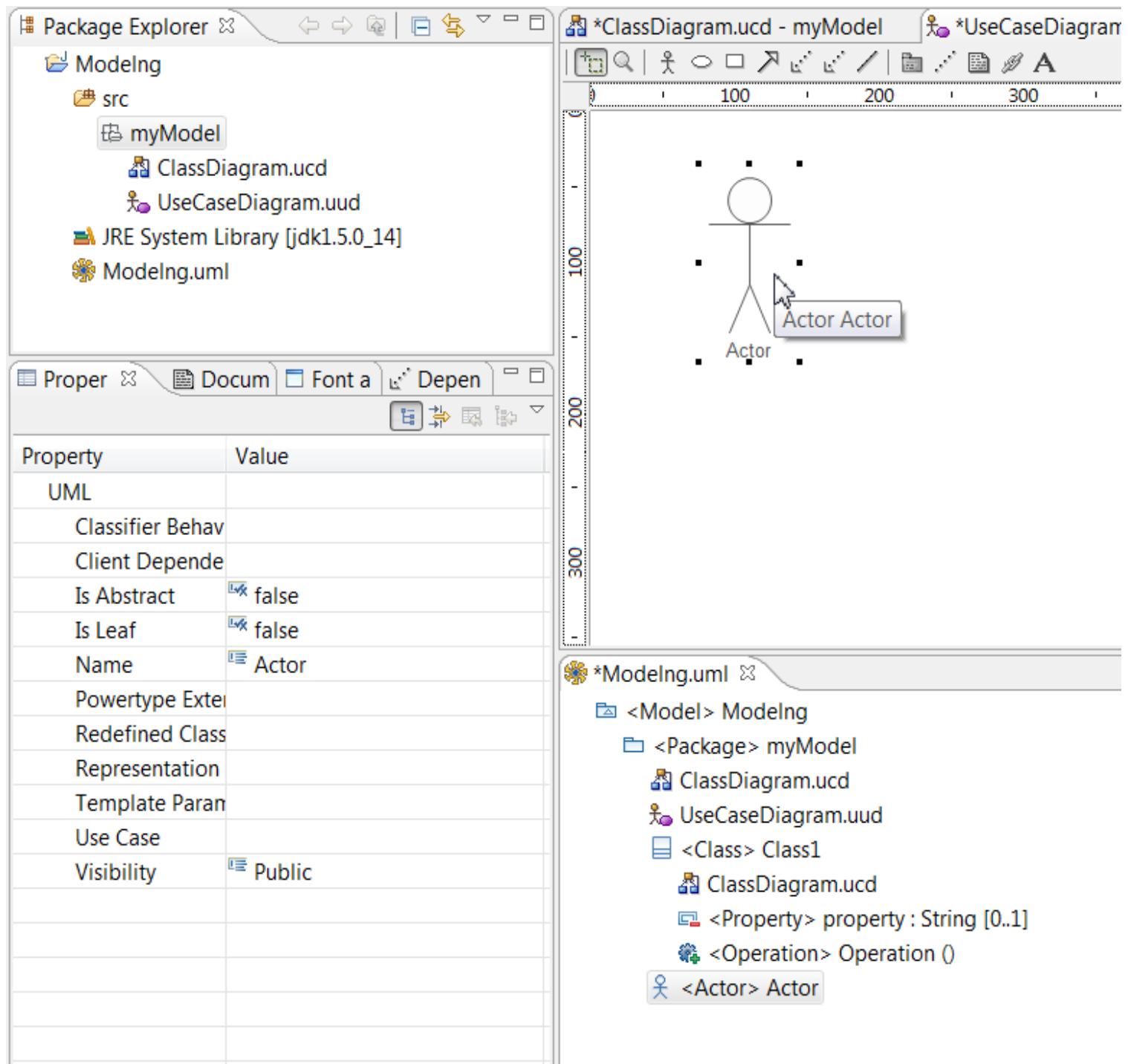
1.2 Add new UML element in UML diagram (except class diagram)

The Omondo toolbar will immediately synchronize the UML editor and the XMI 2.1 Editor.

EclipseUML is using the same metamodel and instance, therefore each change in the editor is immediately updated in the metamodel.

Note that the Class diagram toolbar is generating Java classes, and that the XMI 2.1 Editor is generating new UML Classifiers (e.g. Class, Interface, Enumeration...).

Other diagrams (e.g. not the Class diagram) toolbar are only generating UML metamodel elements, and no Java code. The following example show the Actor creation from the UseCase diagram Toolbar in a UseCase Diagram. You can notice that the metamodel & Properties View is immediately updated



2. How to change UML elements metamodel properties:

If you click on the UML element in the XMI 2.1 Editor then the Properties View will be immediately synchronized.

Changing any property in the Property view will immediately update the UML Editor and the XMI 2.1 Editor.

The following example shows how to rename Actor to UML Actor using the Properties View:

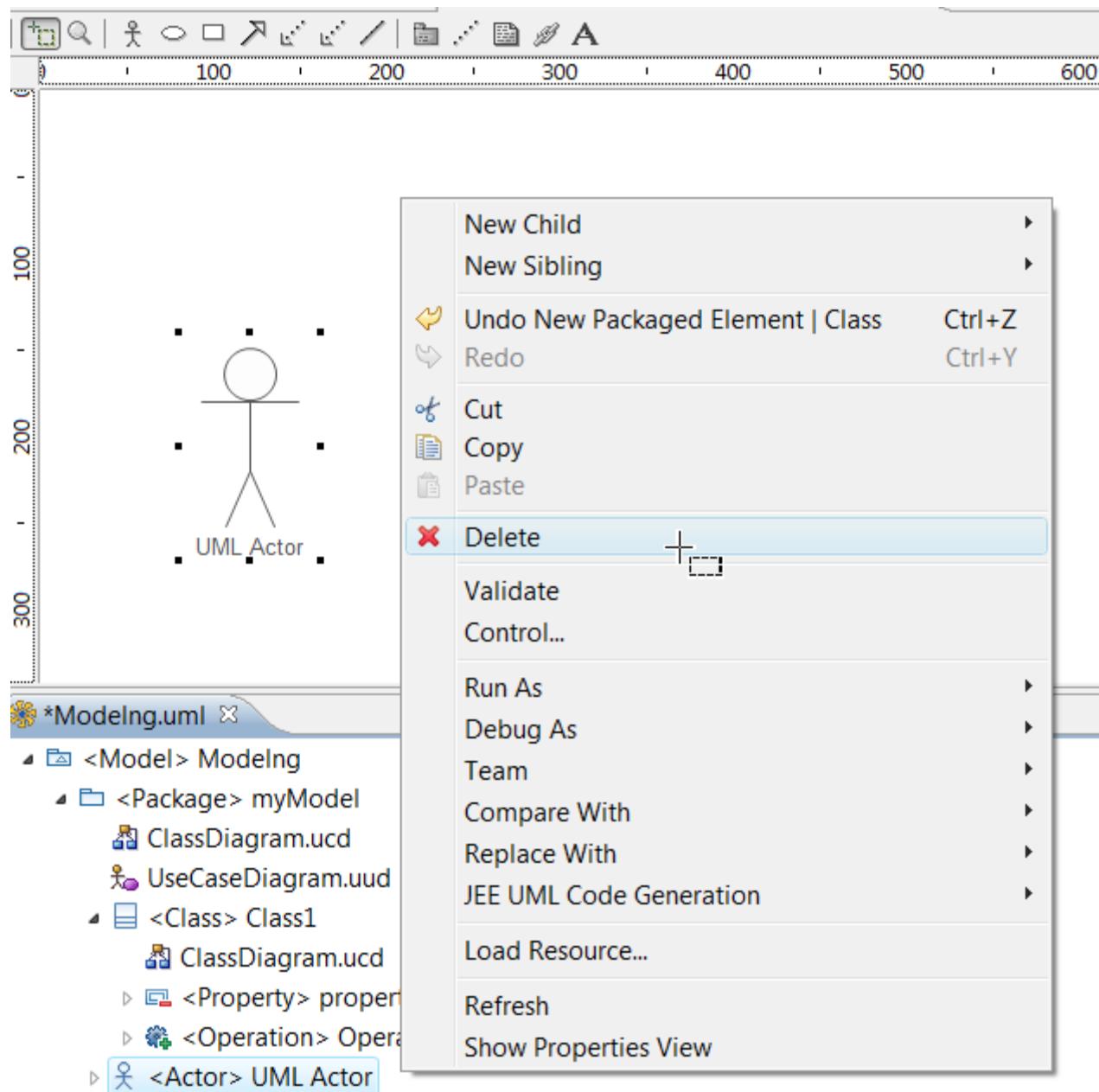
The screenshot displays the Eclipse Modeling Tools interface with the following components:

- Package Explorer:** Shows the project structure with a package named "myModel" containing "ClassDiagram.ucd" and "UseCaseDiagram.uud". It also lists "JRE System Library [jdk1.5.0_14]" and "Modeling.uml".
- Properties View:** Shows the properties of the selected UML Actor element. The "Name" property is currently set to "UML Actor". Other properties listed include Classifier Behavior, Client Dependency, Is Abstract (false), Is Leaf (false), Powertype Extent, Redefined Classifier, Representation, Template Parameter, Use Case, and Visibility (Public).
- UML Editor:** Displays a UML actor diagram with the label "UML Actor".
- XMI 2.1 Editor:** Shows the XML representation of the model, including the class "ClassDiagram.ucd", the use case "UseCaseDiagram.uud", the class "Class1" with its associated properties and operations, and the actor "UML Actor".

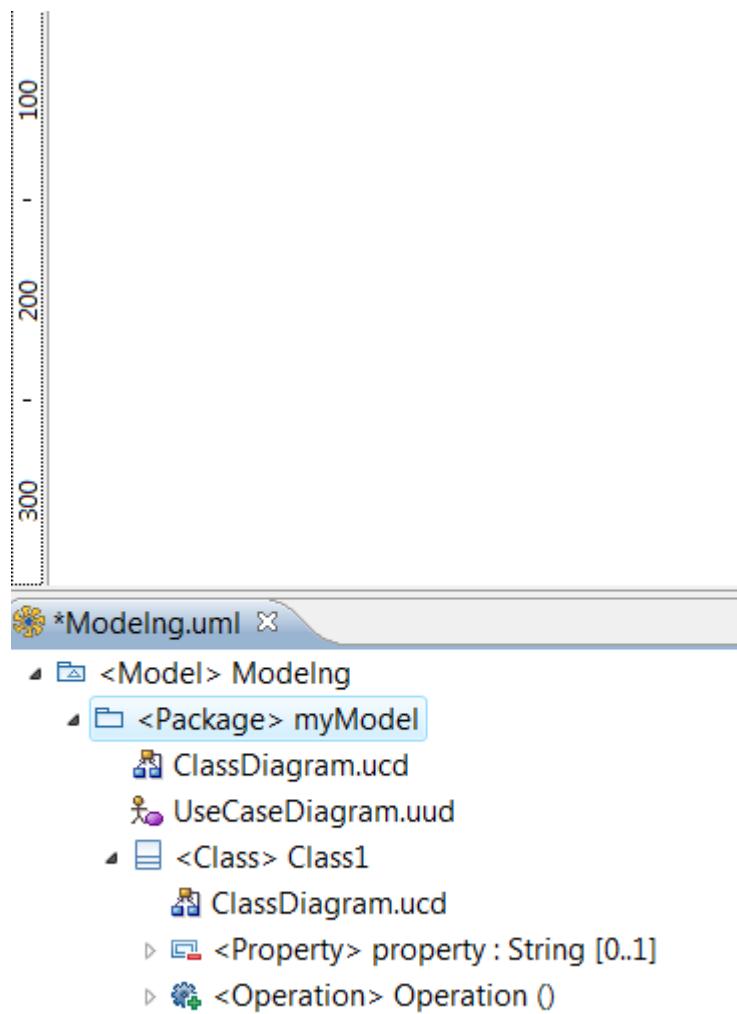
3. How to delete UML Metamodel element:

If you click an element in the Model Editor, you can then delete this element from the metamodel. This element will also be automatically deleted from the UML Editor.

Select UML Actor in the **XMI 2.1 Editor > Delete**



The UML Editor is immediately synchronized and the UML Actor element is deleted from the Model and UML Editor.



Multiple projects model

EclipseUML allows multiple projects modeling.

It means that you can merge inside an existing EclipseUML project one to many models coming from:

- EclipseUML
- Topcased
- Eclipse Modeling Tools project
- RSA 7

The merge is immediately completed at the first time that you drag and drop a model element coming from another project inside an EclipseUML diagram.

It works for all diagrams (e.g. class, usecase, state, activity, component etc...)

You can either create more than one project with EclipseUML or import different tools model inside EclipseUML.

These imported models will be merged inside EclipseUML Model.

Please note that Omondo needs an xmi 2.1 model in order to be able to merge with EclipseUML xmi 2.1 model.

We don't provide a transformation tool between XMI 1.x to XMI 2.1

You can use any transformation solution in order to get a standard UML 2.1 model from your existing models.

Many open source plugins are currently available doing this xmi transformation inside the Eclipse community.

If the generated xmi could be open by the Eclipse UML editor, then it could be used by EclipseUML.

To activate the UML XMI 2.1 multiple projects modeling you need to:

1. [Create a Java, JEE, JPA or other Eclipse project](#)
2. [Import your existing model inside a project](#)
3. [Extend the Omondo project property to your other projects](#)
4. [How to extend your model by adding other models](#)

1. How to create a Java Project?

Click in the package explorer **New > Java Project** then enter the name of your project in the Project name field and click on the Finish button.

2. How to import your model inside Eclipse?

If your model has not been created by Eclipse 3.4 Ganymede, then it is not possible to load the modeling tool inside Eclipse and get a graphical view (e.g. *Only XMI is available and could be used inside Eclipse 3.4 Ganymede*).

*A workaround is to create a project and then manually past and copy the model file in your project using the windows or Linux file manager. You click on your project in the **Package Explorer > Refresh** to see this new model in your project.*

If the model has been created by Eclipse Modeling Tools or Topcased then these models are immediately compatible with EclipseUML.

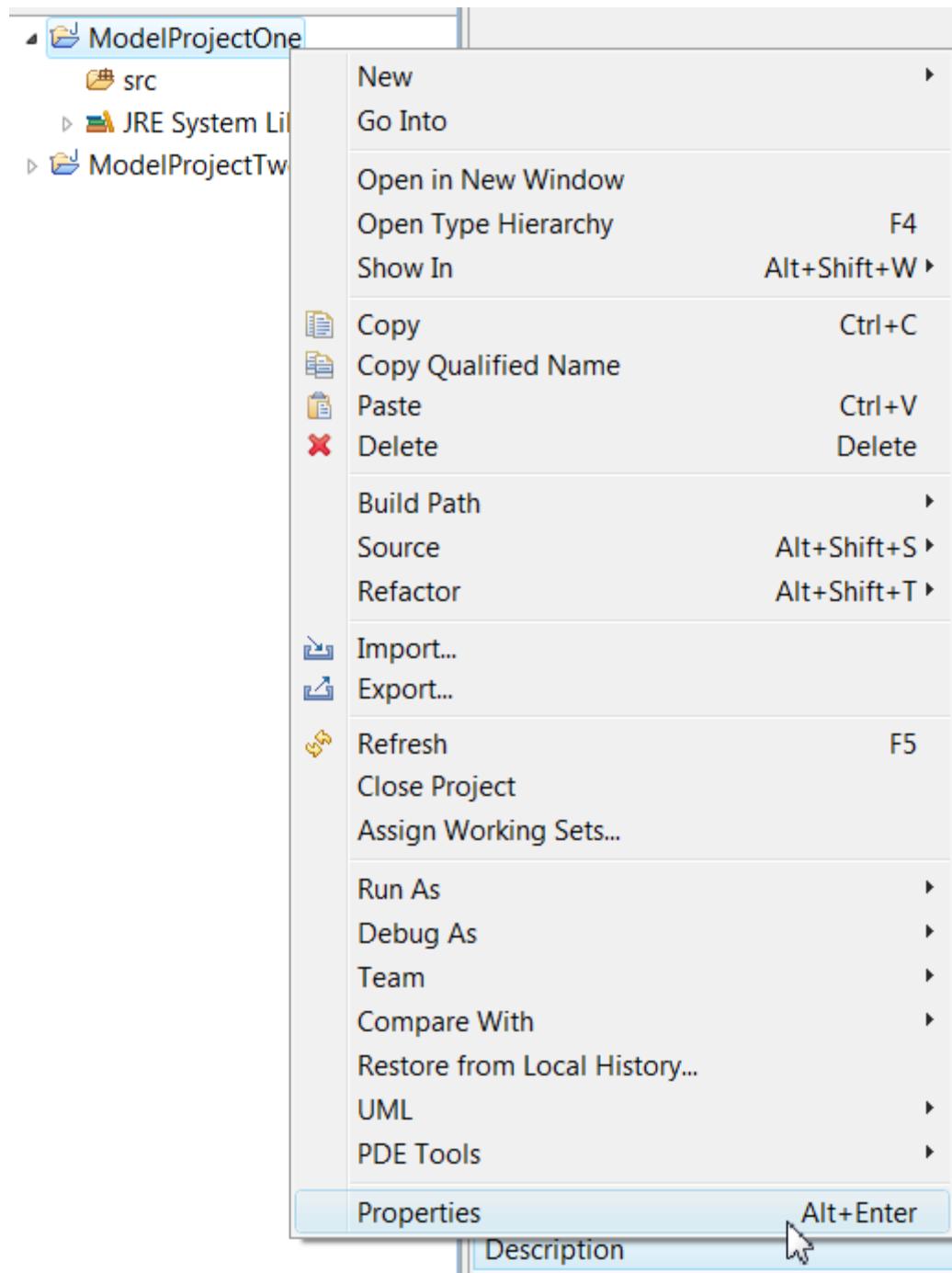
If the model has been created with Rose or RSA 6 then you need to transform your model to xmi 2.1 before loading the model in your project.

If the model has been created by RSA 7 then you just need to load it inside Eclipse 3.4 because the

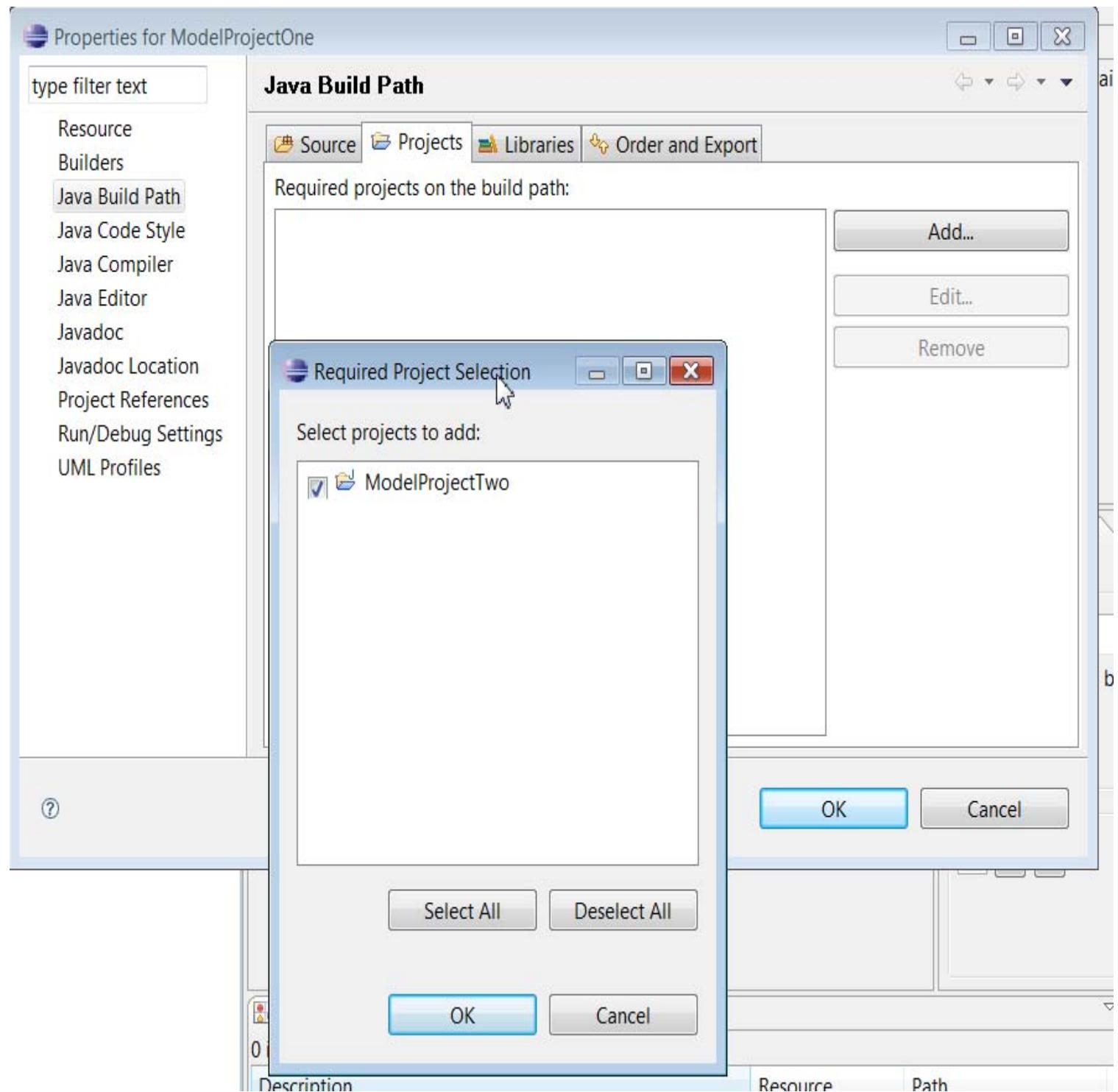
model is already xmi 2.1 compatible (*Note that RSA 7 tool is still not Eclipse 3.4 compatible*).

3. How to extend the Omondo project by adding an additional project?

Click on your **project > Properties > Java Build Path** and select the **Projects tab**.



Click then on the **Add... button** and select the projects (e.g. ModelProjectTwo) to be extended.

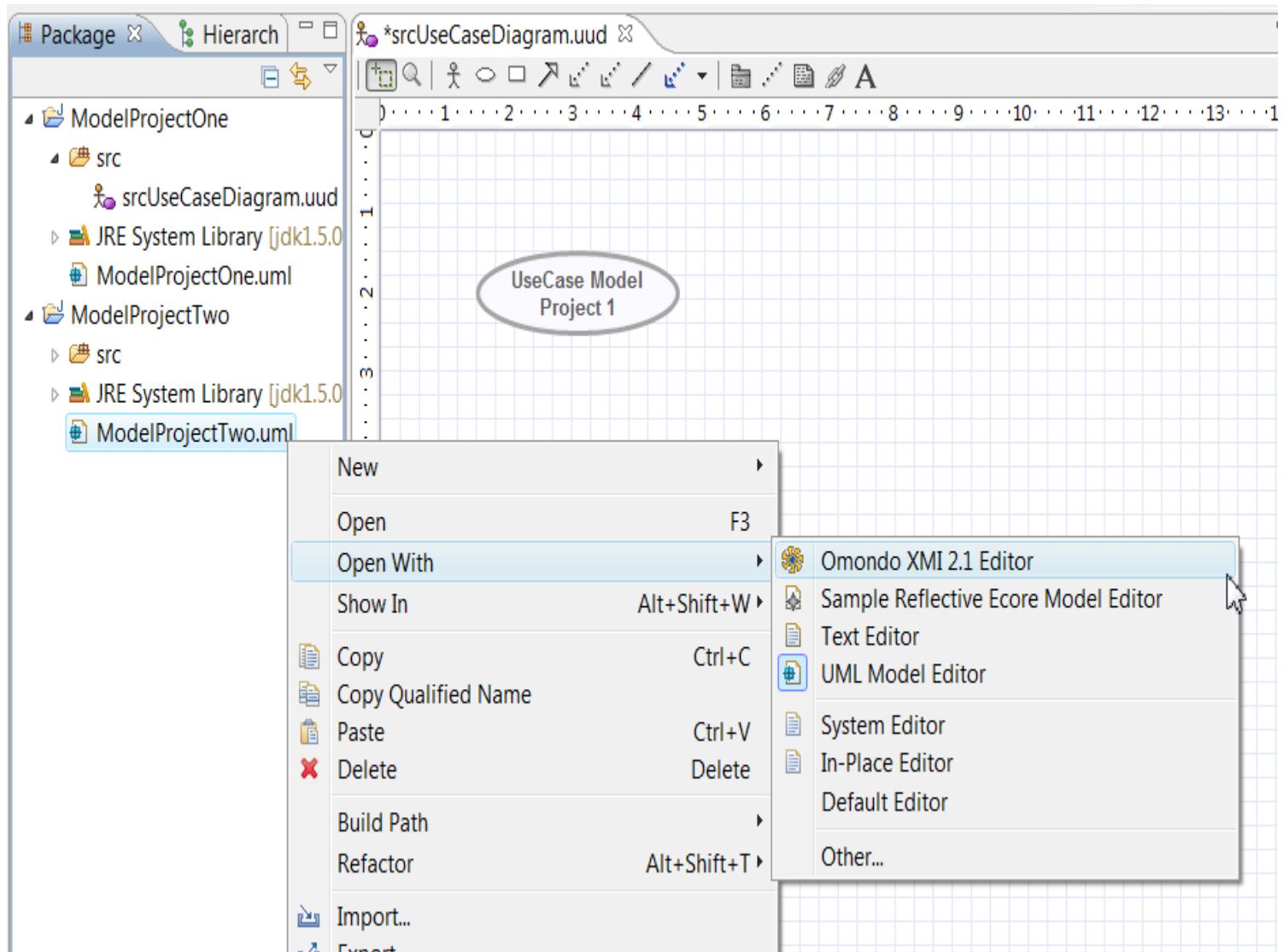


Omundo is using the Eclipse mechanism in order to keep the full power of eclipse plugin approach and to allow multiple model merges.

4. How to extend your EclipseUML model by adding other model elements?

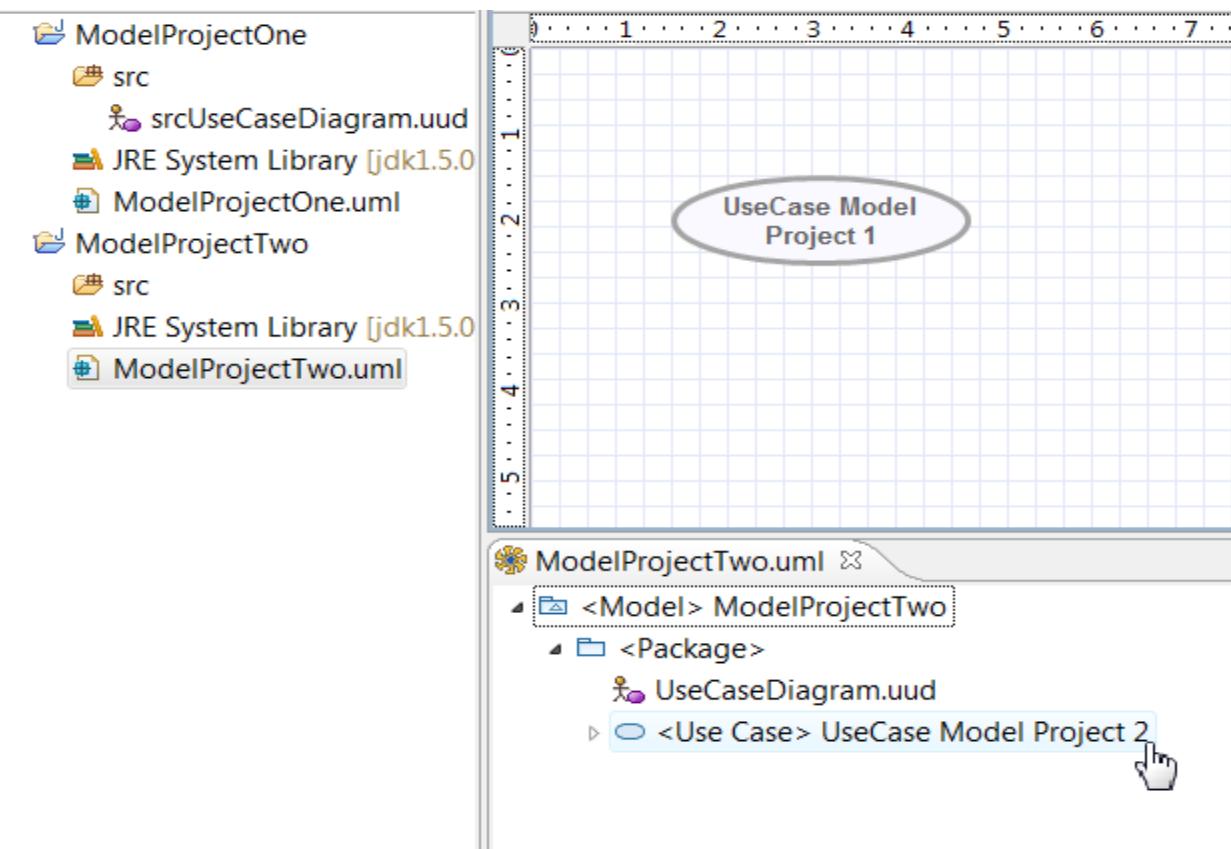
You need to open your model using the Omundo XMI Editor and then drag and drop this model element inside one EclipseUML diagram to get the model merge activated.

Click on the .uml file in your **Package Explorer** > **Open with** > **Omundo XMI 2.1 Editor**

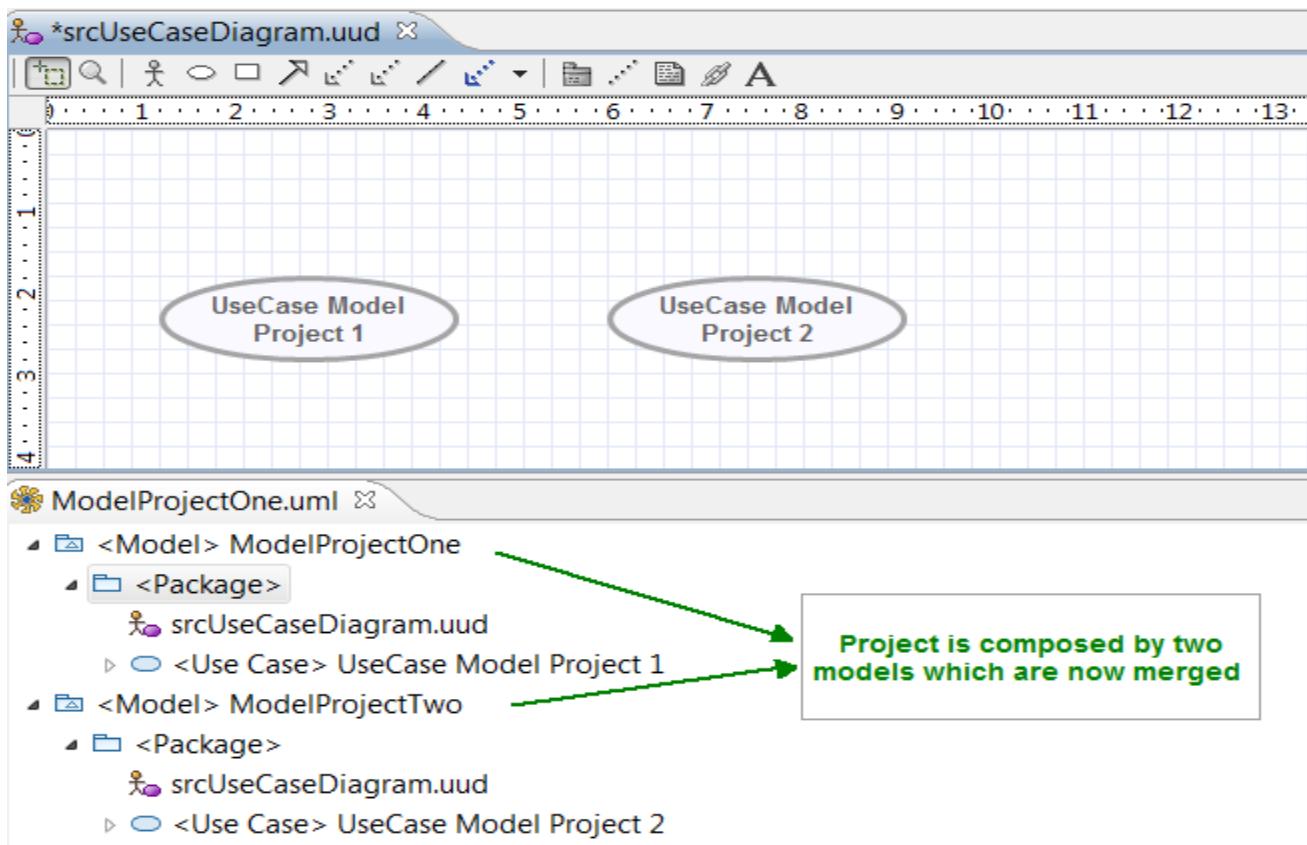


Drag the model element from the xmi source Editor to the central project model (e.g. EclipseUML target model where the merge will be completed)...

Select Usecase named UseCase Model Project 2 in the **XMI Editor** > **drag and drop inside the project one usecase diagram**



If you open the project ModelProjectOne and can see that the dropped element has been merged with existing ModelProjectOne model (*e.g. UseCase Model Project 2 has been added inside the project one model*).



Documentation Generation

This tutorial is designed to get you started on using the Omondo Documentation Generation plugin for Eclipse. The following areas are covered:

- [Launching the Documentation Generation](#)
- [Templates](#)
 - [Information](#)
 - [Content](#)
 - [Format](#)

Launching the Documentation Generation

The documentation generation can be launched from the pop-up menu or from the menu bar after selecting a java element in the package explorer. A wizard dialog appears.

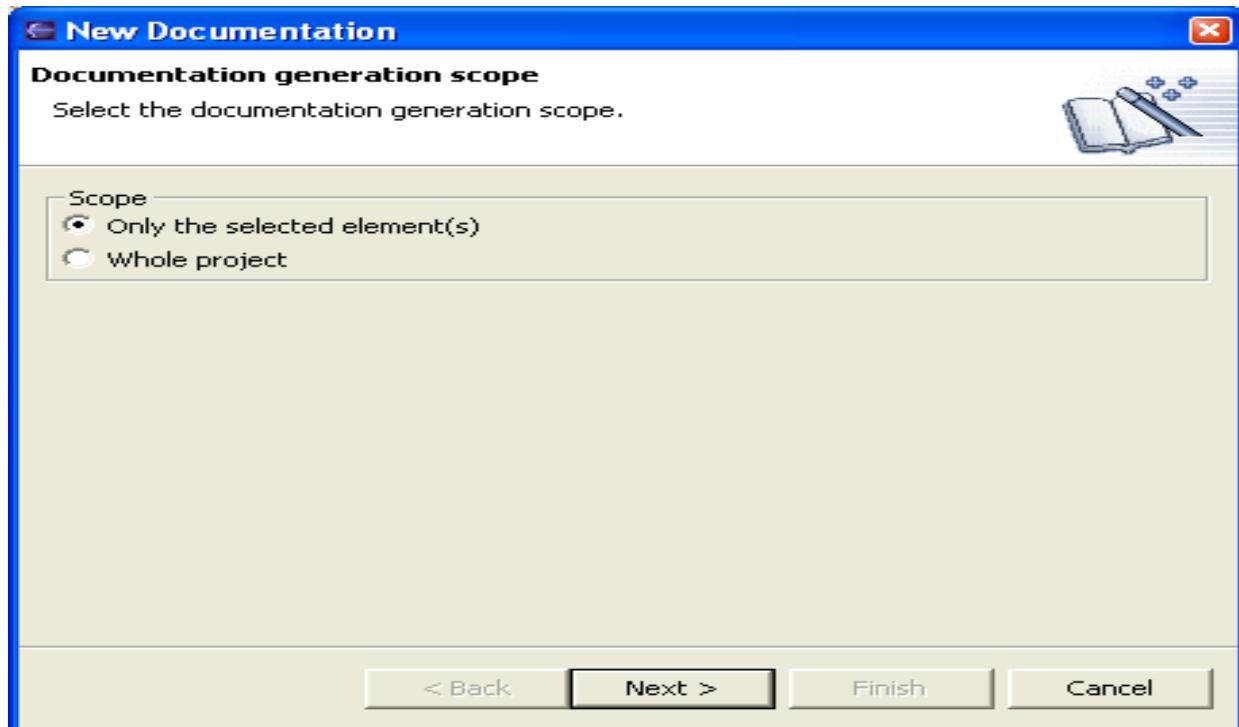
Selection

The documentation generation can be launched from the pop-up menu or from the menu bar after selecting a java element in the package explorer. The generated documentation will be dedicated either to this java element or to the whole java project. A wizard dialog appears.

Documentation Generation Wizard

Selection

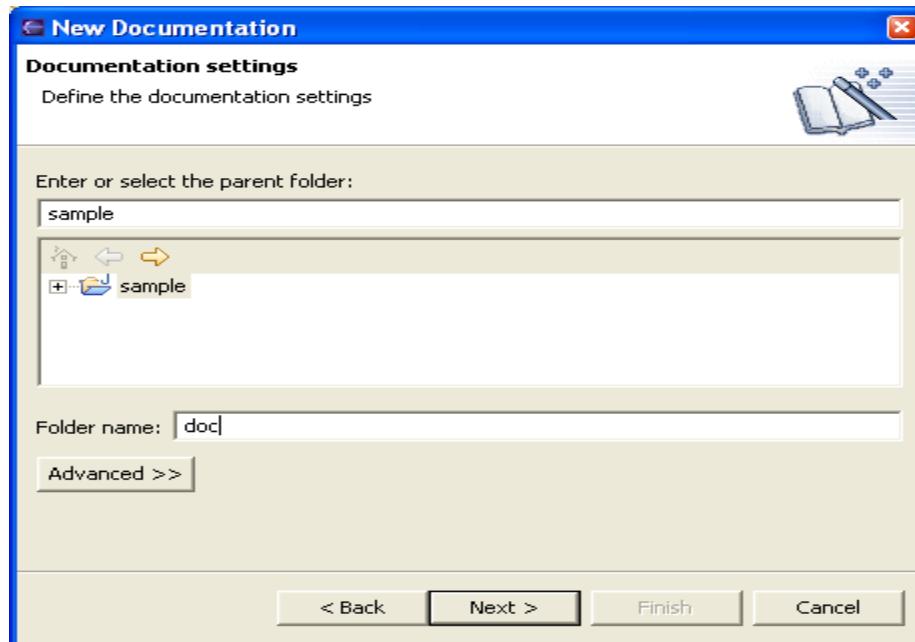
On the first step of the wizard, you can choose either to work on your selection or to work on the whole project.



Documentation directory

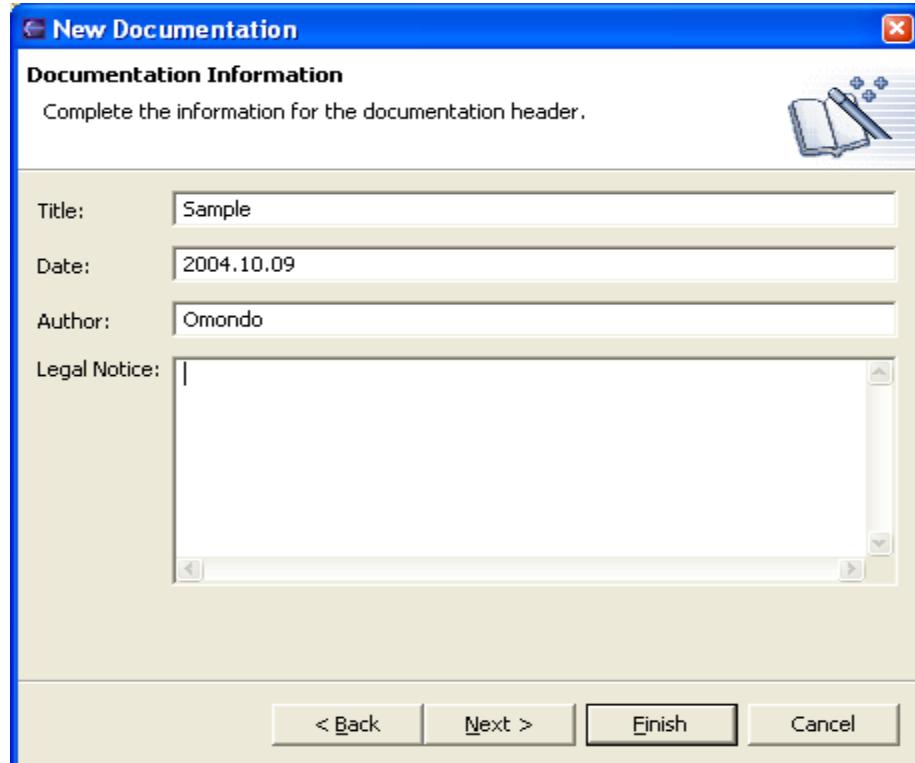
On the second step of the wizard, you must select the directory where the documentation will be generated.

It can be either an existing folder or a new one or a linked one.



Documentation information

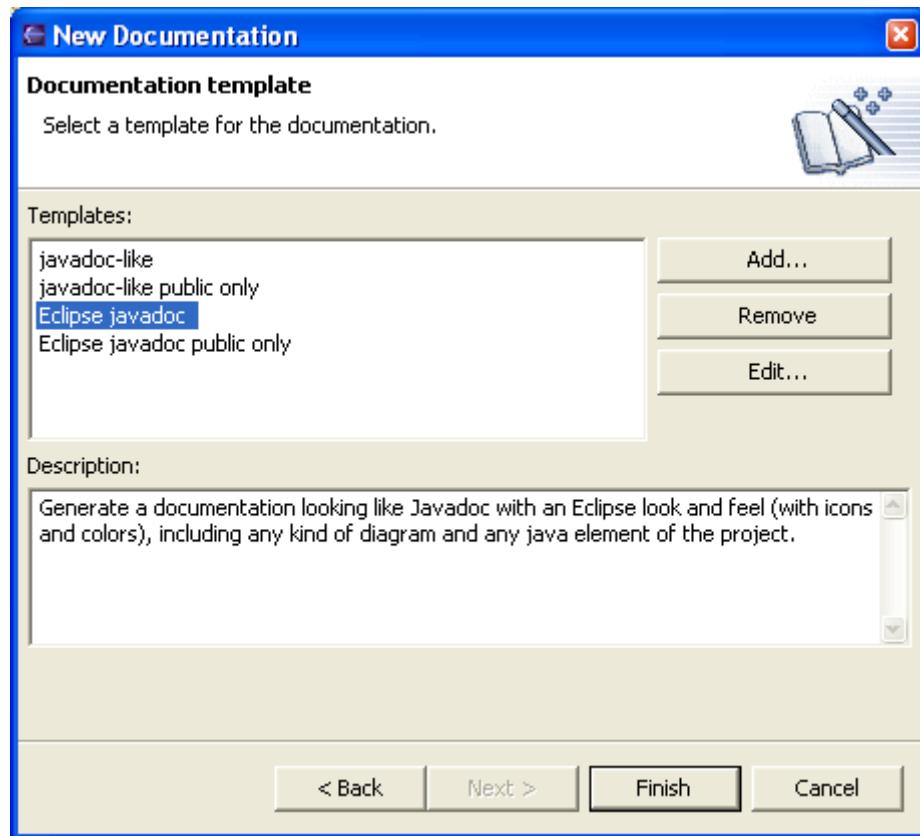
On the third step of the wizard, you can edit the title and date of the documentation. Author and authoring (legal notice, licence, ...) information can be added



Templates

On the last step of the wizard, you must select a [template](#) for the documentation generation. Each template defines which kind of element will be inserted in the generated documentation and how it will be formatted.

So if you selected a single class, you won't have any template proposed by default and will have to define one by yourself.



Generation

The documentation is generated through several steps:

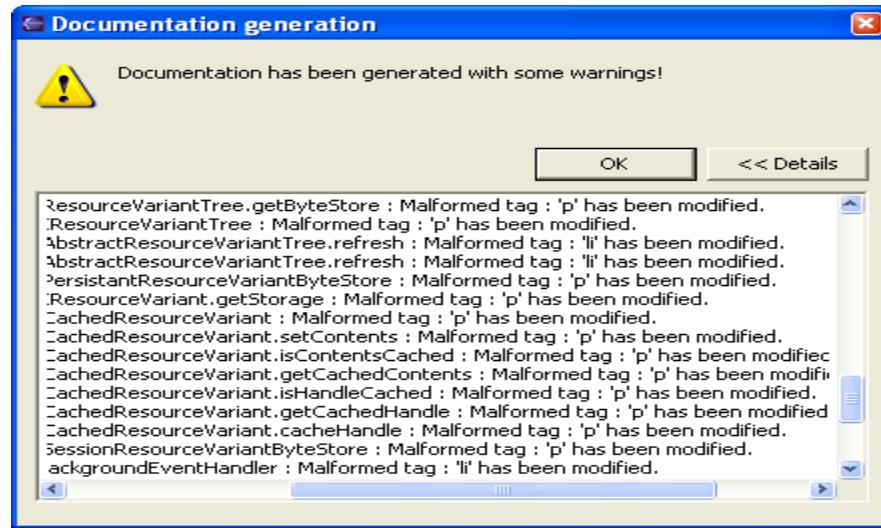
- Documentation elements analysis : the elements to include in the documentation are listed
- Elements javadoc analysis: the javadocs are analyzed, special characters are changed to numeric entities and errors are fixed.
- Diagram images generation: the UML diagram to include in the documentation are open and images are created.
- Documentation format: the final XSL transformation is processed to create the final documentation.

Javadoc errors management

If any error is found in the javadoc, it is fixed in the generated documentation. For example, malformed tag-embedments are skipped and unclosed tags are closed. < and > characters which are not tag

delimiters are changed to numeric entities.

The processed modifications are listed at the end of the generation. Original source files are not modified.



Result

Here is a sample of a generated documentation:

The screenshot shows a Microsoft Internet Explorer window displaying the generated documentation for the **org.eclipse.team.core** package. The page title is "D:\projects\runtime-workspace.RC1\org.eclipse.team.core\doc\index.html". The left sidebar lists "Packages" (org.eclipse.team.core, org.eclipse.team.core.subscribers, org.eclipse.team.core.synchronize, org.eclipse.team.core.variants, org.eclipse.team.internal.core, org.eclipse.team.internal.core.simpleAccess, org.eclipse.team.internal.core.streams) and "All Diagrams" (core-ai, subscribers-ai, synchronize-ai, variants-ai, core-ai, simpleAccess-ai, streams-ai, subscribers-ai). The main content area shows the package overview for **org.eclipse.team.core**, dated 2004.10.09. It lists the packages under **org.eclipse.team.core**: org.eclipse.team.core, org.eclipse.team.core.subscribers, org.eclipse.team.core.synchronize, org.eclipse.team.core.variants, org.eclipse.team.internal.core, org.eclipse.team.internal.core.simpleAccess, org.eclipse.team.internal.core.streams, and org.eclipse.team.internal.core.subscribers. Below this is another "Overview" section.

Templates

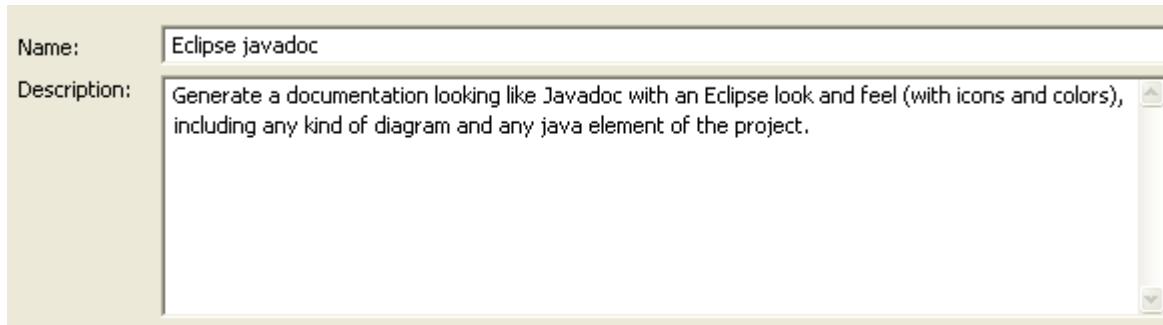
The documentation generation templates are containing three kinds of information:

- [Information on the template](#)
- [Content of the generated documentation](#)
- [Format of the generated documentation](#)

These templates can be edited either from the preferences or from the documentation generation wizard.

Information

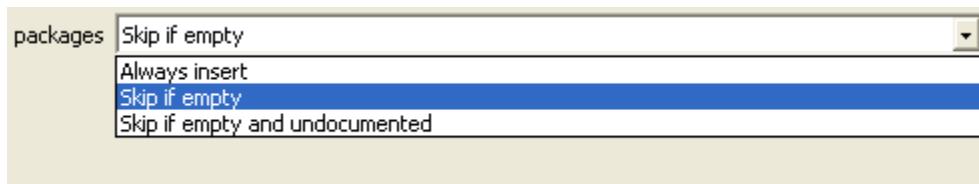
A template has a name and a description to explain what is it dedicated for.



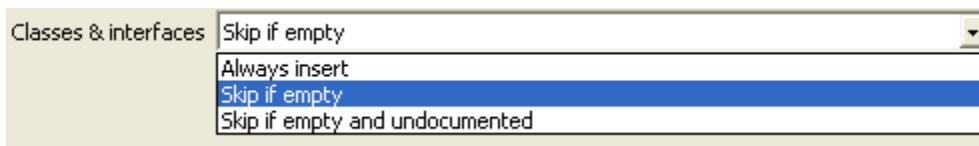
Content

When editing a template, it is possible to define what should be included in the generated documentation.

- Packages



- Class and Interfaces



- Methods

public methods	Always insert
protected methods	Always insert
package methods	Always skip Skip if undocumented Always insert
private methods	Always insert
public constructors	Always insert
protected constructors	Always insert
package constructors	Always insert
private constructors	Always insert
<input type="checkbox"/> Inherit JavaDoc if missing	

- Attributes

public attributes	Always insert
protected attributes	Always insert
package attributes	Always skip Skip if undocumented Always insert
private attributes	Always insert

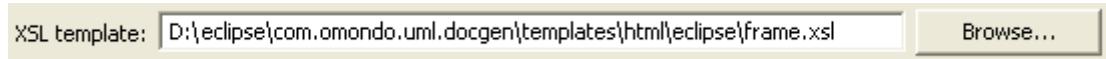
- Diagrams

Activity diagrams	Always insert
Class diagrams	Generate class diagrams on classes and interfaces
Collaboration diagrams	Always insert Always skip
Component diagrams	Generate class diagrams on classes and interfaces
Deployment diagrams	Always insert
Object diagrams	Always insert
Sequence diagrams	Always insert
State diagrams	Always insert
Use case diagrams	Always insert
Robustness diagrams	Always insert

Class diagram dedicated to a class can be automatically generated.

Format

The final format of the generated documentation depends on a XSL transformation. The XSLT template to use must be indicated.



For now, the DTD used for the XSL transformation is not published. It will be enhanced and published on a further coming version.

Two default XSLT templates are provided :

- Javadoc-like documentation
- Eclipse Javadoc-like documentation (with Eclipse icons and decoration added to a standard-style javadoc)

Jee Code Generation

- [Create an AndroMDA Project](#)
- [EJB Code Generation](#)
- [Hibernate Code Generation](#)
- [JSF/STRUTS Code Generation](#)
- EJB3 modeling (5min) : [Flash demo](#) / [.exe file](#)
- EJB 3 deployment with JBoss (3 min): [Flash demo](#) / [.exe file](#)

Create AndroMDA Project

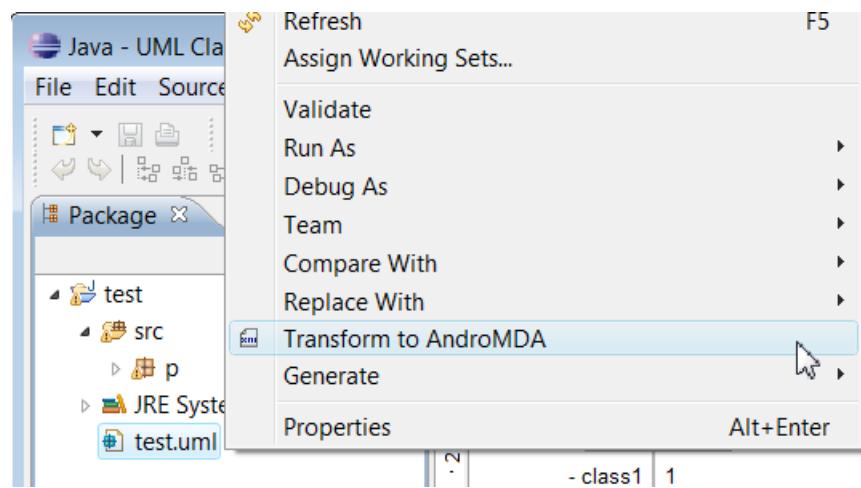
It is possible to either use WTP/AndroMDA and Eclipse with EclipseUML 3.3 or 3.4 or directly AndroMDA 3.3 code generator with the EclipseUML Java and Java EE build.

1. [EclipseUML for Java and Java EE modelers \(June 18th, 2009 and after\)](#)
2. [EclipseUML for Java EE modelers \(previous build\)](#)

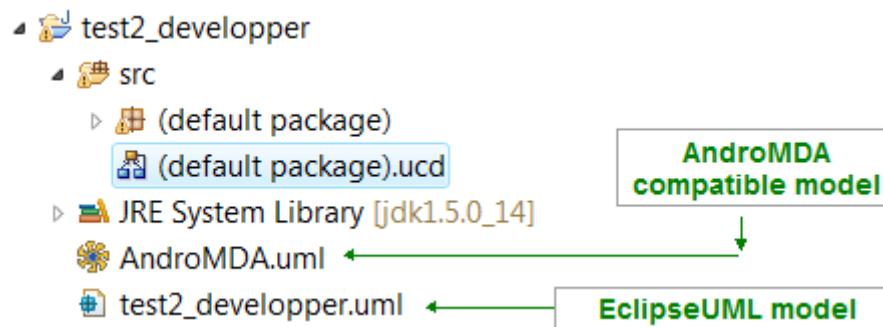
1.EclipseUML for Java and Java EE modelers (after June 18th, 2009)

You can use EclipseUML and generate a compatible AndroMDA 3.3 model.

Click on the EclipseUML model in the Package Explorer **Project name.uml file > Transform to AndroMDA**



The AndroMDA compatible file has been generated in the Package Explorer.



The Jee code generator is not using the traditional Omondo EMF code generator. The live code and model synchronization is therefore not possible.

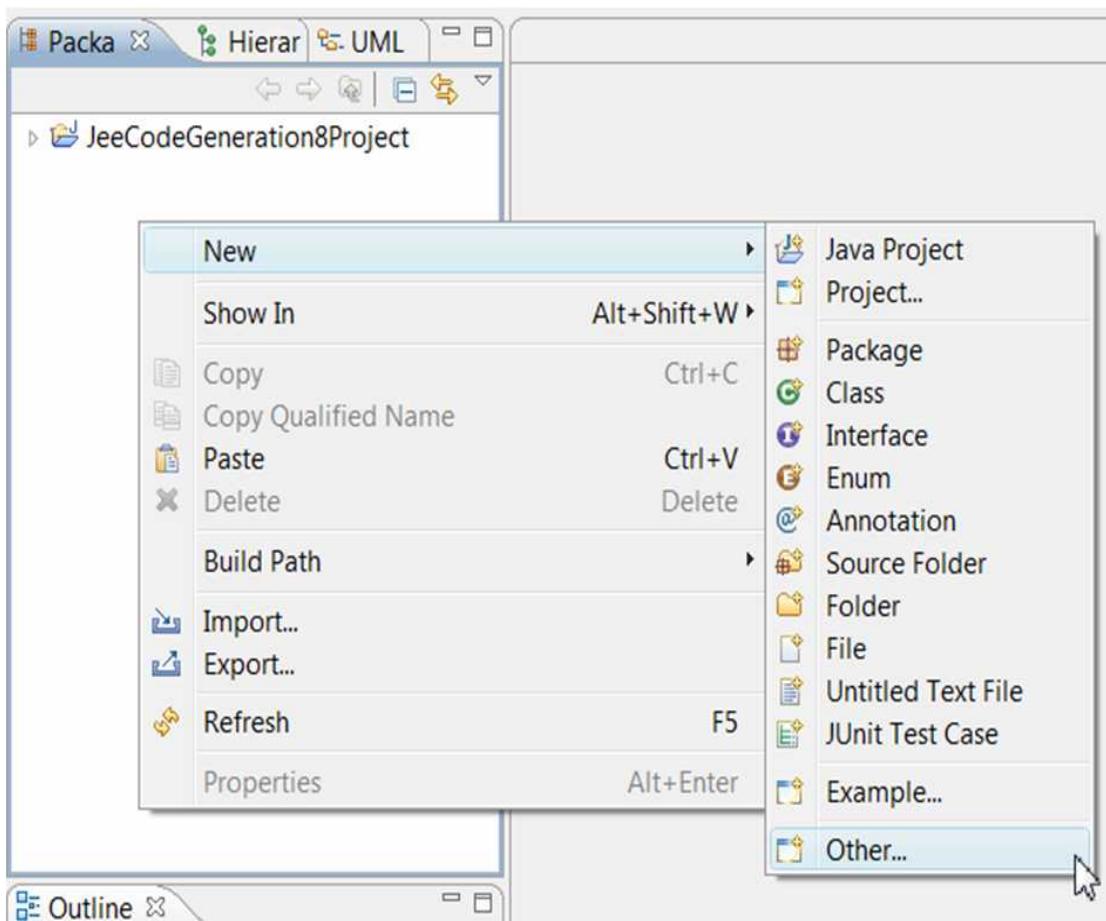
It is only possible to generate your class diagram Jee code by using the AndroMDA code generator.

In order to use the AndroMDA code generator, you need to:

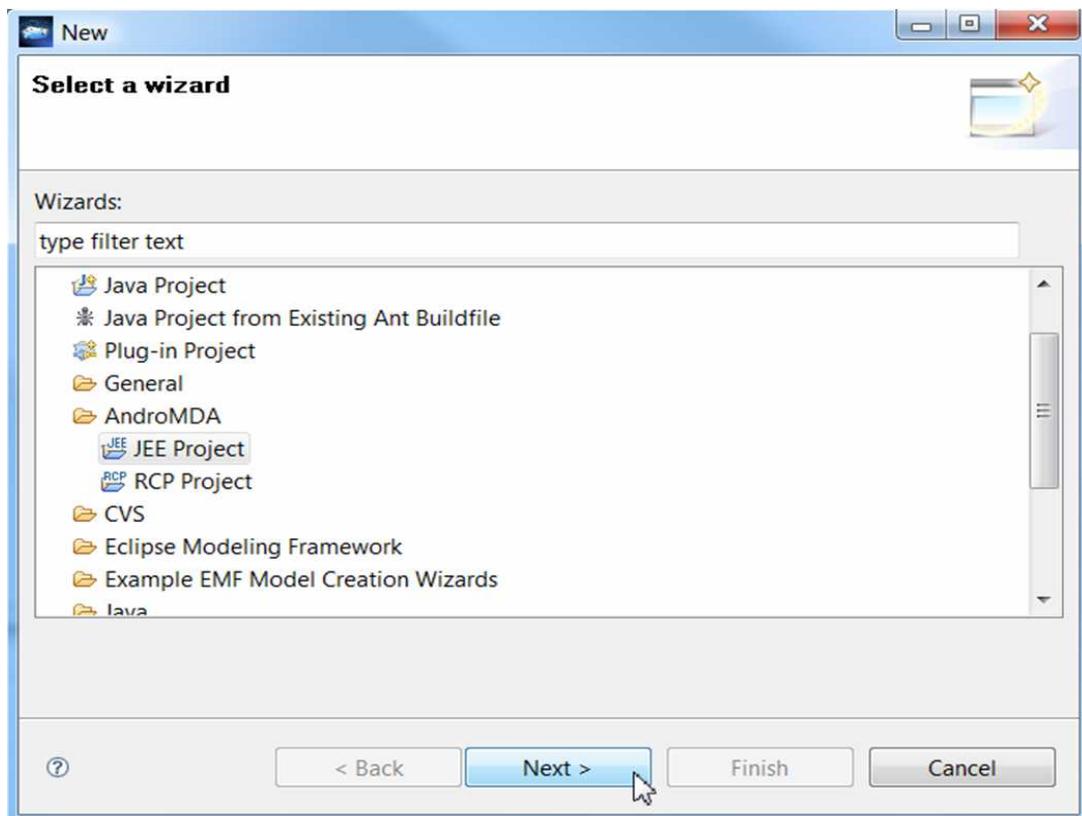
1. [Create a new and empty AndroMDA project](#)
2. [Select the project UML model and launch the Jee templates engine to fullfile the previously created AndroMDA project](#)

1. Create a new AndroMDA Project

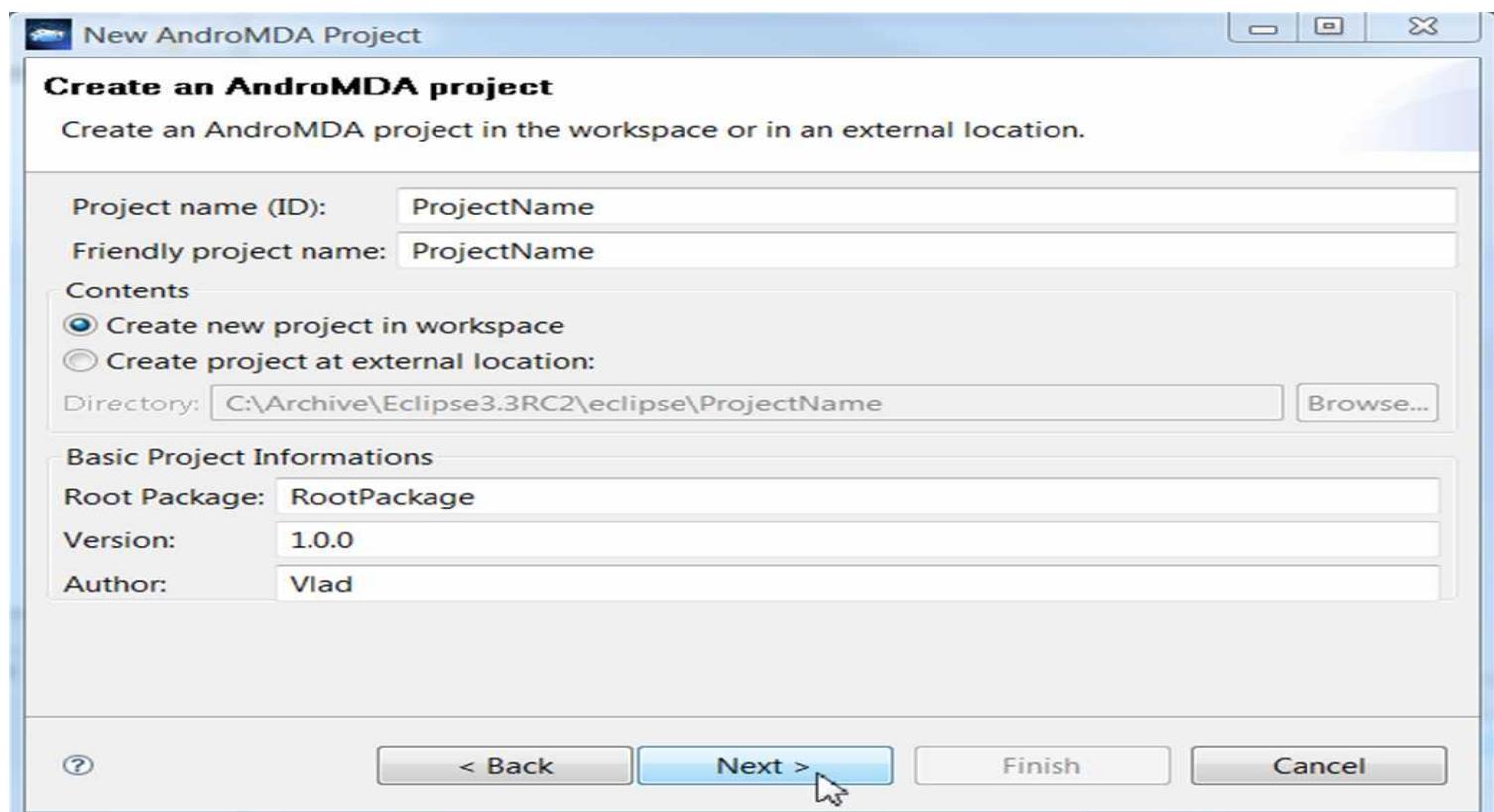
Click in the package explorer to open the contextual menu, then select **NEW > Other...**



Select **AndroMDA > JEE Project** and click on the next button

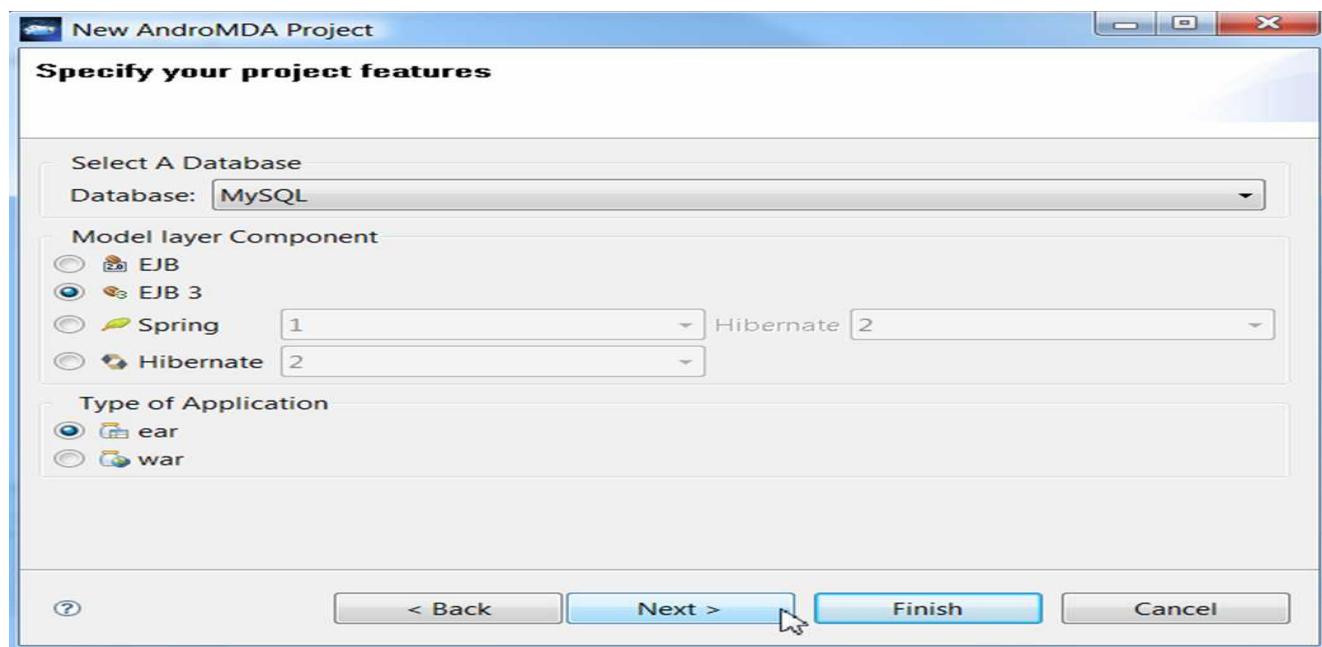


Enter the project name, the Root Package and click on the next button

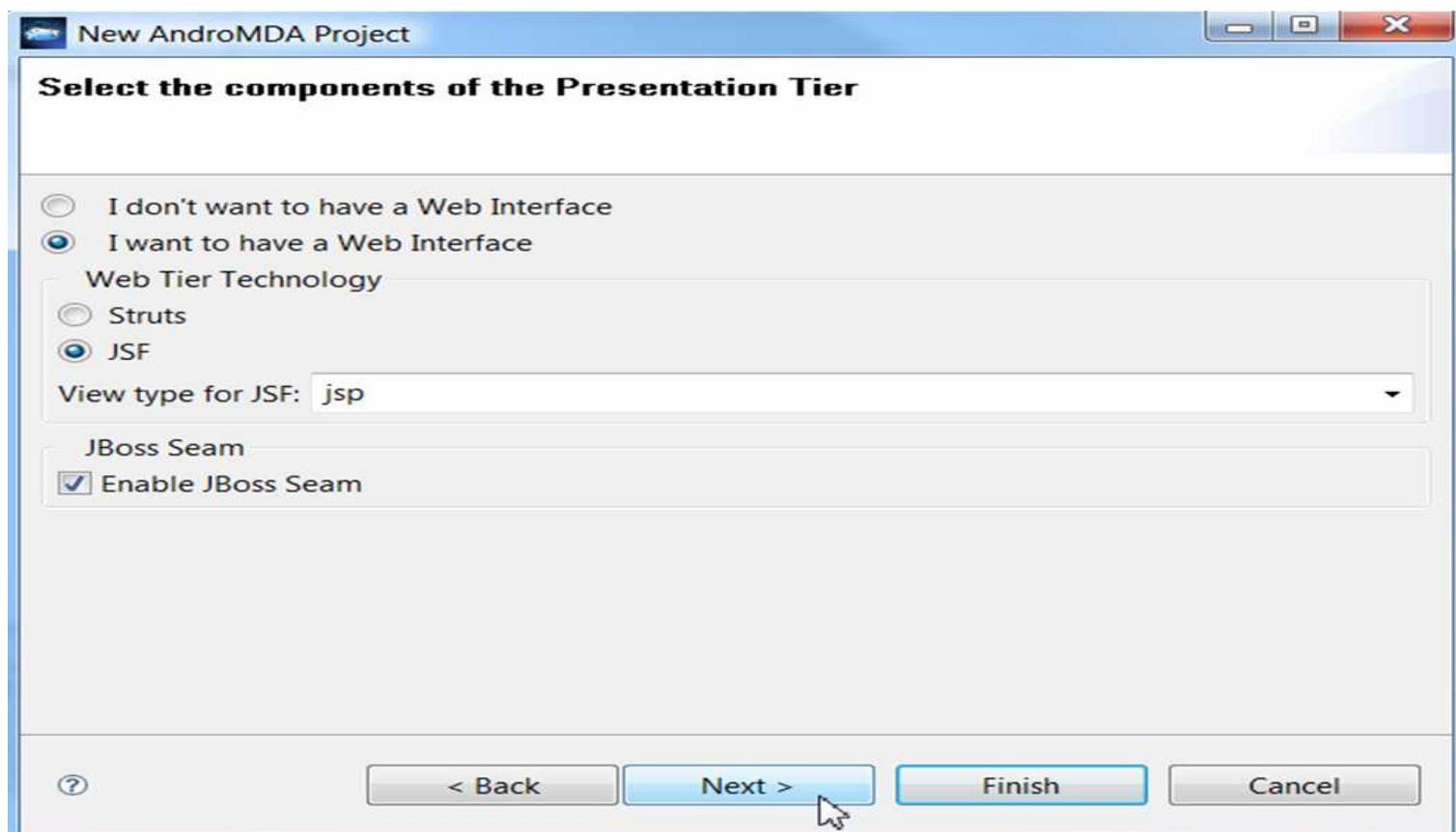


Select in the new AndroMDA Project wizard:

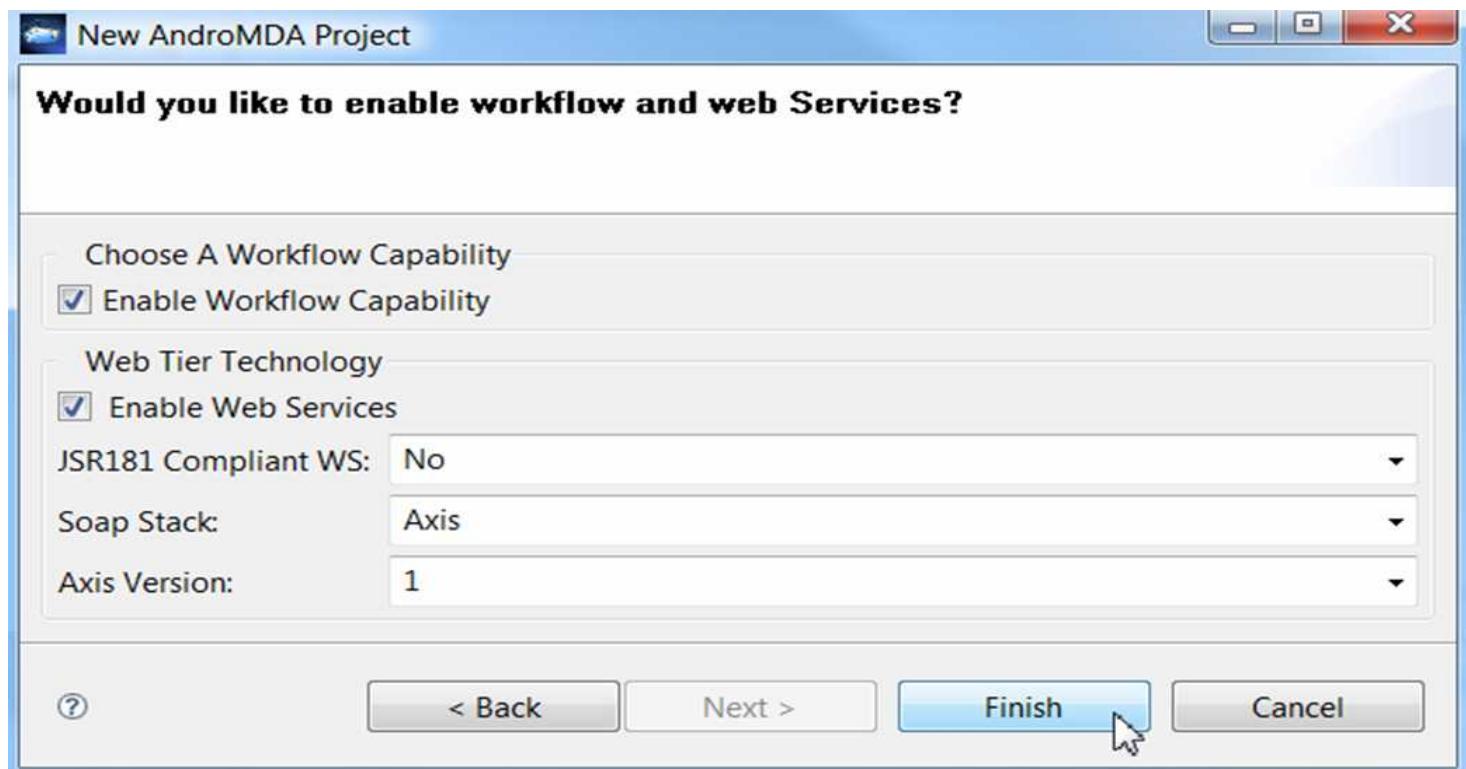
- The database
- The model Layer
- The type of Application



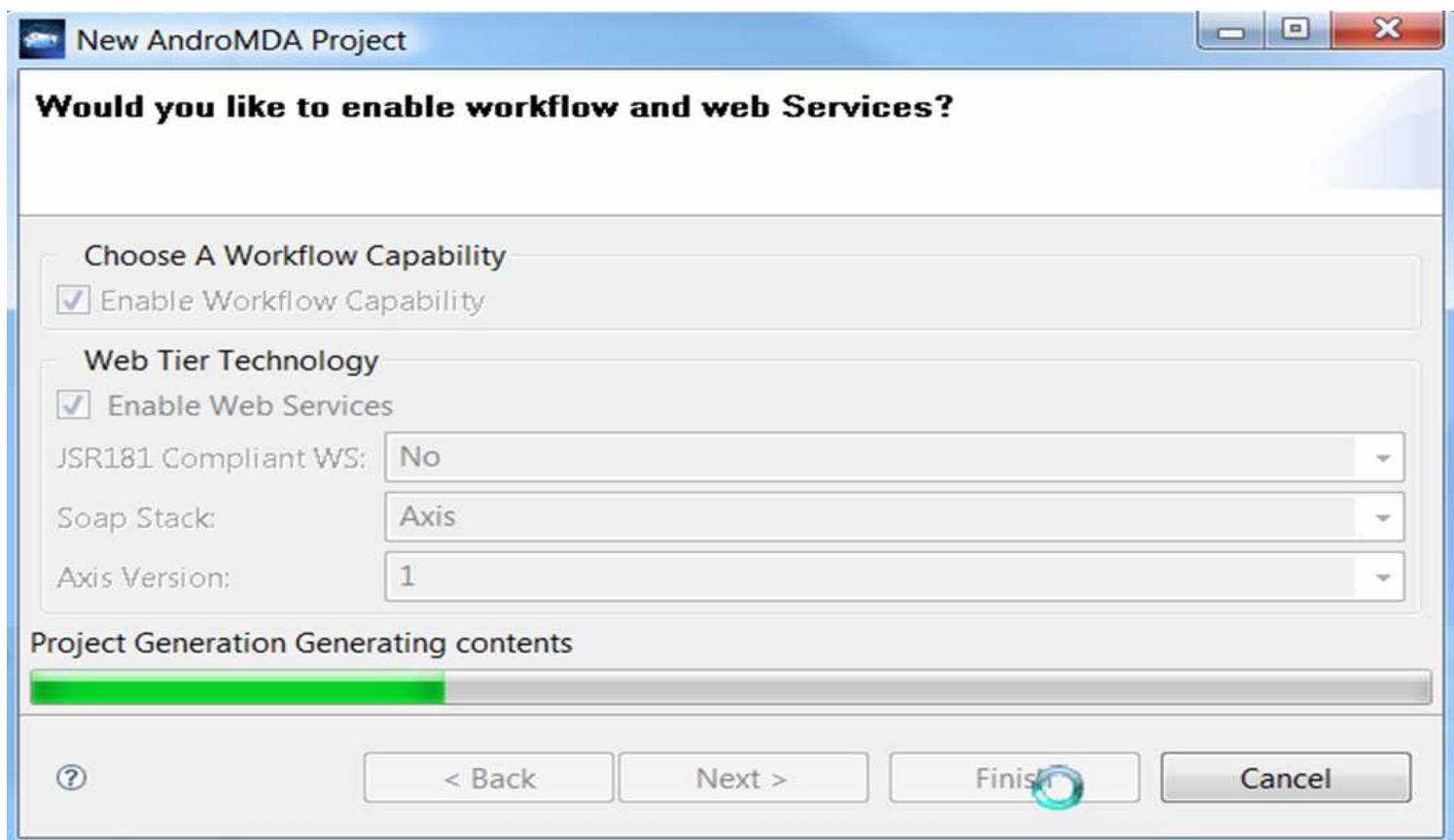
Select the components of the Presentation Tier



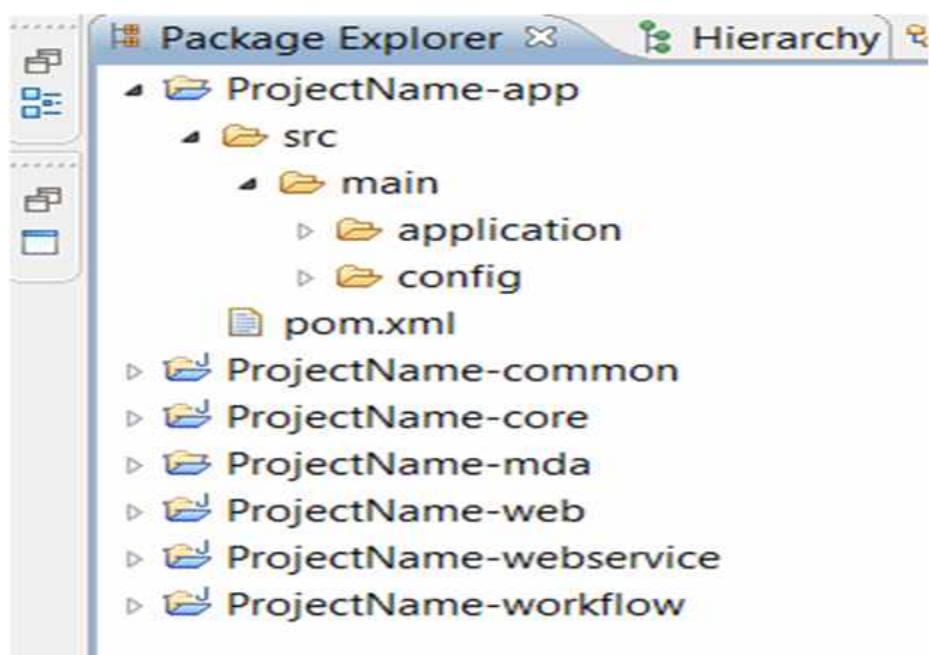
Select the workflow and web Services options, and then click on the finish button



The templates engine is building the new AndroMDA empty project

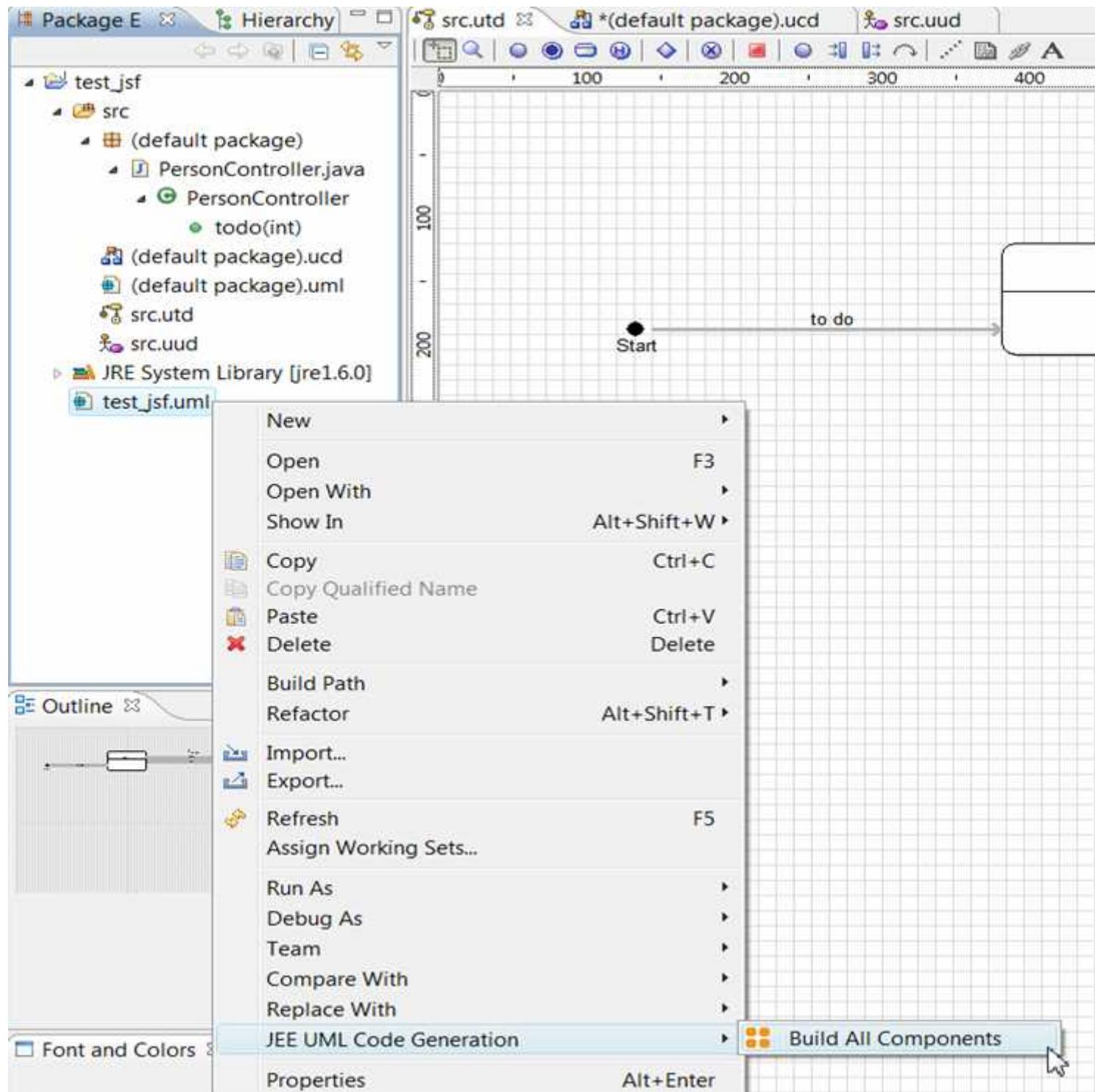


A new AndroMDA project has been created in the package explorer.
It is composed by 7 projects having dependencies between them.



2. Launch the templates engines

Select your UML Project, and click to open the contextual menu **JEE UML Code Generation > Build All Components.**



EJB Code Generation

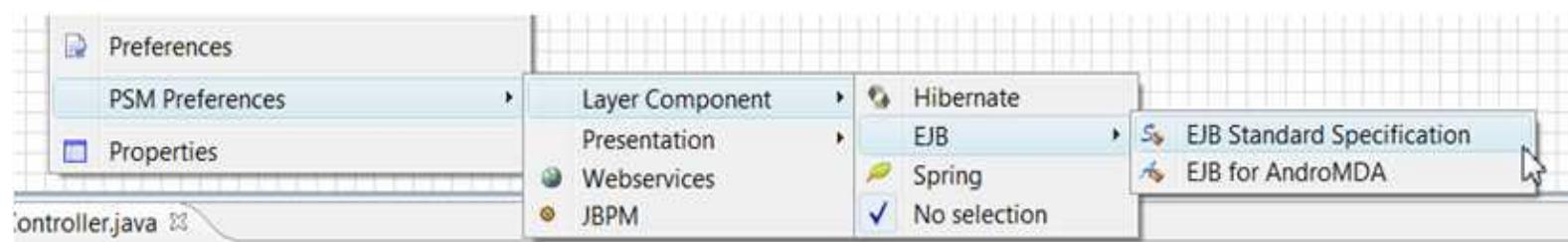
To activate the EJB code generation you need to:

1. Create an empty AndroMDA project and select EJB as the Model Layer Component
2. Create a class diagram using EJB stereotype
3. Generate Jee code from the project.uml file inside the previously created AndroMDA project

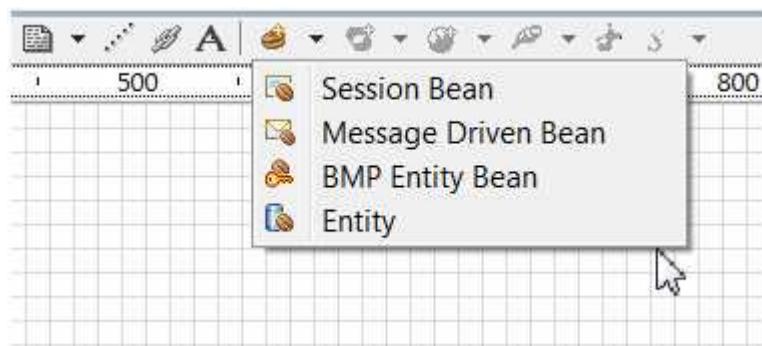
The AndroMDA new project creation has been explained in the JEE code generation part.

In order to create a class diagram using EJB stereotype,

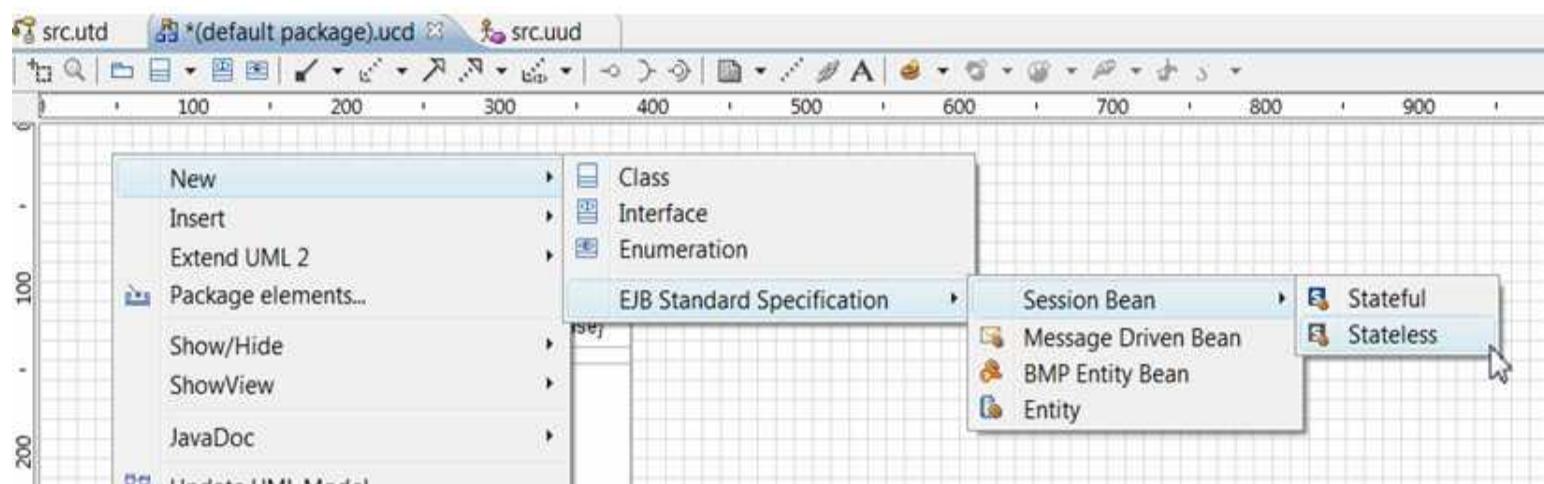
you need to open the diagram contextual menu and select **PSM Preferences > Layer Component > EJB**



The EJB modeling menu will then be available in the toolbar



And also in the class diagram contextual menu



After all the EJB business logic has been modeled, select your project.uml file in the package explorer
JEE UML Code Generation > Build All Components (with EclipseUML 3.3 and EclipseUML 3.4)

Click on the EclipseUML model in the Package Explorer *Project name.uml* file > Transform to AndroMDA (with the latest EclipseUML builds)

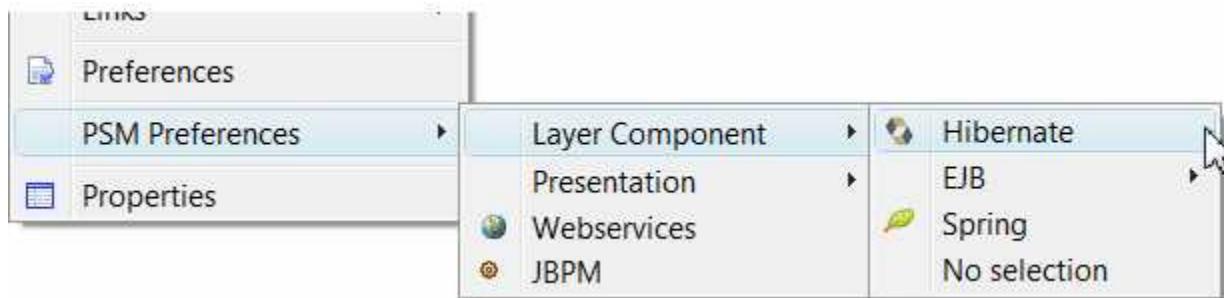
Hibernate Code Generation.

To activate the Hibernate code generation you need to:

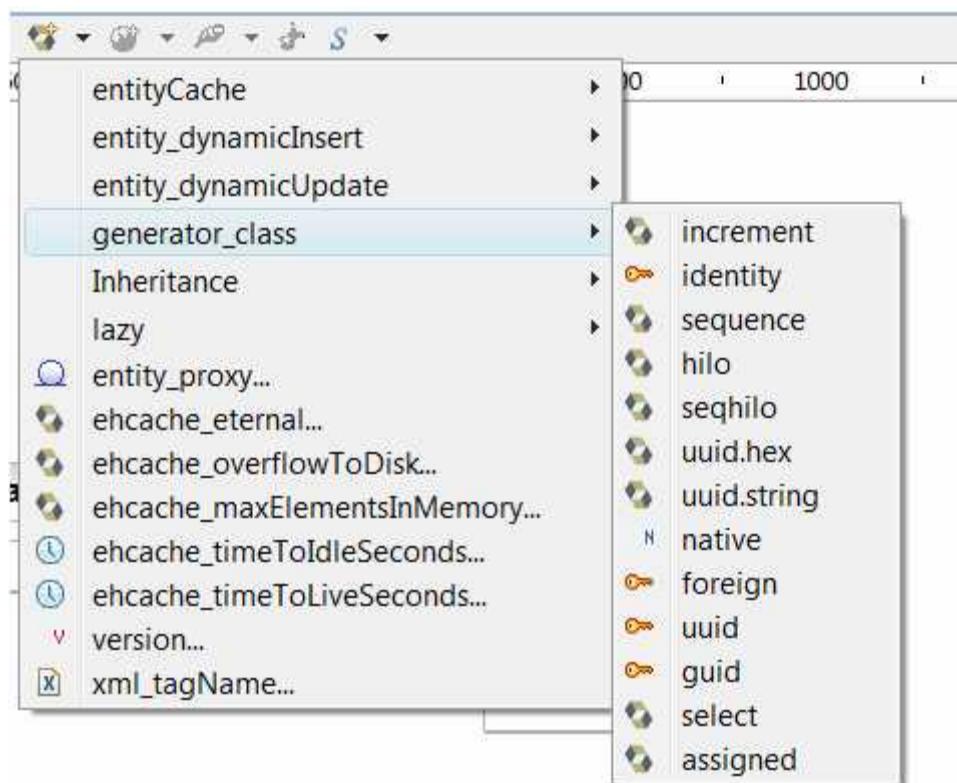
1. Create an empty AndroMDA project and select Hibernate as the Model Layer Component
2. Create a class diagram using Hibernate stereotype
3. Generate Jee code from the project.uml file inside the previously created AndroMDA project

The AndroMDA new project creation has been explained in the JEE code generation part.

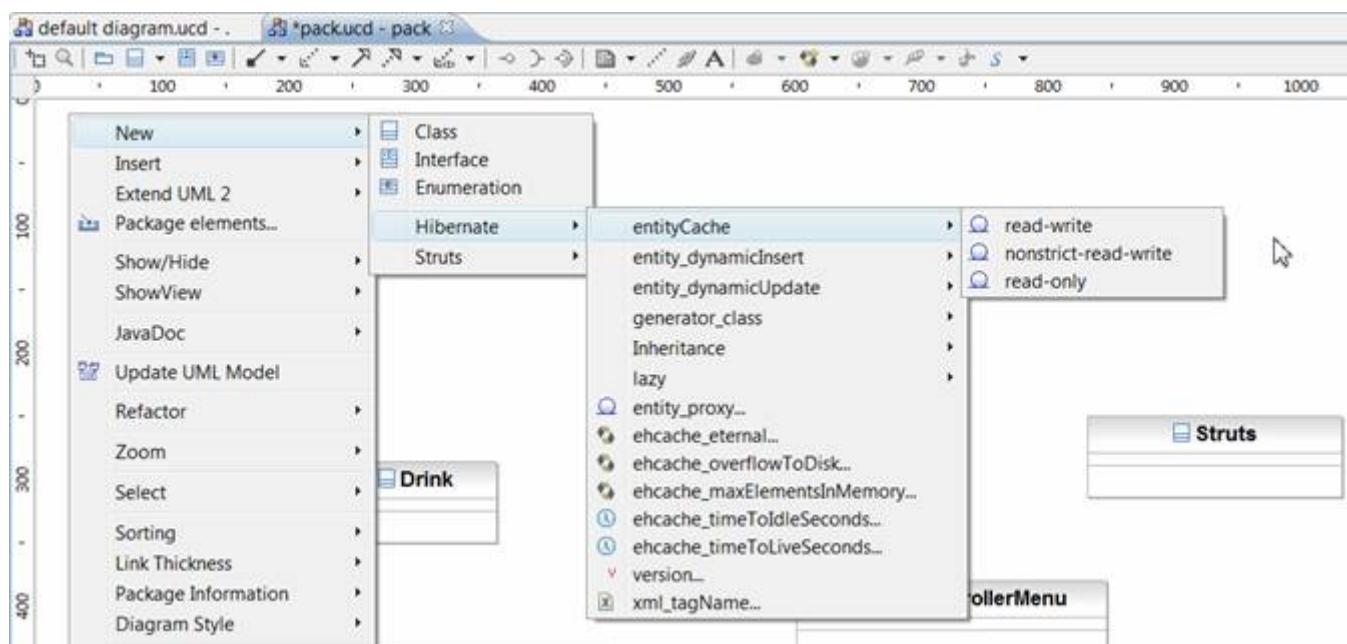
In order to create a class diagram using EJB stereotype, you need to open the diagram contextual menu and select **PSM Preferences > Layer Component > Hibernate**



The Hibernate modeling menu will then be available in the toolbar



And also in the class diagram contextual menu



After all the Hibernate business logic has been modeled, select your project.uml file in the package explorer **JEE UML Code Generation > Build All Components** (with EclipseUML 3.3 and EclipseUML 3.4)

Click on the EclipseUML model in the Package Explorer **Project name.uml file > Transform to AndroMDA** (with the latest EclipseUML builds)

JSF/STRUTS Code Generation

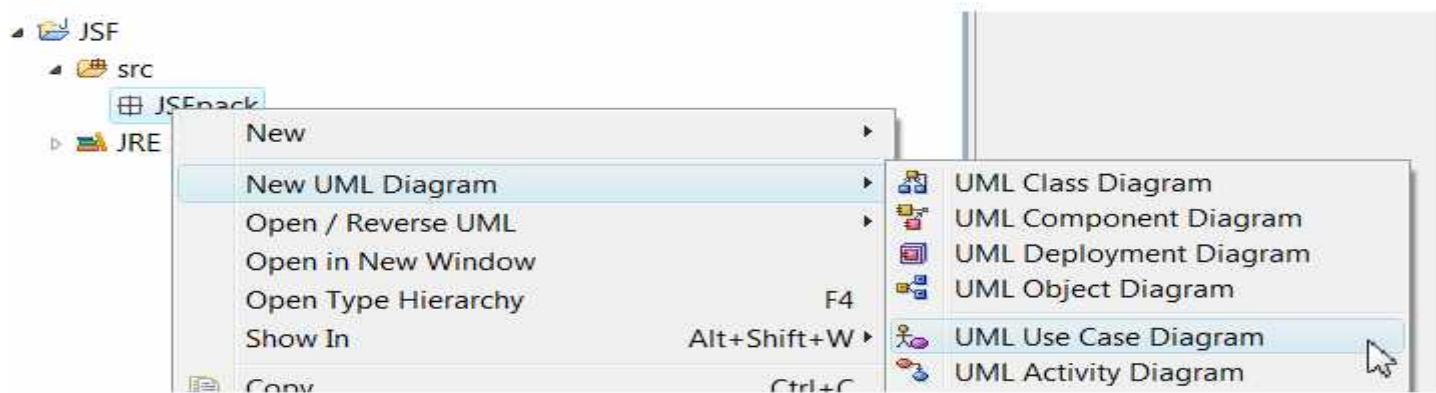
The JSF code generation needs three diagrams to unify business logic and web tier presentation.

You need one or more:

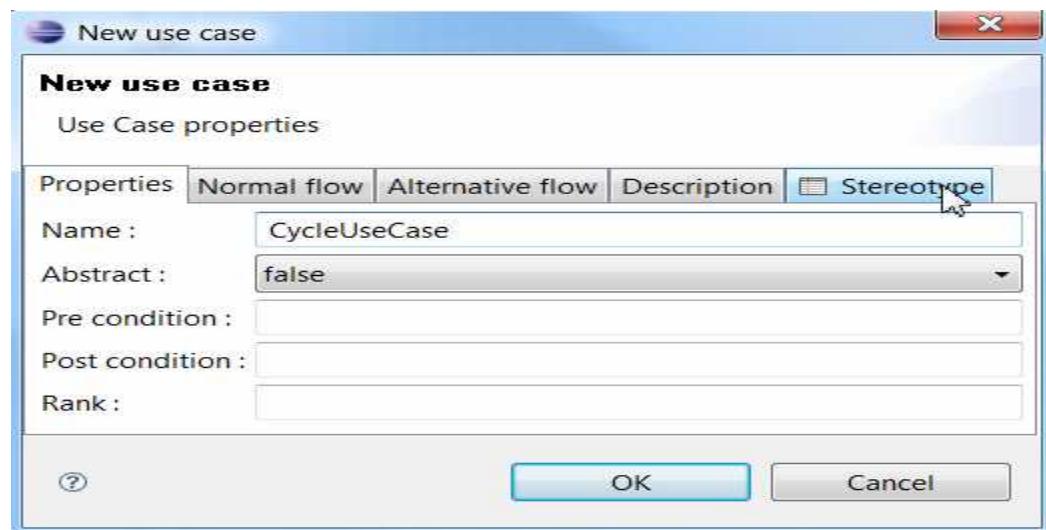
1. [UseCase Diagram](#)
2. [Class Diagram](#)
3. [StateChart Diagram](#)

The following example is a quick start to explain how to generate code

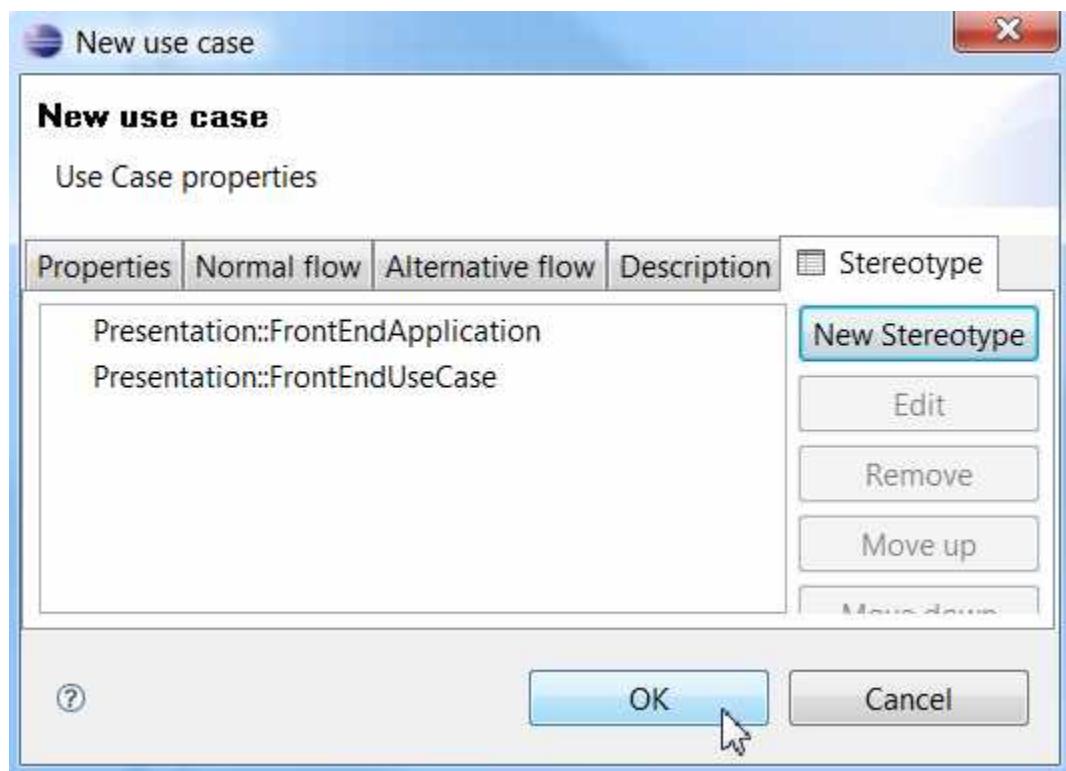
Create a UseCase Diagram



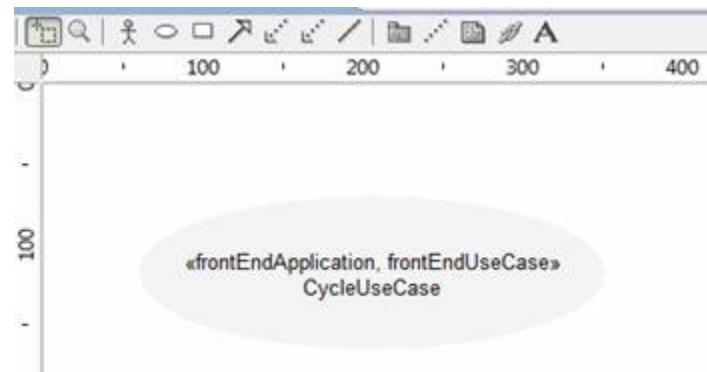
1. First Create a UseCase inside your UseCase Diagram:



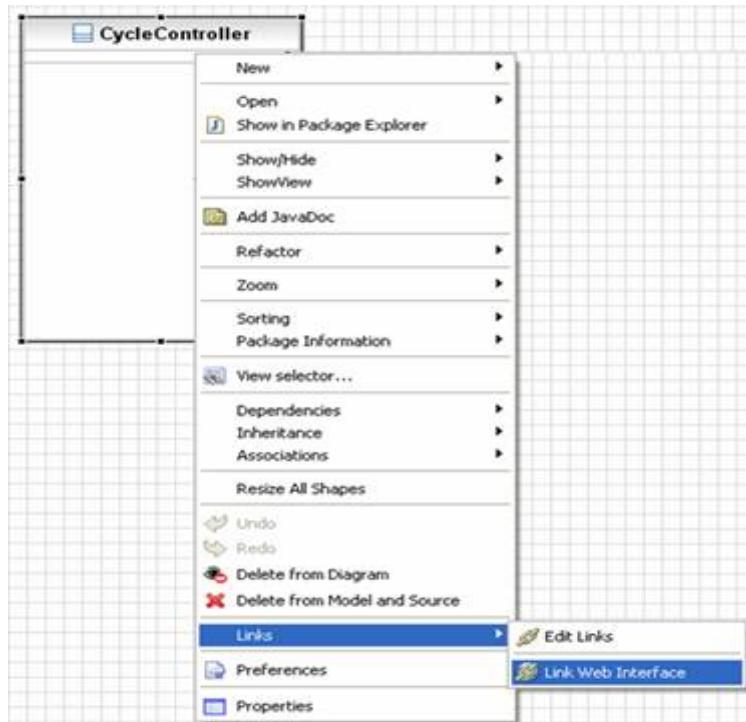
Select Stereotypes as below:



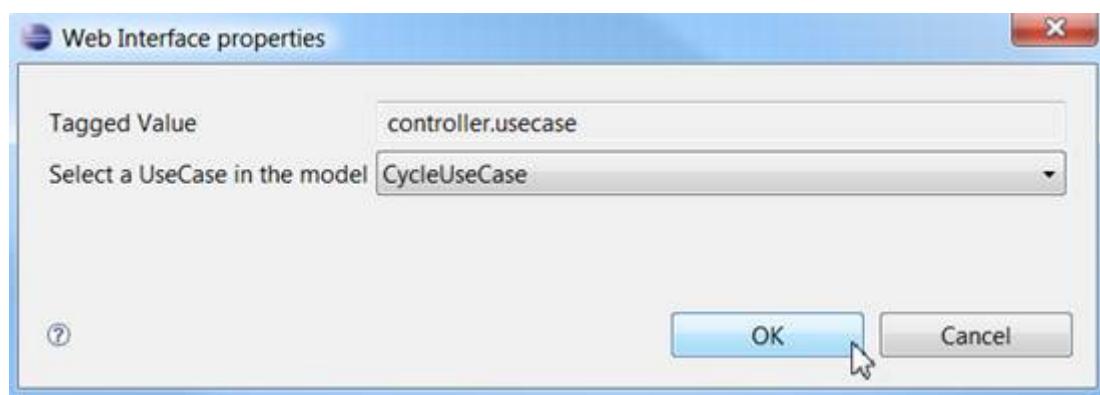
You get the following usecase



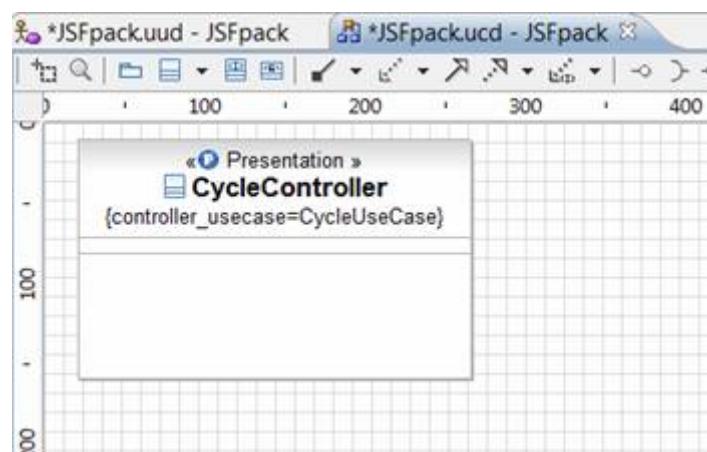
2. Then Create a Class diagram, a class named CycleController



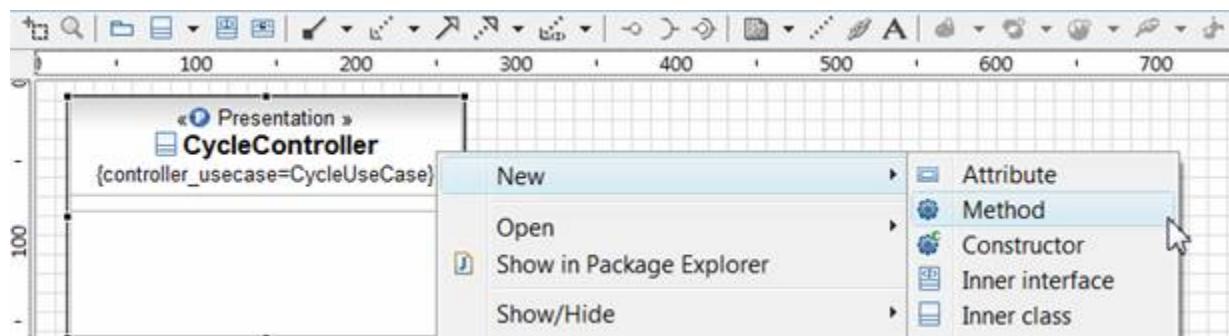
Select the UseCase in charge of the web Interface and click the OK button



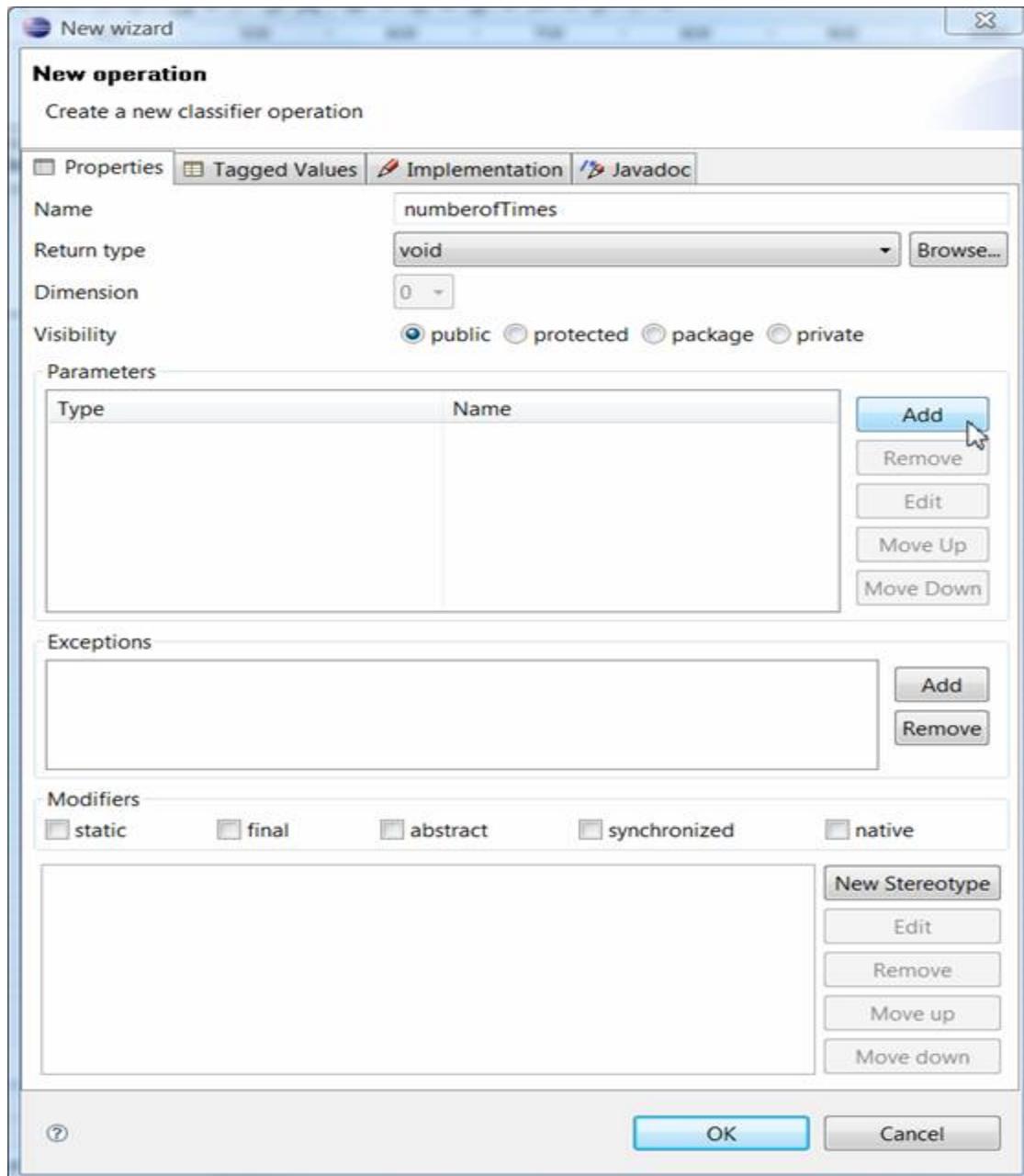
You automatically get



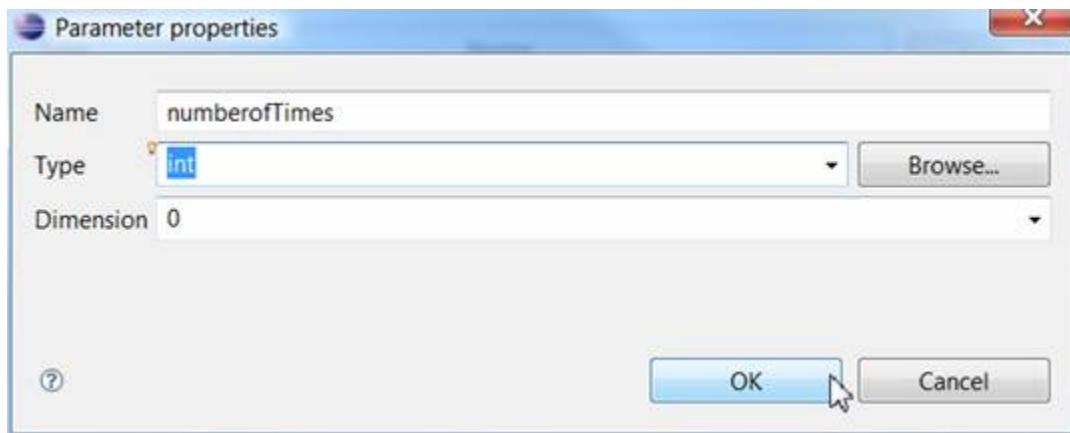
You now need to add a method and a parameter to your class diagram



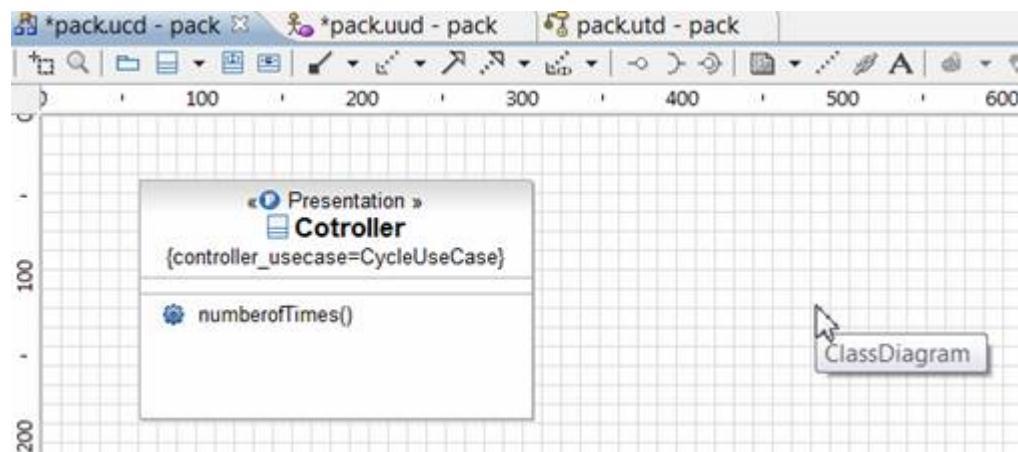
Enter the method name “numberOfTimes” and click on the Add button.



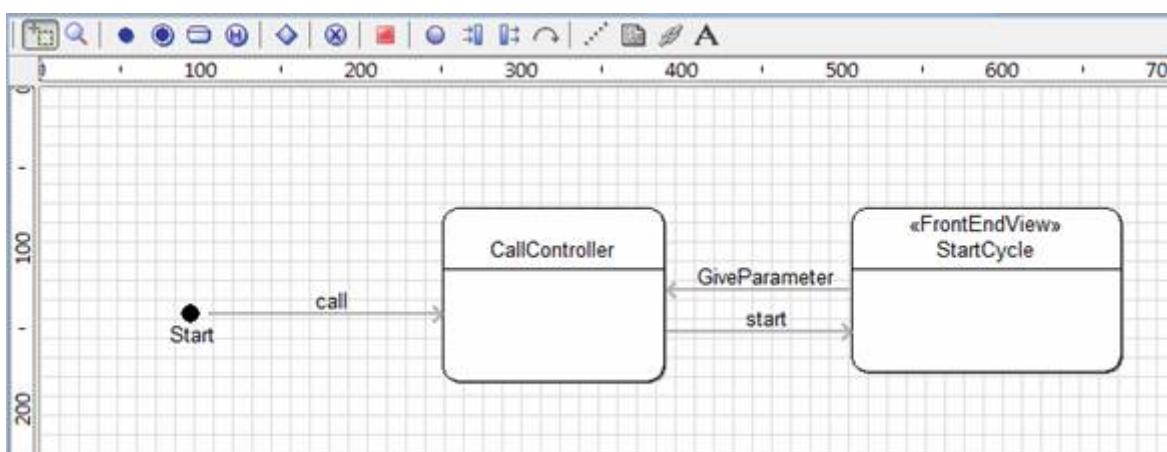
Enter the parameter name and type



You now get a method inside your class diagram

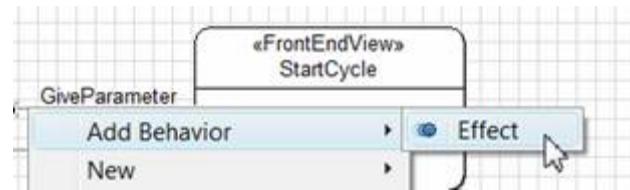


3. Finlay, you now need to create a statechart diagram representing the JSF work flow

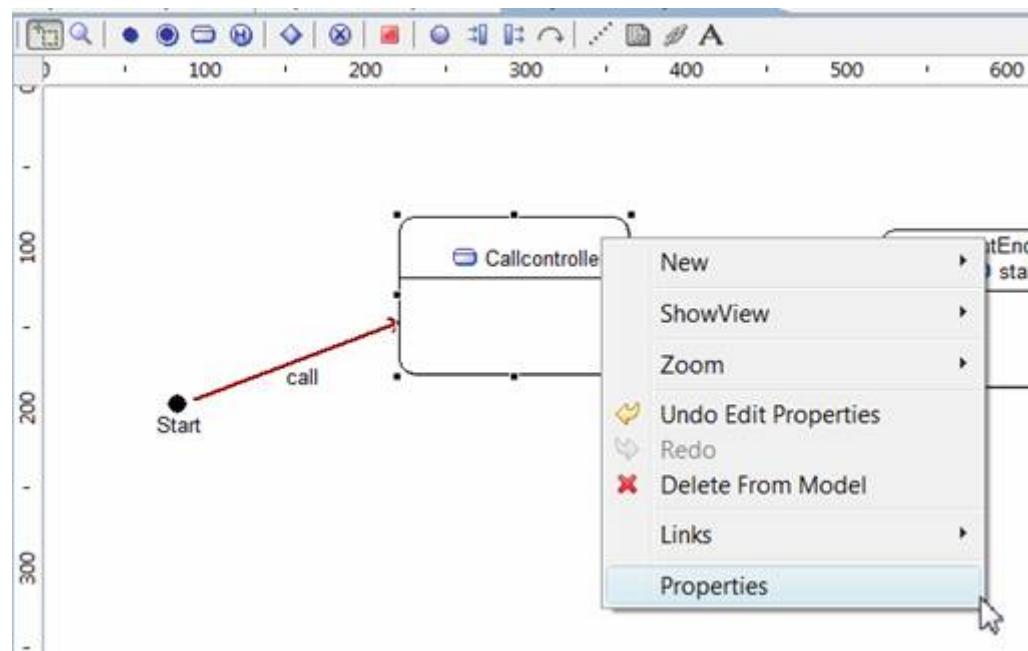


The GiveParameter need to have an Effect.

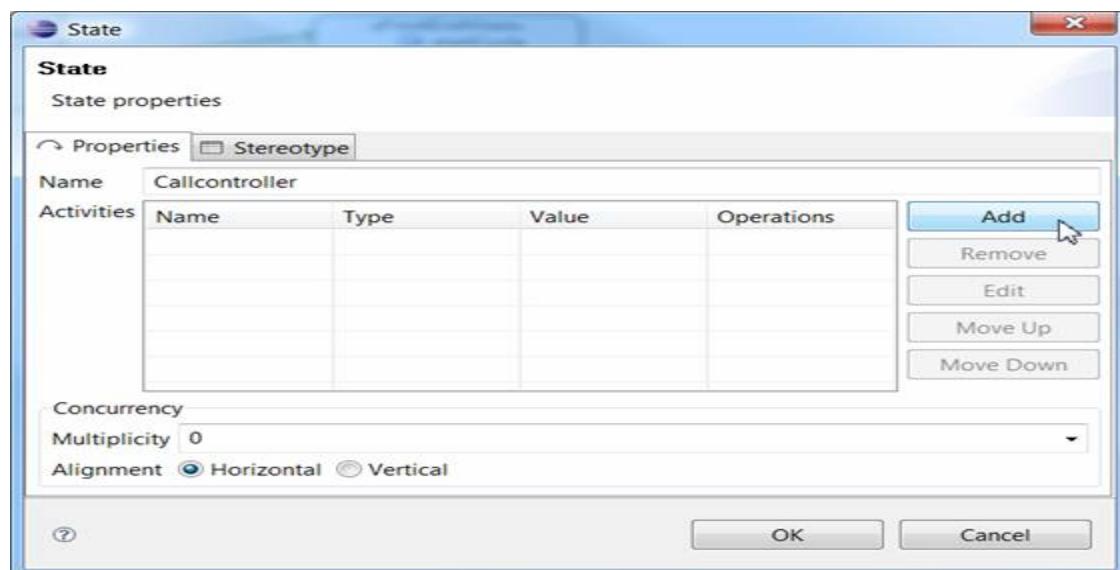
Click on the transition and open the contextual menu > Add Behavior > Effect



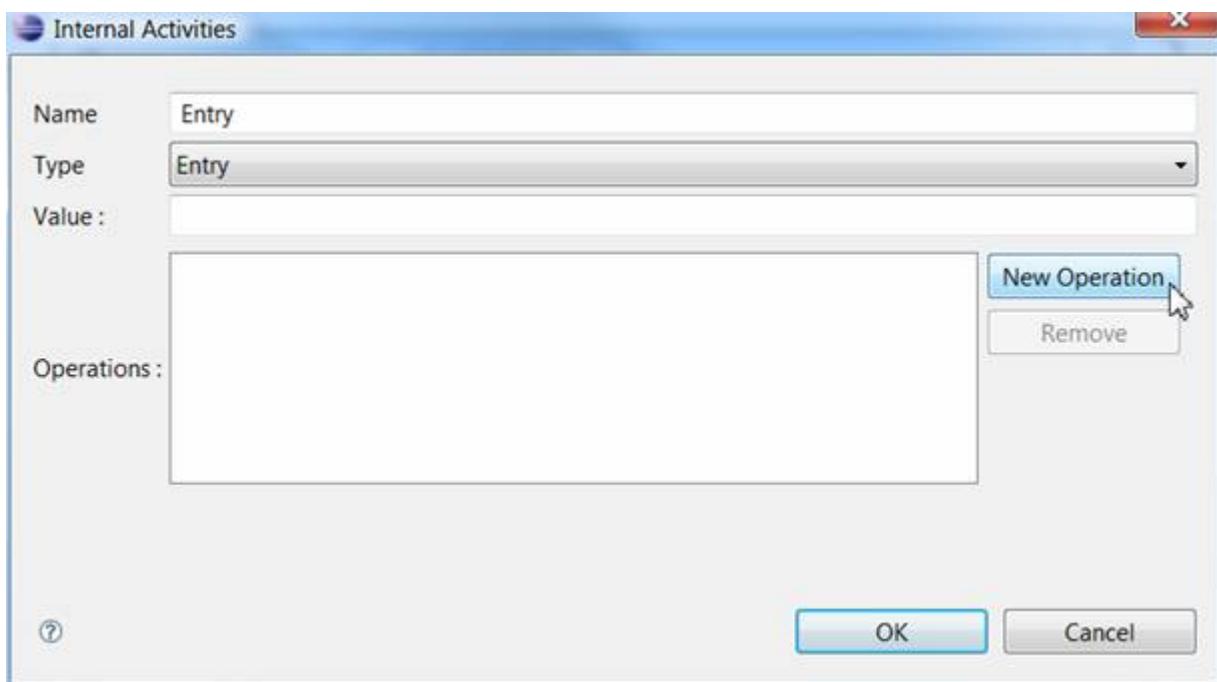
Click on the CallController State



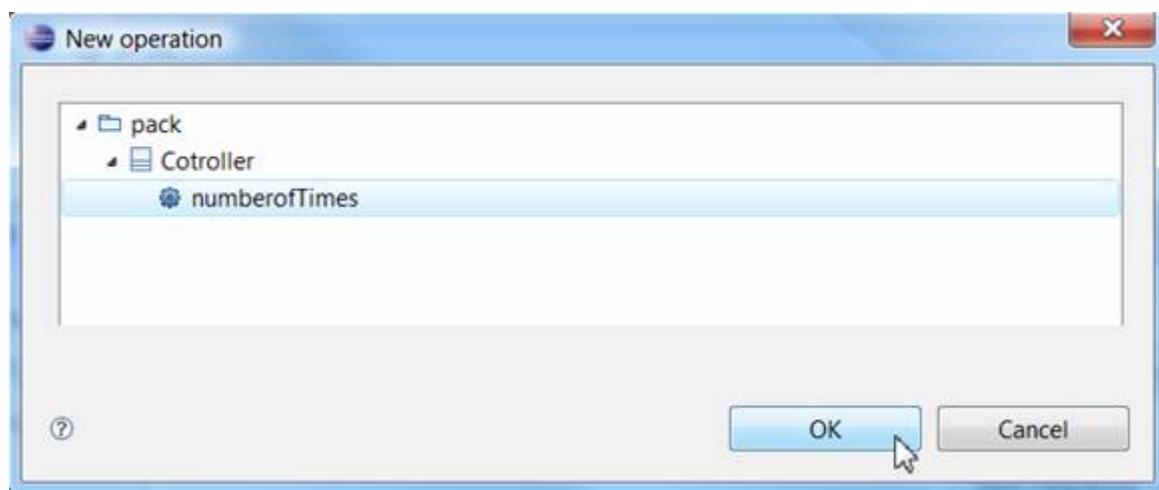
Click on the Add button



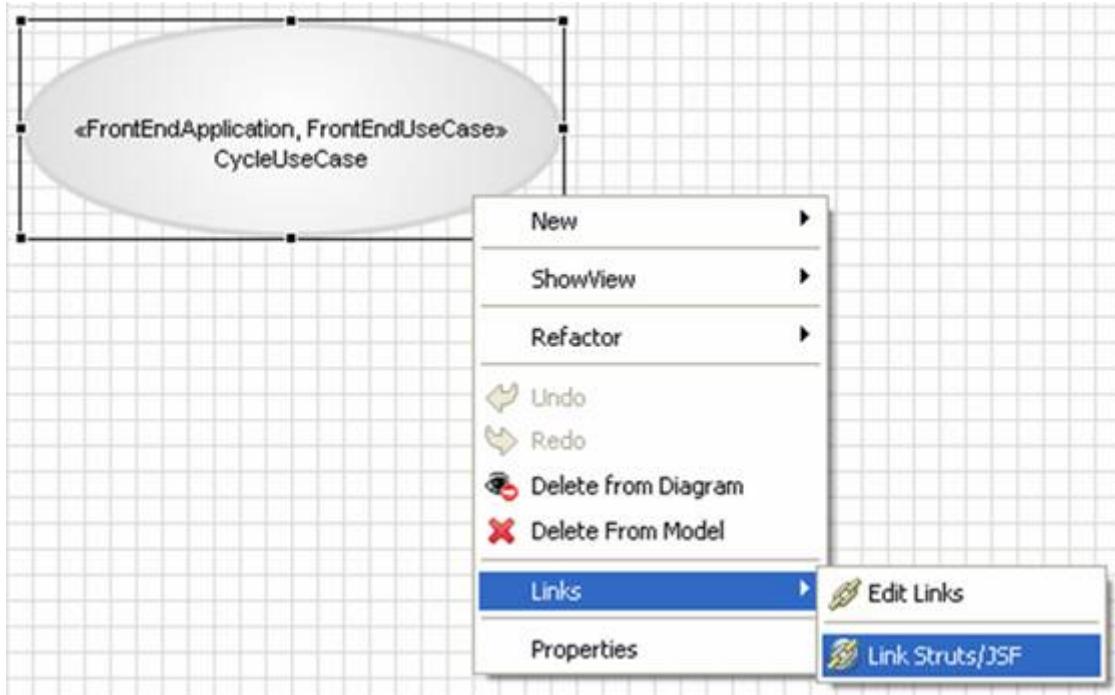
Type a name and select Entry in the internal Activities wizard, then select New operation in order to connect the class diagram method and the state.



You will open the New Operation selector, which will display all classes having parameters and a stereotype “controller_usecase”.



You finally need to connect your usecase to its Statechart Diagram



Select the Statechart diagram related to the usecase



JDK 5 – JDK 6 Support

JDK 1.5 - JDK 6 Support

EclipseUML fully support all JDK 5 & 6 elements:

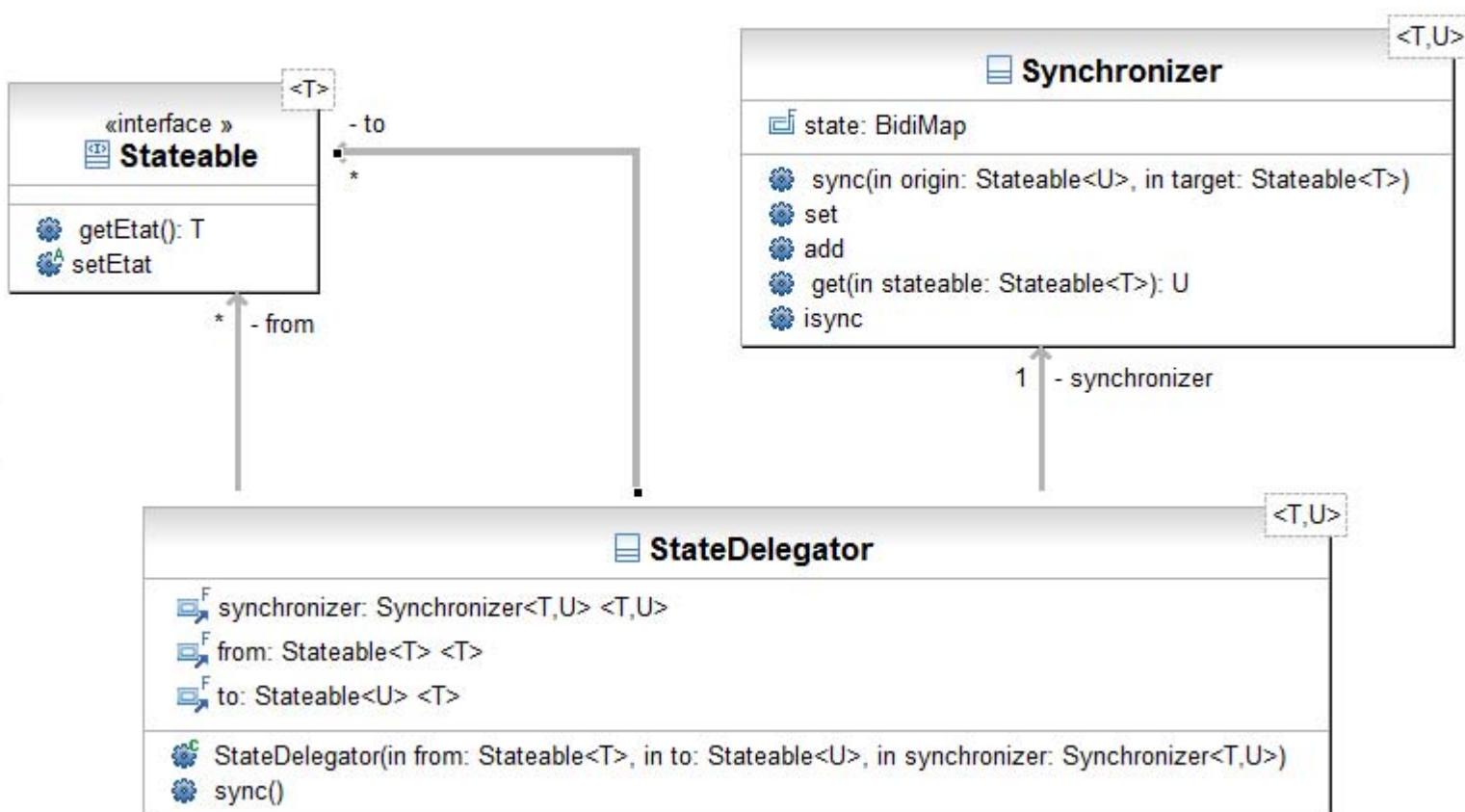
- Enumeration edition
- Generic type
- Java Annotation

This chapter deals with:

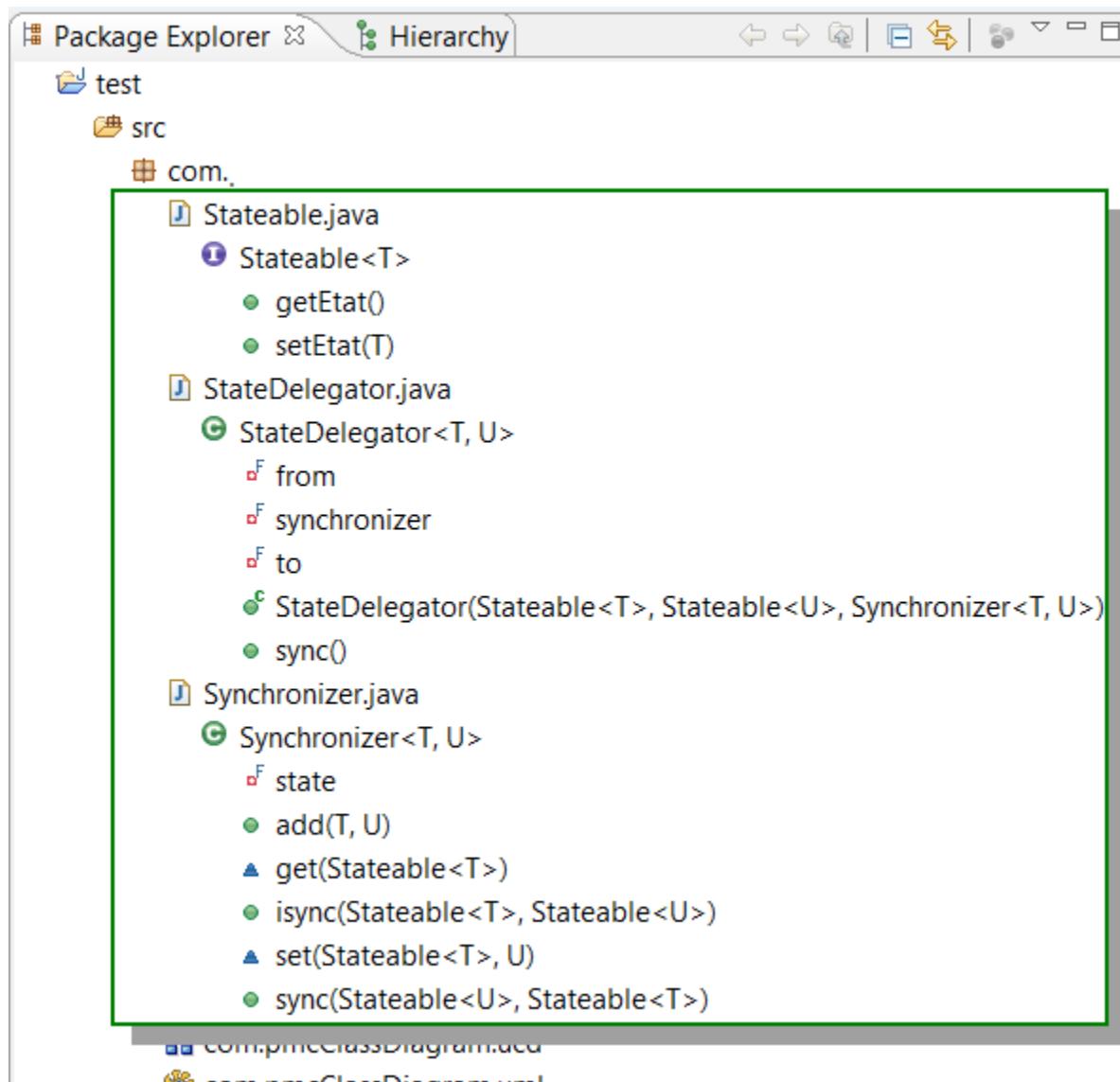
1. [Enumeration](#)
2. [Generic type](#)

Here is an example of generic type support:

UML Editor:



Package Explorer:



[Home](#) > [JDK Support](#)

Enumeration

1. [Enumeration Model](#)
2. [Diagram presentation](#)
3. [Tutorials](#)
 1. [Simple mode](#)
 2. [Advanced mode](#)
 3. [Interface provider connection](#)

1. Enumeration Model

An enumeration in Java is a specific kind of **class**, which can have some private constructors, attributes, methods and static instances as literals. The literals are the static members.

Here is a simple example:

```
public enum Category
{
    History, Drama, ScienceFiction;
```

It should be possible that a literal has some additional attributes. For example to add the nature language name:

```
public enum Category
{
    Documentary("Documentary"), Drama("Drama"), ScienceFiction("Science
Fiction");
    protected String name;
    private Category(String name)
    {
        this.name = name;
    }
    public String getName()
    {
        return name;
    }
}
```

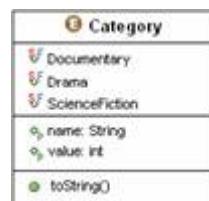
All properties must be read-only. An enumeration cannot have a super class, but it is possible to implement some interfaces.

There is a mismatch in terms of model between Java and UML2. In Java, it is possible to have an inner enumeration:

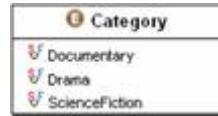
```
public enum Outer
{
    Literal1;
    public enum Inner
    {
        Literal1;
    }
}
```

2. Diagram presentation

For the graphical presentation, all literals are grouped in a new compartment on top of the attribute compartment:



We introduce a new enumeration mode, which contains only the literals. This will be used as the default mode:



This mode can be switched to “Advanced mode” via the context menu “Switch to advanced mode”. It can be brought back to “Simple mode” via the context menu “Switch to simple mode”.

Each literal can be hidden individually like attribute and method. It can be selected to show up via the “View selector”, or the “undo” operation of “hide”. In general, we show or hide all literals together.

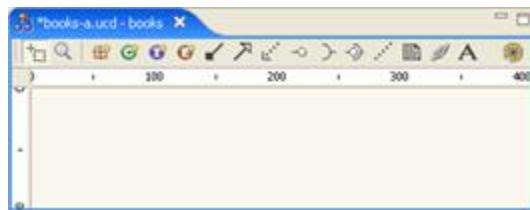
3. Enumeration Tutorials

These tutorials illustrate how to create and edit an enumeration from a simple case to a more complex one.

3.1. Simple mode

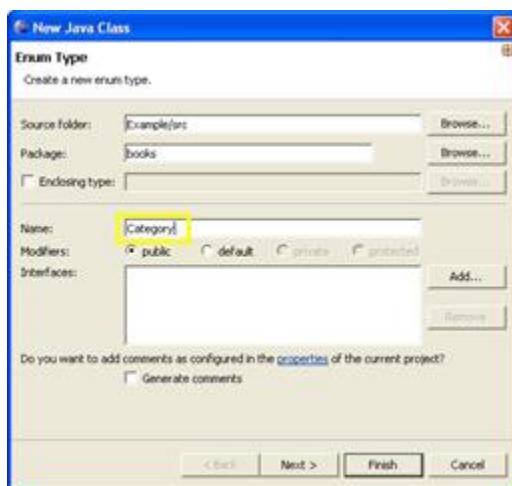
1. Open a diagram in a UML project with JDK 1.5 features

First we need to create a diagram using the New Wizard or package context menu Open UML->Class diagram editor.

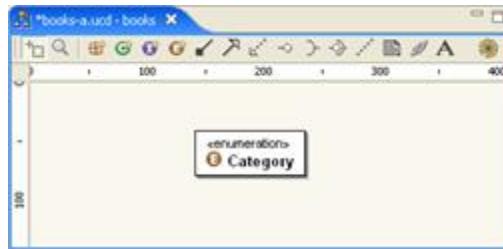


2. Create an enumeration via the tool bar

Select the enumeration icon tool in the tool bar of the diagram editor.

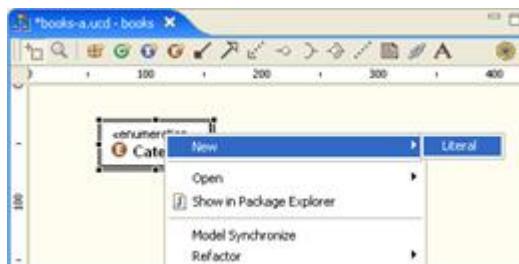


In the Enumeration Wizard, type the name “Category”. You will have an enumeration shown as below.

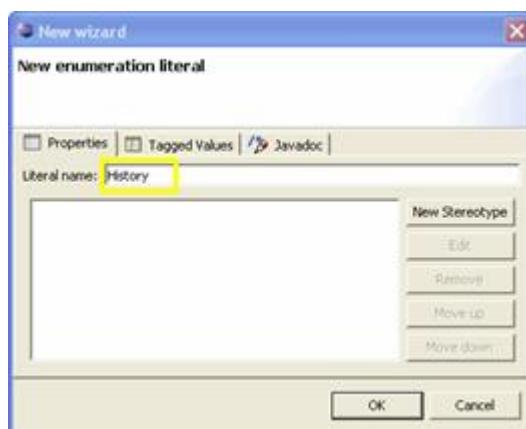


3. Add a literal

Select the enumeration above; open the context menu and select New->Literal.



Type the new literal name “History”, and then click the button OK to validate this operation.



A new literal is created.



Here is the Java code generated for this enumeration:

```
Package books;

public enum Category
```

```
{  
    History;  
}
```

3.2. Advanced mode

1. Open a diagram in a UML project with JDK 1.5 features

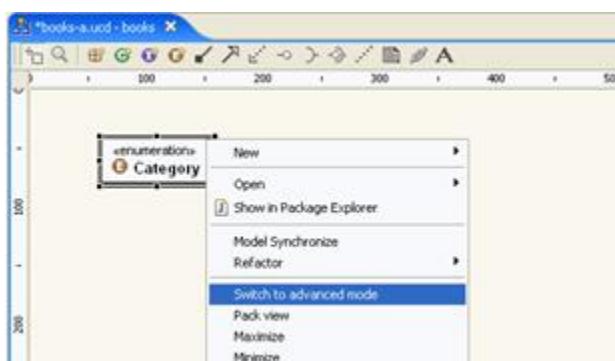
Same as the simple mode.

2. Create an enumeration via the tool bar

Same as the simple mode.

3. Switch to advanced mode

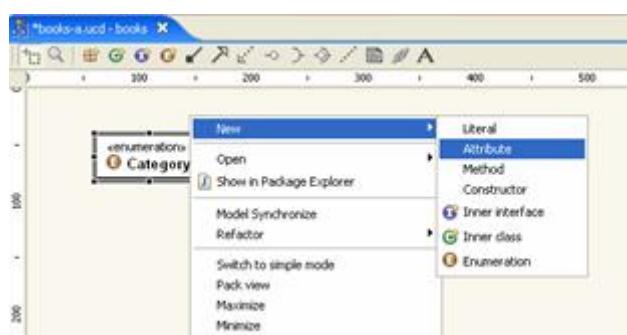
Right click on the enumeration and select the context menu “Switch to advanced mode”.



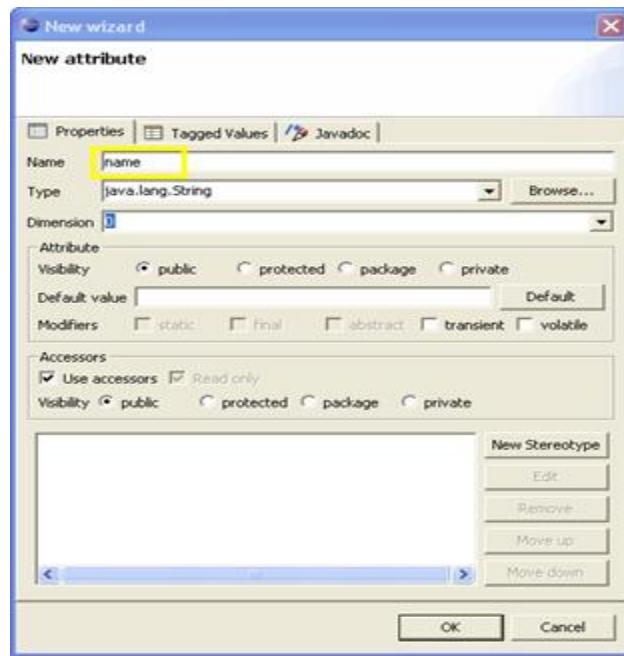
Afterwards, this enumeration is considered as a class and the menus to create/insert properties and methods are shown.

4. Create a property

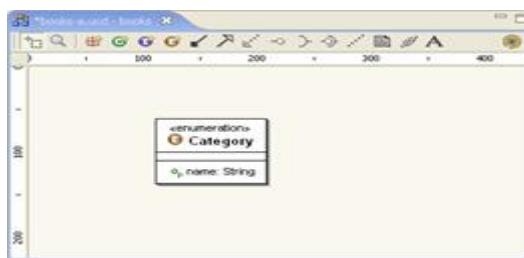
Open the enumeration context menu and select New -> Attribute.



Put the “name” in the field name and click the button OK with the default options.

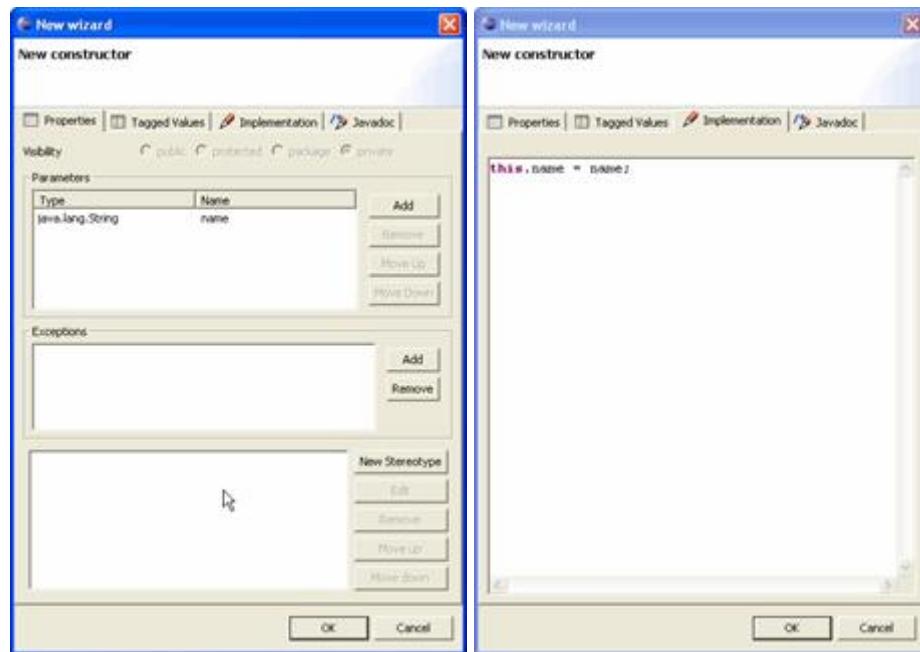


A new attribute and read-only accessors are created.



5. Create the default constructor

Open the enumeration context menu and select New -> Constructor.

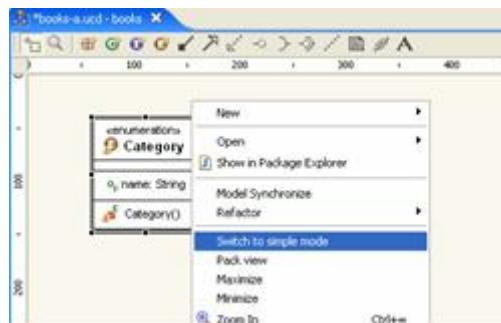


In the constructor wizard, all parameters and body implementation are pre-filled automatically. Just click on the OK button.



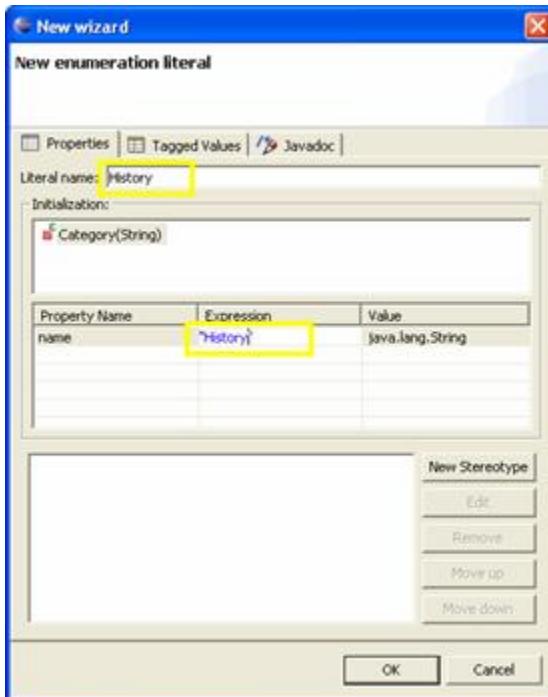
6. Switch back to simple mode

Select enumeration and open the context menu, select Switch to simple mode.



7. Add a literal

Same as operation 3 in the simple mode open the context menu and select New->Literal.



In this new look wizard; enter the literal name “History” and a constant expression in the second column for the literal initialization. If there is more than one constructor, it is possible to select the appropriate one in the list. The corresponding parameters will be updated in the table below.

Here is the Java code generated for this enumeration:

```
Package books;

public enum Category
{
    History("History");

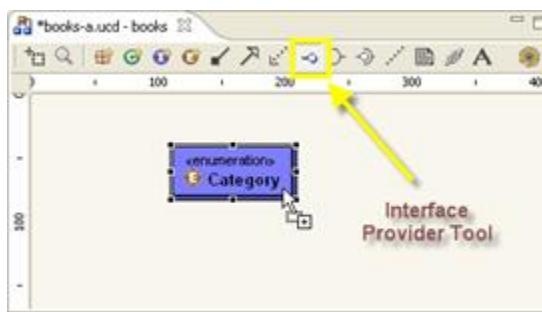
    private String name = "";

    private Category(String name) {
        this.name = name;
    }

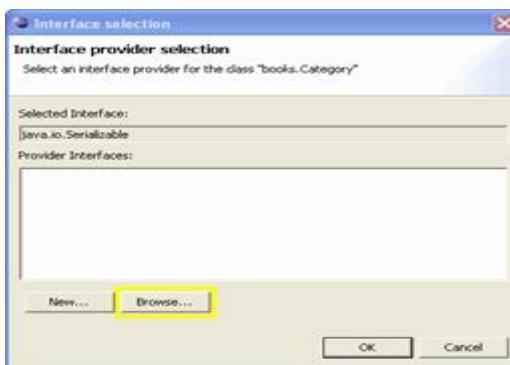
    /**
     * Getter of the property <tt>name</tt>
     *
     * @return Returns the name.
     * @uml.property name="name"
     */
    public String getName() {
        return name;
    }
}
```

3.3. Interface provider connection

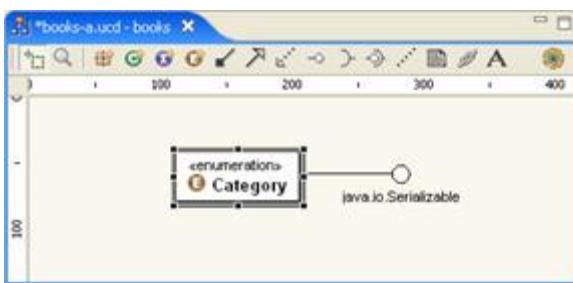
An enumeration can implement an interface so it can be an interface provider. Like a class, it is possible to create an interface provider via the tool bar. In our example, suppose we need a serializable enumeration. Select the “interface provider” tool icon and then highlight the enumeration.



A new interface selection dialog is shown.



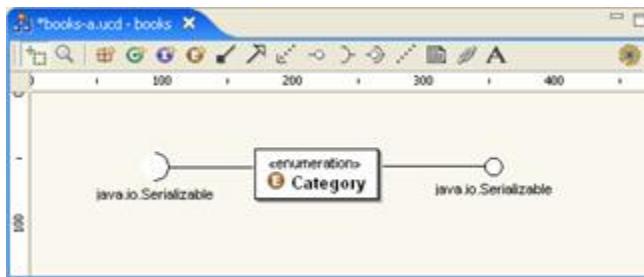
To create a new interface, click on the button “New...”. Or select an existing one via the button “Browse...”. In our case, we select the existing one **java.util.Serializable**.



This operation in fact creates an implementation relationship between the enumeration and the interface. Here is the generated code:

```
Package books;

public enum Category implements java.io.Serializable { }
```



In the same way, we can create a “required interface”. Here is the code generated:

```
Package books;

/**
 * @uml.dependency supplier="java.io.Serializable"
 */
public enum Category implements java.io.Serializable
{
    History;
}
```

For an enumeration with an “interface provider” and “required interface”, it is possible to establish the connection by dragging one interface icon into another or using the “Interface connection” icon in the tool bar to create the connection in one step.



Generic Type

The generic type support touches three levels:

- Kernel to capture the type defined in generic type and create the corresponding UML type using the Templates package. From the user's perspective, this capability is hidden.
 - Display the template type in the diagram
 - Templates for Java Code generation
1. [Template type presentation](#)
 1. [Template definition](#)
 2. [Template binding](#)
 2. [Java Code generation](#)
 1. [JDT template](#)
 2. [Velocity template](#)

1. ***Template type presentation***

The template support deals with two issues: Template definition and Template binding.

1.1 ***Template definition***

The first requirement of this integration is to capture the generic type data, convert it into UML and then display it in a UML diagram. Here is an example of **java.util.Collection**.

In the diagram above, we find the class template, operation template and operation parameter template. In the class template, “**E**” is a template variable type, which is used in the method **add(E)** .

We can also find an operation template using a local template variable:

```
<T> T[] toArray(T[] a);
```

where the variable “**T**” is an operation template variable. It is used in the scope of this method.

1.2 ***Template binding***

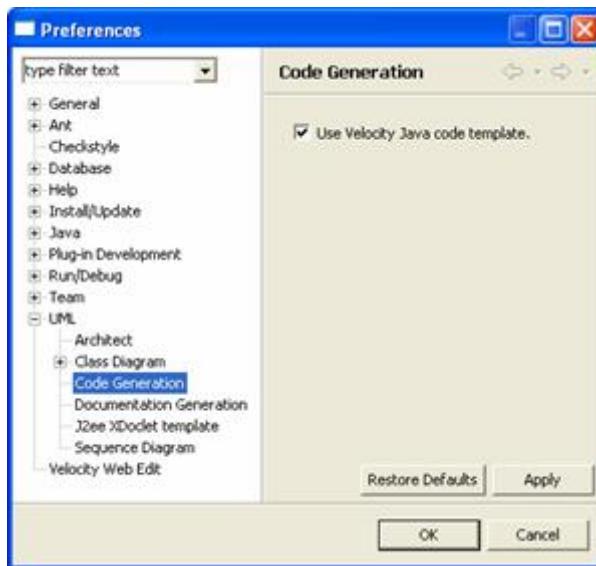
Besides the template definition, it is necessary to show the template utilization relationship, called template binding. In the above diagram, this relationship is shown as an arrow “**E -> ?**”. The first example is the following method:

```
boolean retainAll(Collection<?> c);
```

The class template variable “**E**” of the collection binds to an unspecified type in the argument for this method.

2. Java Code generation

EclipseUML uses a template engine to generate the Java code. In the Free edition, the template engine is JDT's one, which provides only getter/setter. In Studio edition, it is possible to use the powerful Velocity template engine. Of course, it can be disabled in the Preferences.



2.1 JDT template

JDT template provides a getter/setter code generation pattern. This pattern is used by the EclipseUML Free edition or when the Velocity template engine is disabled in the Studio edition.

Here is the standard JDT template pattern.

```
$(modifiers) ${field_type} ${field};  
  
/**  
 * @return Returns the ${bare_field_name}.  
 */  
$(modifiers) ${field_type} get${bare_field_name} () {  
    return ${field};  
}  
  
/**  
 * @param ${param} The ${bare_field_name} to set.  
 */  
$(modifiers) void set${bare_field_name} (${field_type} ${param}) {  
    ${field} = ${param};  
}
```

In the case of the type **Boolean**, the prefix of the getter is “**is**” instead of “**get**”.

The key-words in blue color and bold/italic enclosed in brackets followed by a \$ character, are template variables. Here is the detailed description of each variable

Variable name	Description
\$(modifiers)	Java member such as public, protected,

	private, final static etc...
<code> \${field_type}</code>	Field member type
<code> \${field}</code>	Field member name
<code> \${bare_field_name}</code>	Normalized file name by removing the prefix/suffix defined in Preferences->Java->Code Style. In general, the first letter is capital.
<code> \${param}</code>	The parameter name based on the prefix/suffix defined in Preferences->Java->Code Style.

EclipseUML inserts two new variables for UML model storage purposes:

Variable name	Description
<code> \$(uml_property_specification)</code>	This variable contains all UML information about the property such as property name, cardinality, stereotype, dependencies etc...
<code> \${uml_property_member}</code>	This variable is used to mark the property membership of an accessor. The syntax is <code> @uml.property name="\${property_name}"</code>

The template patterns are changed as following:

```

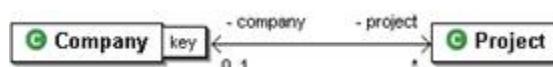
/**
 * ${uml_property_specification}
 */
${modifiers} ${field_type} ${field};

/**
 * @return Returns the ${bare_field_name}.
 * ${uml_property_member}
 */
${modifiers} ${field_type} get${bare_field_name} () {
    return ${field};
}

/**
 * @param ${param} The ${bare_field_name} to set.
 * ${uml_property_member}
 */
${modifiers} void set${bare_field_name} (${field_type} ${param}) {
    ${field} = ${param};
}

```

In the following model, the Company and Project have a qualified association. The field type of this association is designed as **java.util.Map**.



The generated codes based on the JDT templates are as follows:

```

private Map<Object, Collection<Project>> projectMap;
```

```

public Map<Object, Collection<Project>> getProject()
{
    return projectMap;
}
```

```

*/
public void setCompany(Map<Object, Collection<Project>> project) {
    this.projectMap = project;
}

```

2.2 Velocity template

On top of the velocity template engine, EclipseUML Studio provides more powerful template capabilities. Not only can users easily change the template definition, but also override the template definition or/and change the application scope at project, package, class and even property granularity.

Three template files are provided to handle three kinds of collection:

Variable name	Template file name	Description
java.util.Collection	uml2.Property- java.util.Collection.vm	General collection template
java.util.List	uml2.Property- java.util.List.vm	General ordered collection template.
Java.util.Map	uml2.Property- java.util.Map.vm	Qualified association template.

Here is the outline of the methods defined in these files. Please reference the Studio documentation for detailed information.

Collection template definition

```

{ElementType} get{PropertyName};
boolean is{PropertyName}Empty;
boolean contains{PropertyName}({ElementType} o);
Iterator<{ElementType}> {PropertyName}Iterator();
{ElementType} [] toArray();
<T extends {ElementType}> T[] toArray(T[] a);
boolean add({ElementType} o);
Object remove({ElementType} o);
boolean containsAll{PropertyName}(Collection<? extends {ElementType}> c);
boolean addAll{PropertyName}(Collection<? extends {ElementType}> c);
void clear{PropertyName}();

```

List template definition

```

int {PropertyName}Size();
boolean is{PropertyName}Empty();
boolean contains{PropertyName}({ElementType} o);
Iterator<{ElementType}> iterator();
{ElementType} [] toArray();
<T extends {ElementType}> T[] toArray(T[] a);
boolean add{PropertyName}({ElementType} o);

```

```

Object remove(PropertyName) (Object o);
{ElementType} get(PropertyName) (int index);
void add(PropertyName) (int index, {ElementType} element);

```

Map template definition

```

intPropertyNameSize();
boolean isPropertyNameEmpty();
boolean containsKey(PropertyName) ({Qualifier.Key} o);
boolean containsValue(PropertyName) ({Qualifier.Value} value);
{Qualifier.Value} get(PropertyName) ({Qualifier.Key} key);
{Qualifier.Value} put(PropertyName) ({Qualifier.Key} key,
{Qualifier.Value} value);
{Qualifier.Value} remove(PropertyName) ({Qualifier.Key} key);
Set<{Qualifier.Key}> keySet(PropertyName) ();
Collection<{Qualifier.Value}> values(PropertyName) ();

```

Back to the same example as for JDT template, the new version of the implementation is as follows:

```

private Map<Object, Collection<Project>> projectMap;
public Map<Object, Collection<Project>> getProject() {
    return projectMap;
}
public Set<Object> projectKeySet() {
    return projectMap.keySet();
}
public Collection<Collection<Project>> projectValues() {
    return projectMap.values();
}
public boolean projectContainsKey(Object object) {
    return projectMap.containsKey(object);
}
public boolean projectContainsValue(Collection<Project> project) {
    return projectMap.containsValue(project);
}
public Collection<Project> getProject(Object object) {
    return (Collection<Project>) projectMap.get(object);
}
public boolean isProjectEmpty() {
    return projectMap.isEmpty();
}
public int projectSize() {
    return projectMap.size();
}
public void setProject(Map<Object, Collection<Project>> project) {
    projectMap = project;
}
public Collection<Project> putProject(Object object, Collection<Project> project) {
    return (Collection<Project>) projectMap.put(object, project);
}
public Collection<Project> removeProject(Object object) {
    return (Collection<Project>) projectMap.remove(object);
}
public void clearProject() {
    projectMap.clear();
}

```

Persistence Development

- [Activate EclipseUML in the JPA perspective](#)
- [Create new JPA UML classes](#)
- [Add JPA properties to a class, attribute and method](#)
- [Create Associations between Classes](#)
- [Reverse an existing Database into the class diagram](#)
- [Use Oracle Enterprise Pack for Eclipse 11g](#)
- Using JPA with EclipseUML (6min) : [Flash demo](#)
- From UML to Database (7min): [Flash demo](#) / [.exe file](#)
- From Oracle to UML 2.1 (6min): [Flash demo](#) / [.exe file](#)

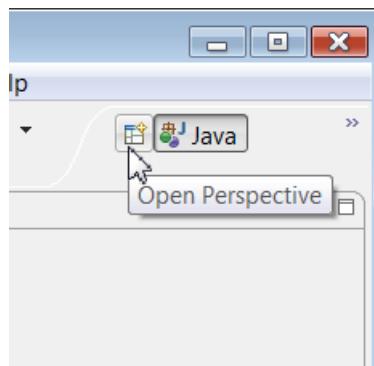
Activate EclipseUML in the JPA Perspective

EclipseUML could be directly activated inside the JPA Perspective.
This section is about:

1. [How to activate the JPA Perspective](#)
2. [How to create a Class diagram](#)
3. [How to activate the Class Diagram Persistence Development](#)

We recommend using Dali and JPA Perspective with EclipseUML, and not only the modelling perspective, in order to take the full advantage of the tight UML integration within Eclipse with JPA. For example, the reverse of existing Database is generated by Dali, and then EclipseUML upload all information in the class diagram. It allows to extend the model and generate metamodel + Java Annotation in the code. Finally powerful mapper such as toplink or Hibernate can generate code for Oracle, MySQL etc...

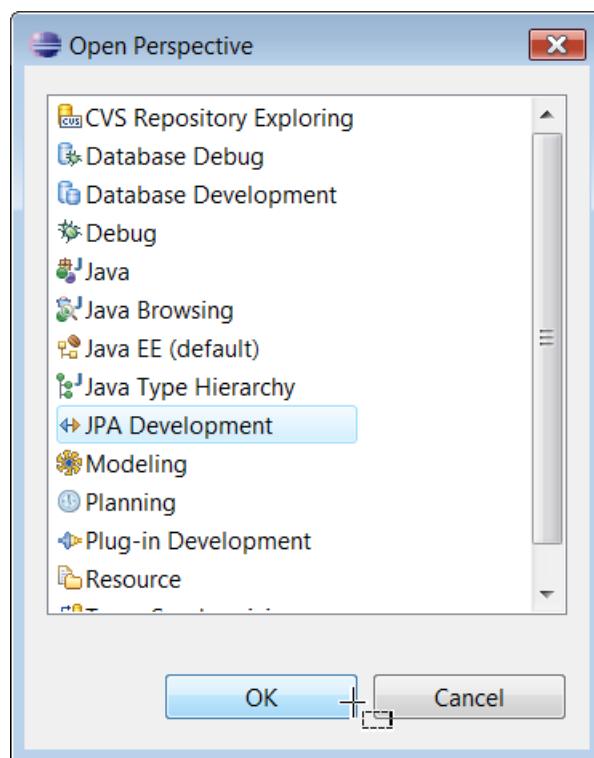
1. To activate the JPA perspective you need to **click on the Open Perspective icon** in the upper right corner of Eclipse.



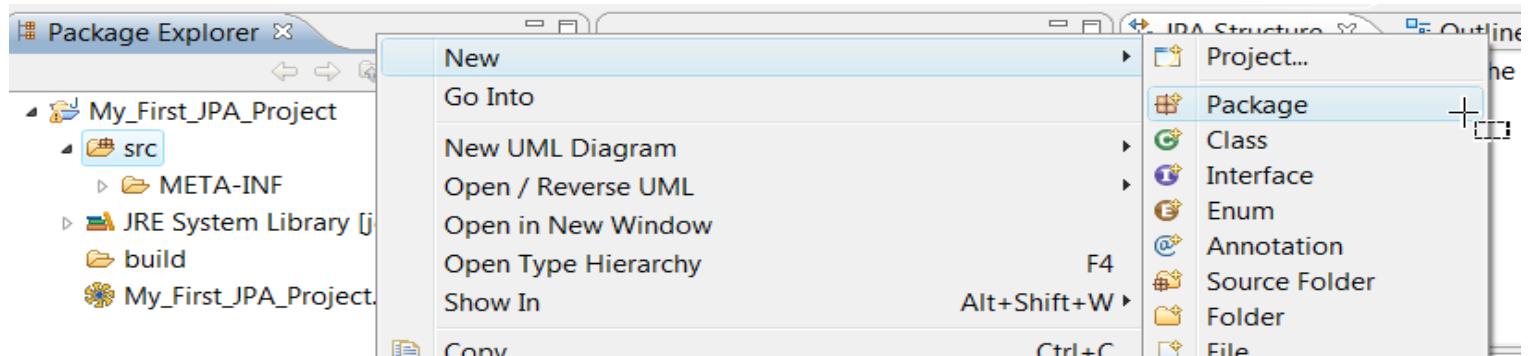
Select Other...



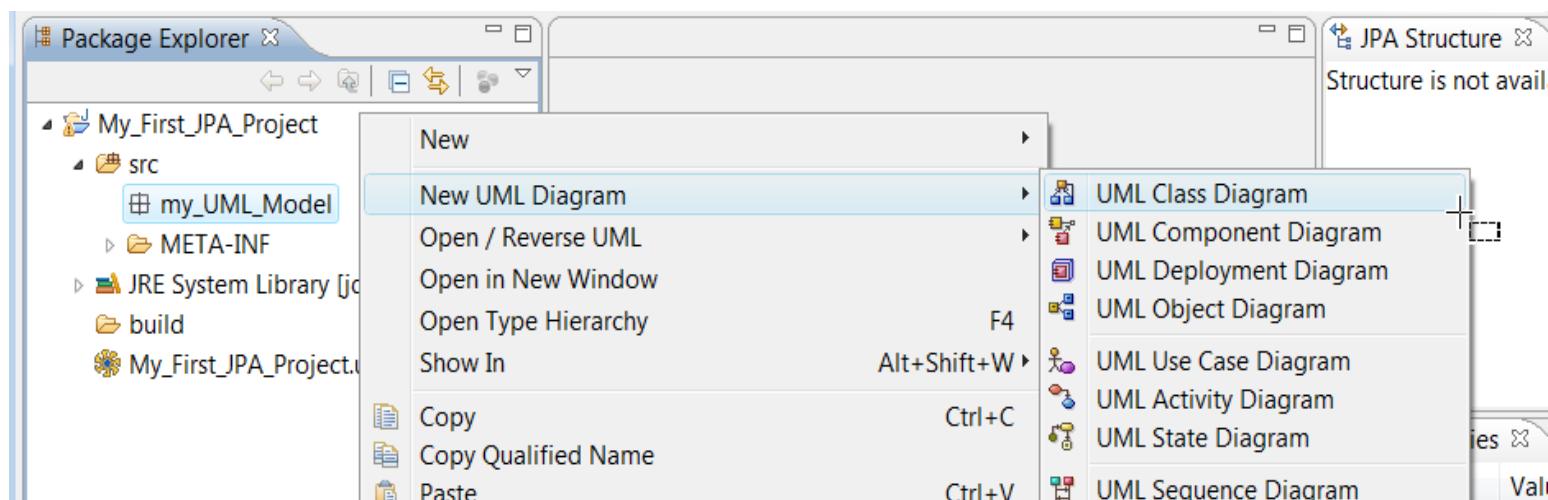
Select JPA Development



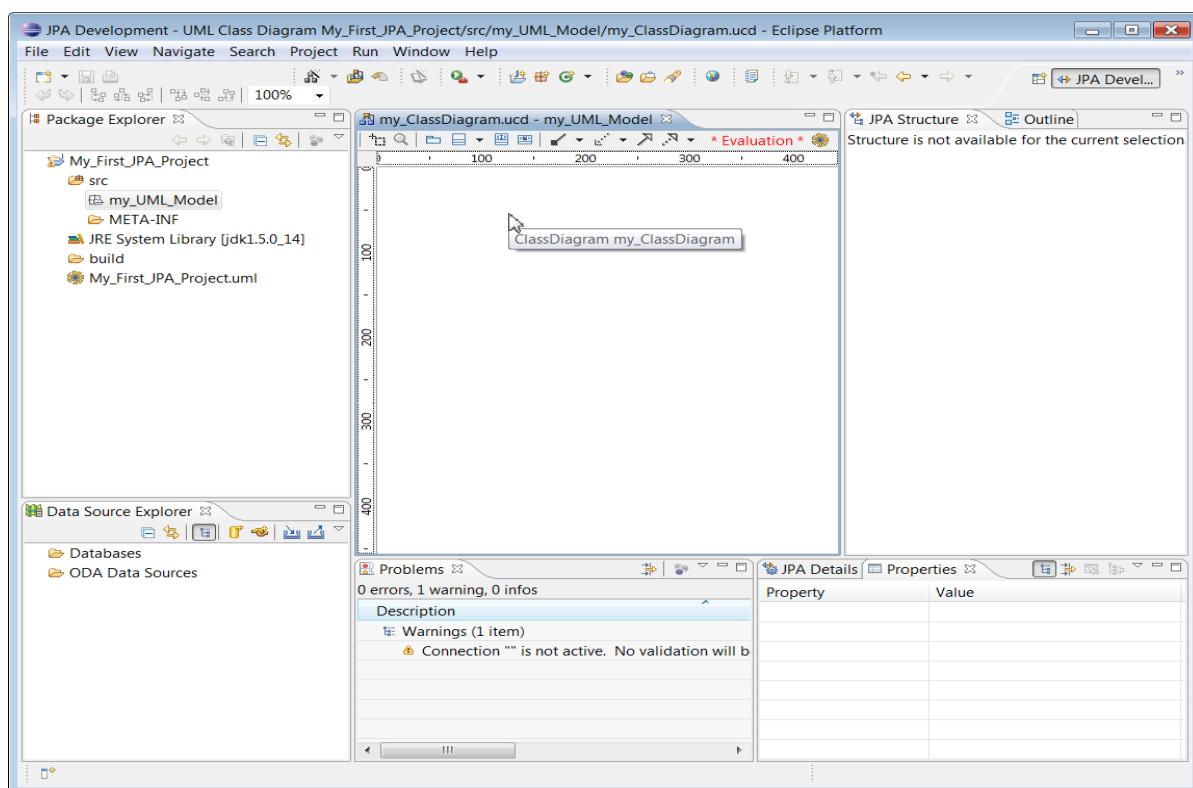
2. Immediately after the click on the OK button, Eclipse switch to JPA Perspective.
Go in the Package Explorer and create a JPA project, then add a package to your src in order to activate the class diagram creation.
You can not create class diagram if you don't create a package.



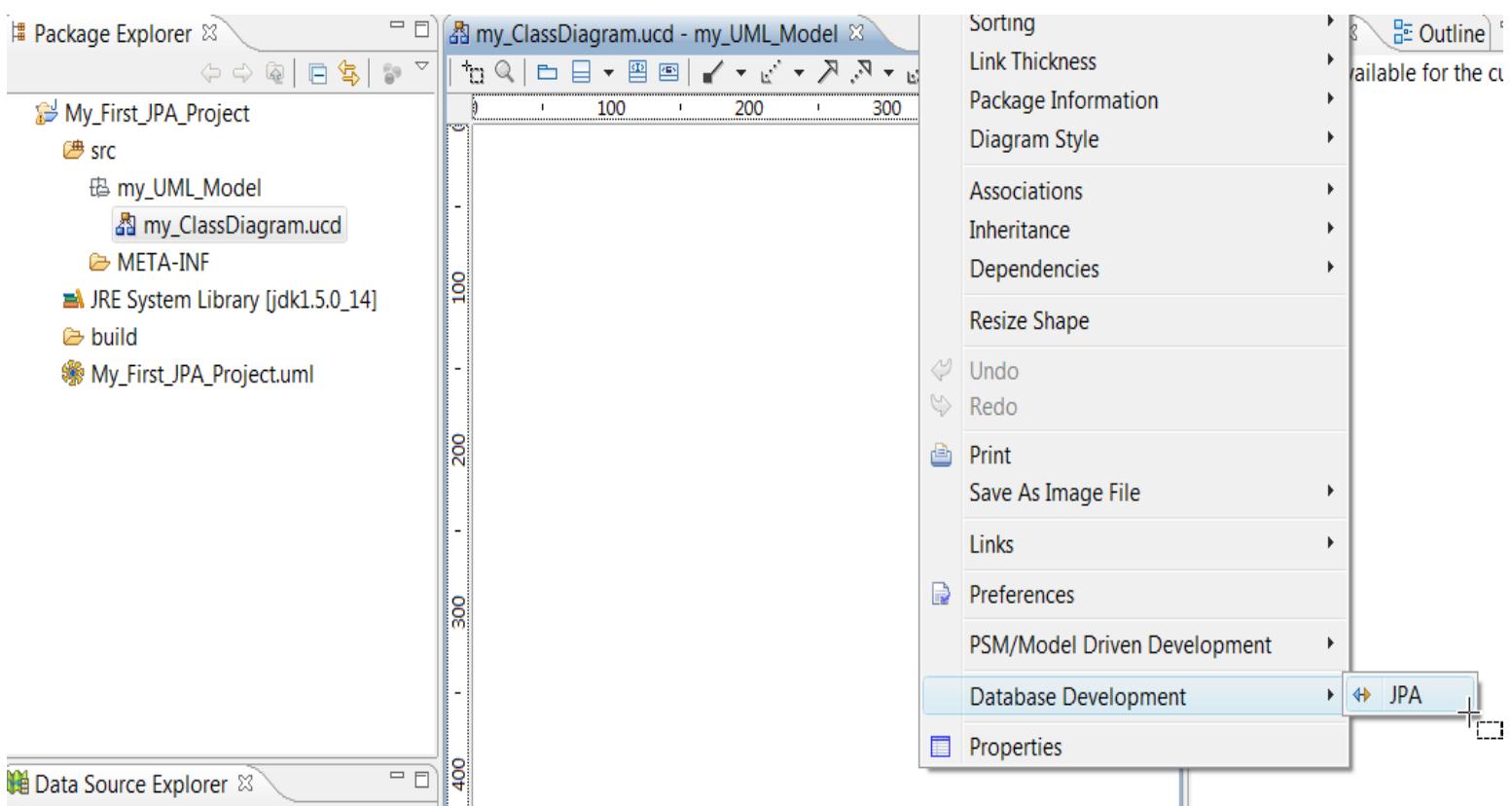
Click on the package and select **New UML Diagram > UML Class Diagram**



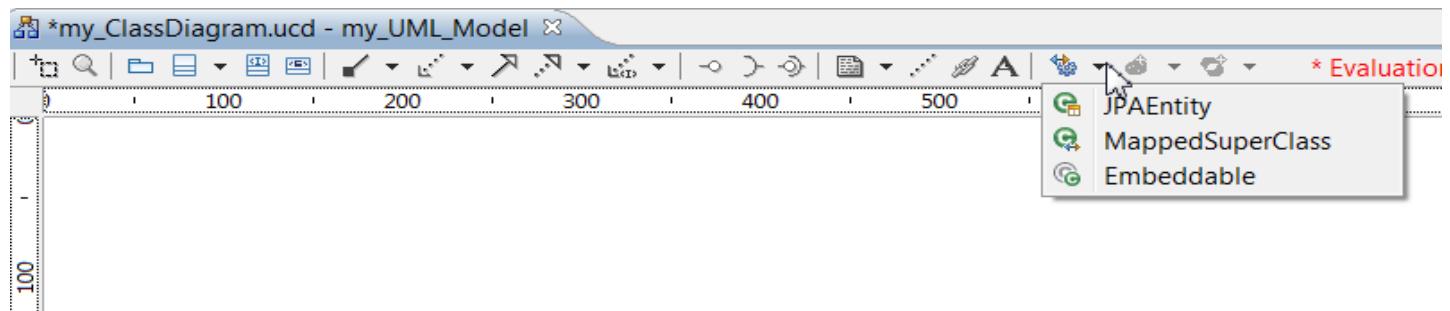
Your Class Diagram has been created inside the JPA Perspective.



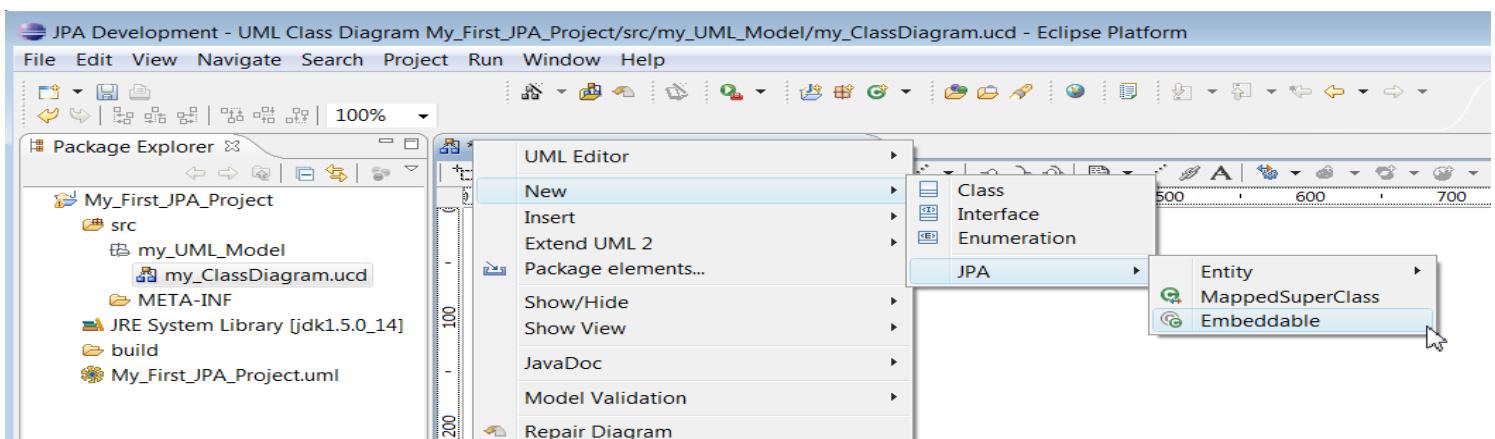
3. In order to activate the Database preferences inside EclipseUML, click on the class diagram background and select **Persistence Development > JPA**.



Once the JPA menu is selected, **the JPA toolbar** is immediately activated in the Class diagram



And the class diagram contextual menu **New > JPA** is now available.



Create New JPA UML Classes

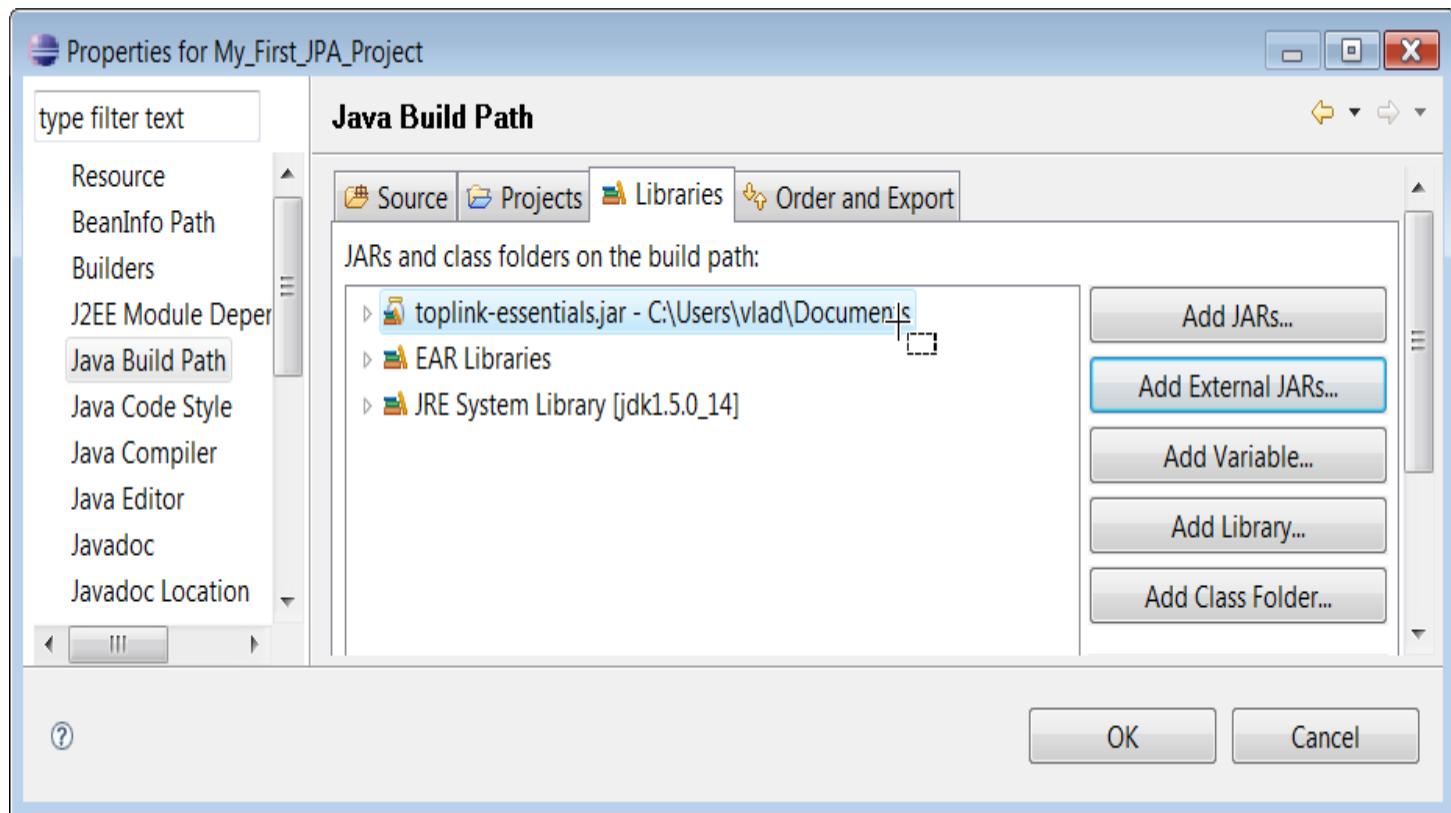
This section is composed by:

1. [Add JDK 5/6 annotations support to your project](#)
2. [Create a JPA Class from the Toolbar](#)
3. [Create a JPA Class from the Class Diagram contextual menu](#)
4. [Extend existing Class by adding JPA properties using EclipseUML or Dali plugin](#)
5. [Why my new JPA Class is in error](#)

1. Add JDK 5/6 annotation support to your project properties

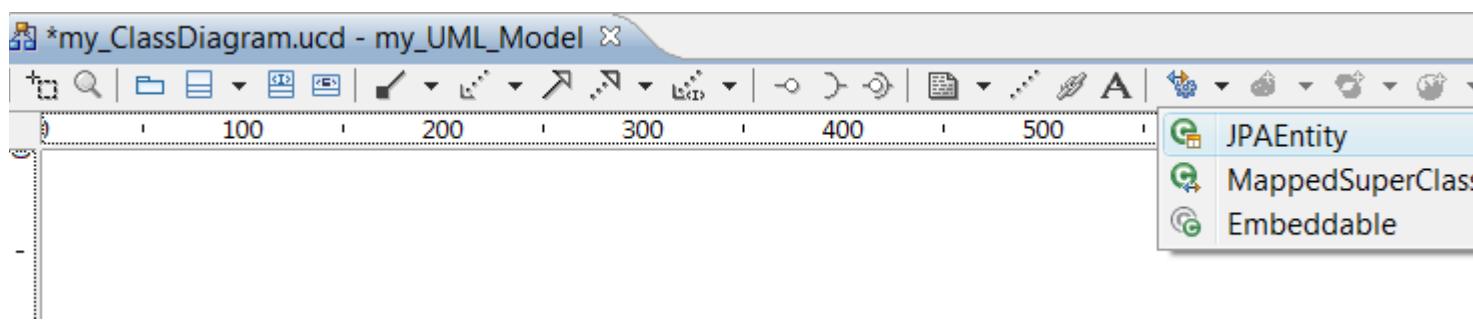
Don't forget to add the toplink-essential or any other annotation jar to your JPA project property in order to be able to manage JDK 5/6 annotations in your java code.

Click on the **JPA Project > Property > Java Build Path > click on the Libraries tab > Add External Jars...**

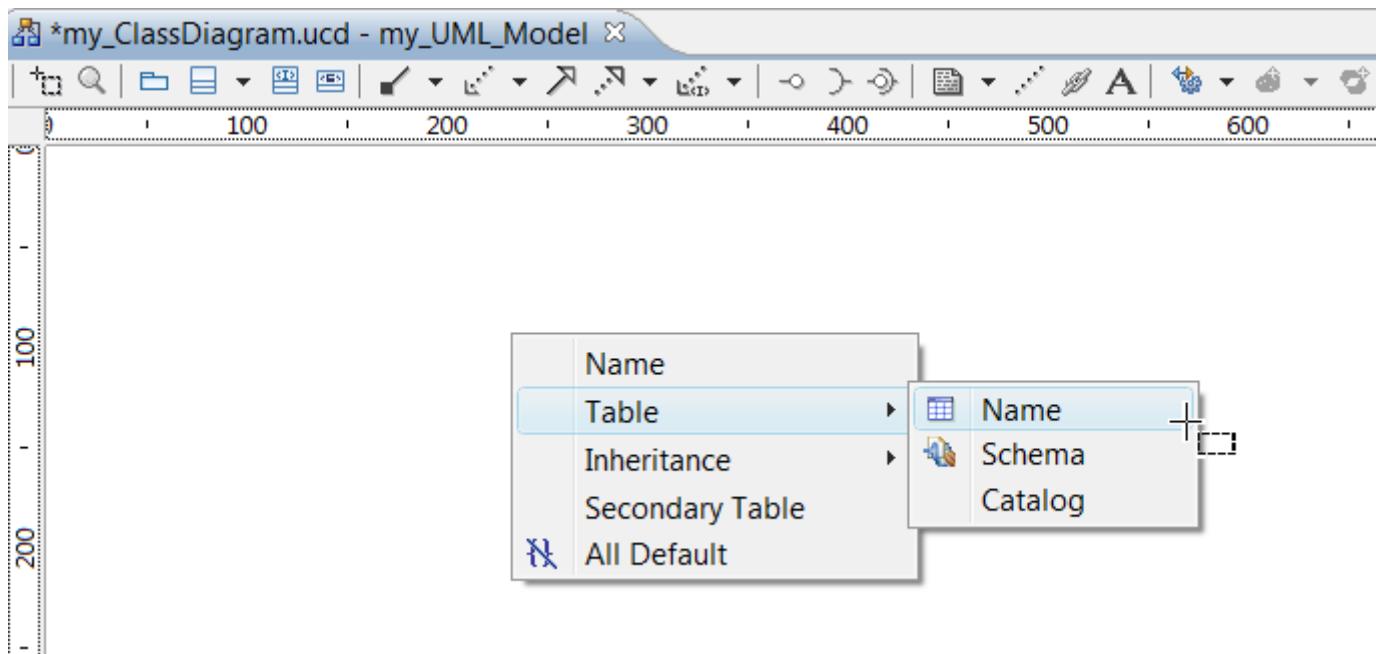


2. Create a JPA Class from the Editor Toolbar

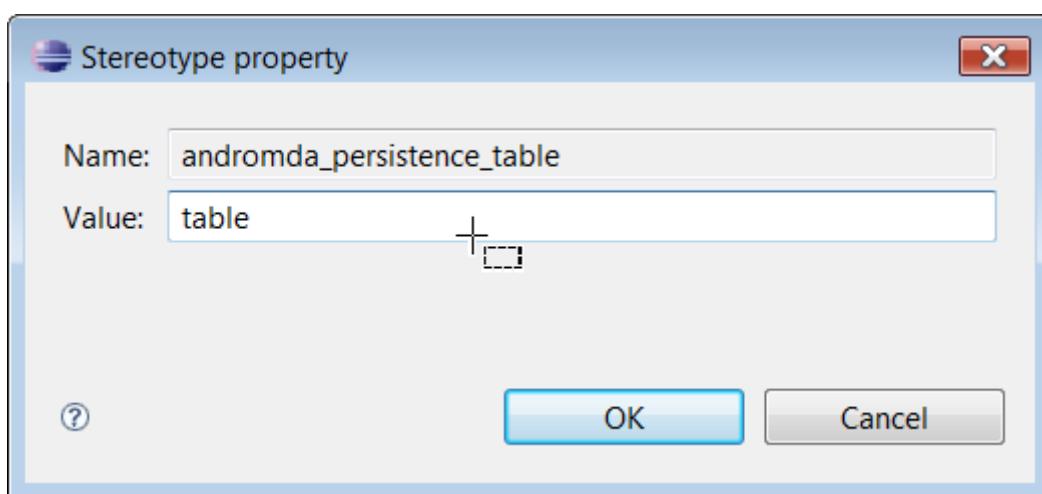
Click on the JPA icon in the Class Diagram Editor.
Select JPA Entity for example.



Select the Entity type from the list.
Select Table > Name for example.



Enter the Stereotype Value.
Enter table in the Value Field for example.



Enter the name of the JPA Id.
Enter Id in the Name field for example.

New JPA Id

Create a new Entity Id

Properties Tagged Values Javadoc

Name: id

Type: long

Generic Type Parameter

Dimension: 0

Attribute

Visibility: public protected package private

Default value

Modifiers: static final abstract transient volatile

Accessors

Use accessors Read only

Visibility: public protected package private

Generated Value

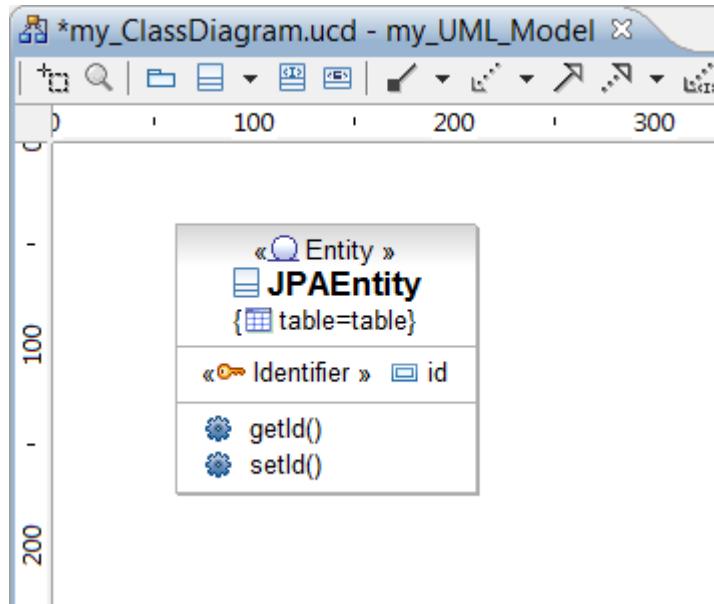
Generator Type: None Auto Identity Table Sequence

Generator Name:

?

OK Cancel

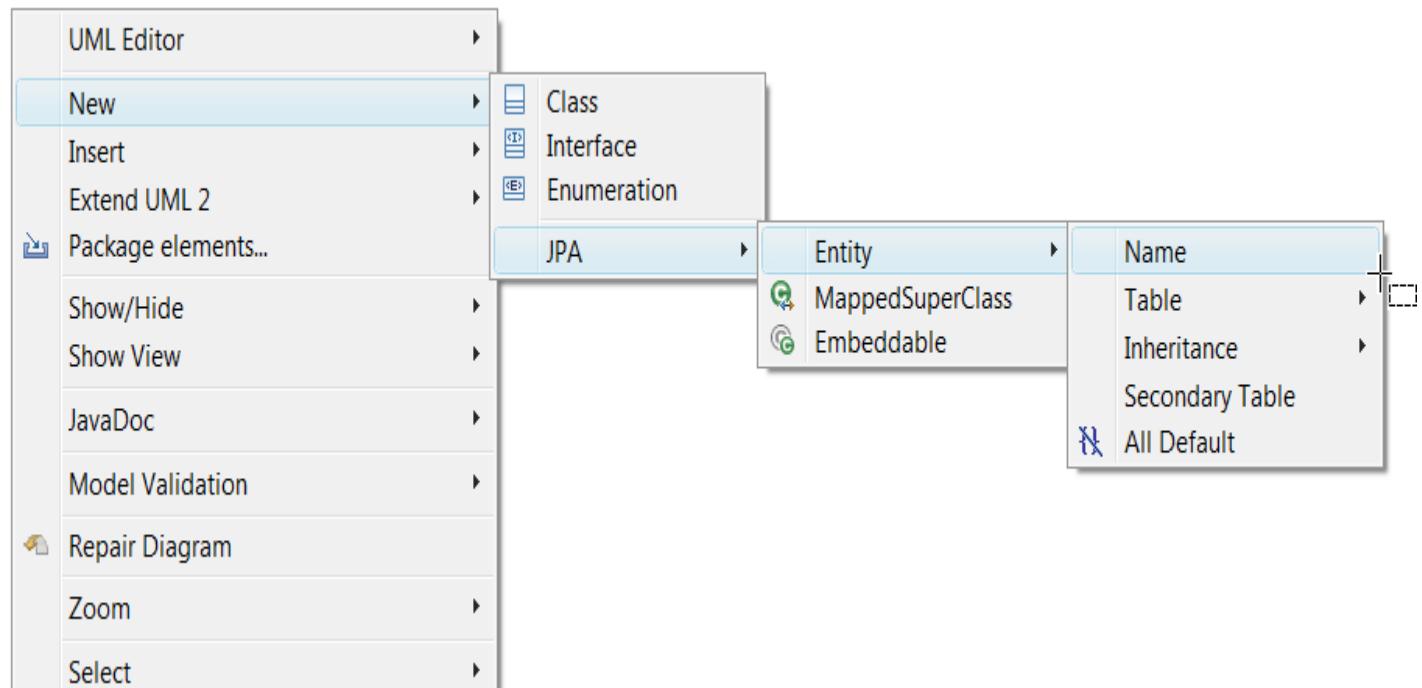
Your JPA Class has immediately been created in your Class Diagram and in your Java code.



3. Create a JPA Class from the Class Diagram contextual menu.

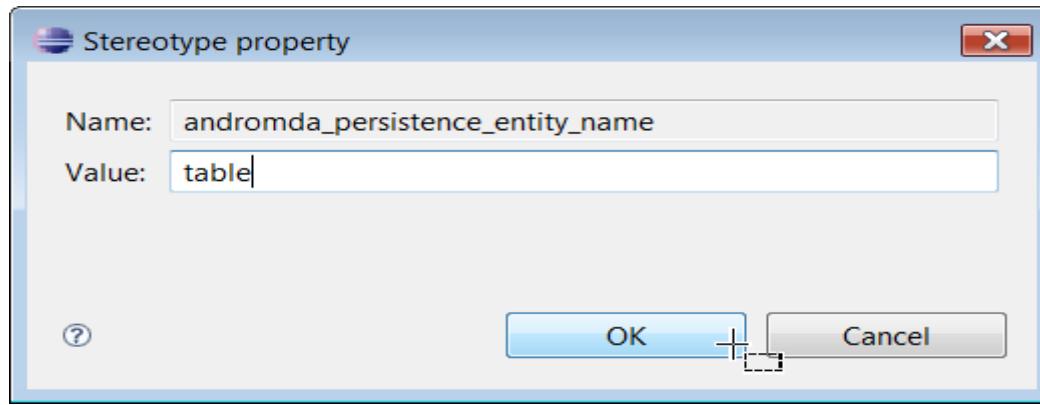
Click on the Class Diagram background > New.

Select New > JPA > Entity > Name for example.



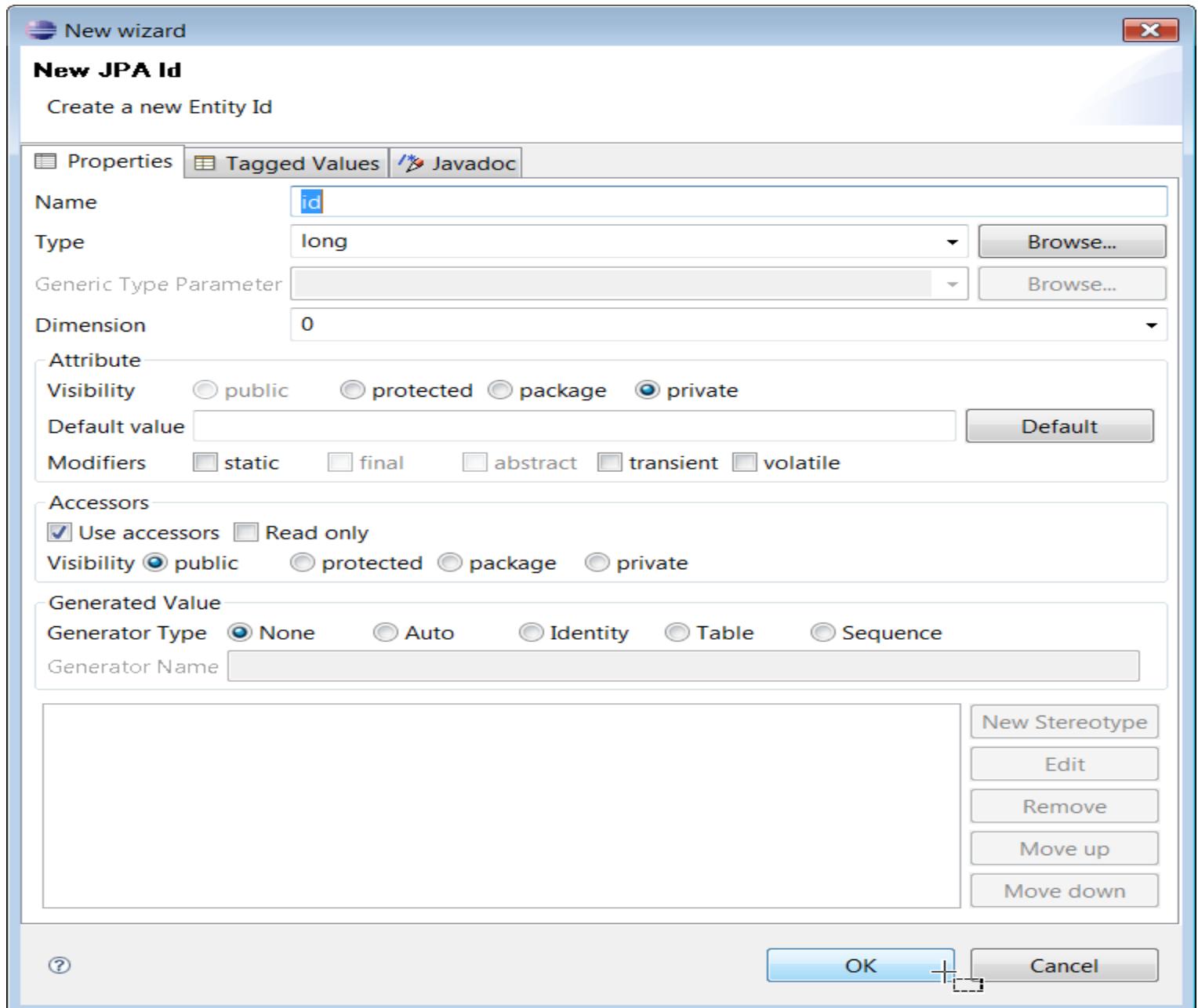
Enter the Stereotype Value.

Enter table in the Value Field for example.

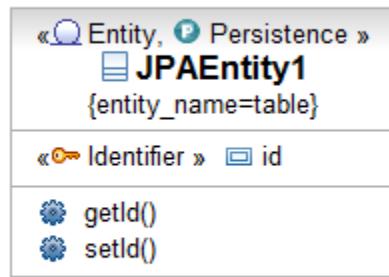


Enter the name of the JPA Id.

Enter Id in the Name field for example.



Your JPA Class has immediately been created in your Class Diagram and in your Java code.



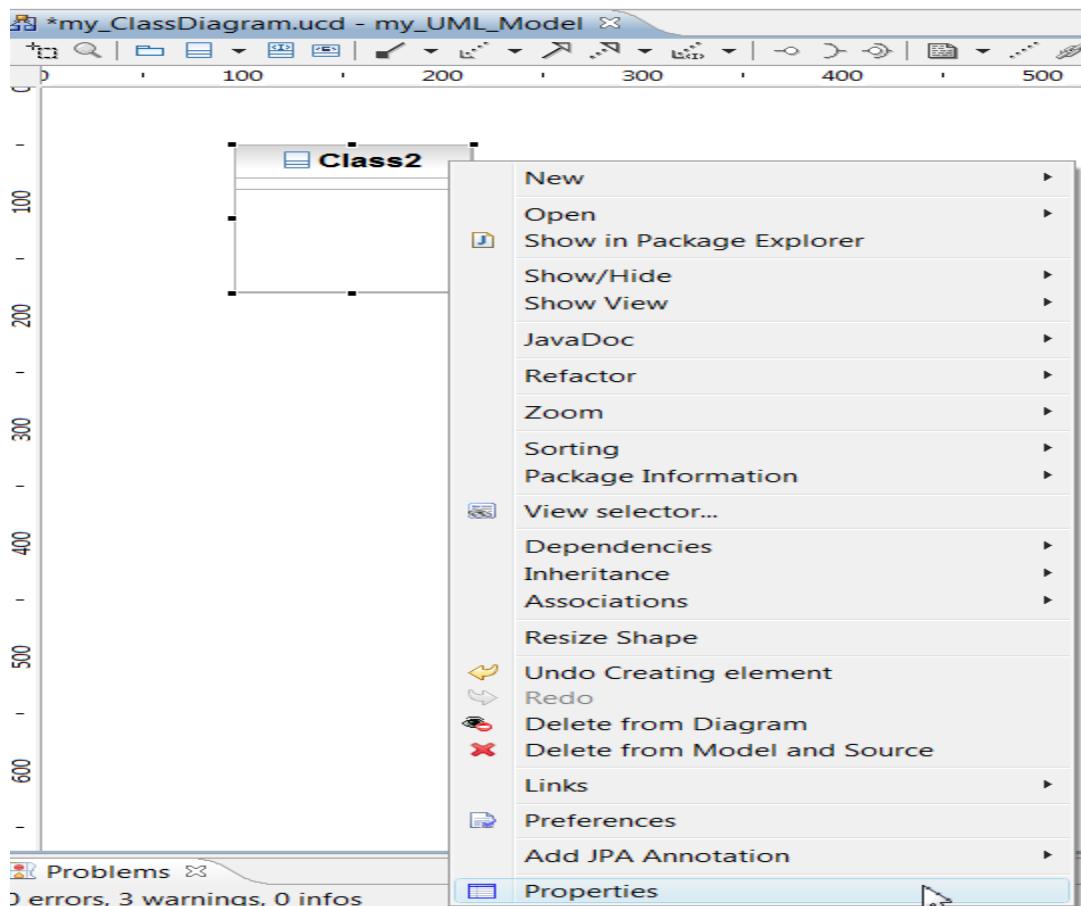
4. Extend existing Class by adding JPA properties using:

1. **Adding EclipseUML stereotype into the UML model** and getting live code synchronization to add JPA annotation in your code.

You can extend JPA properties to existing classes.

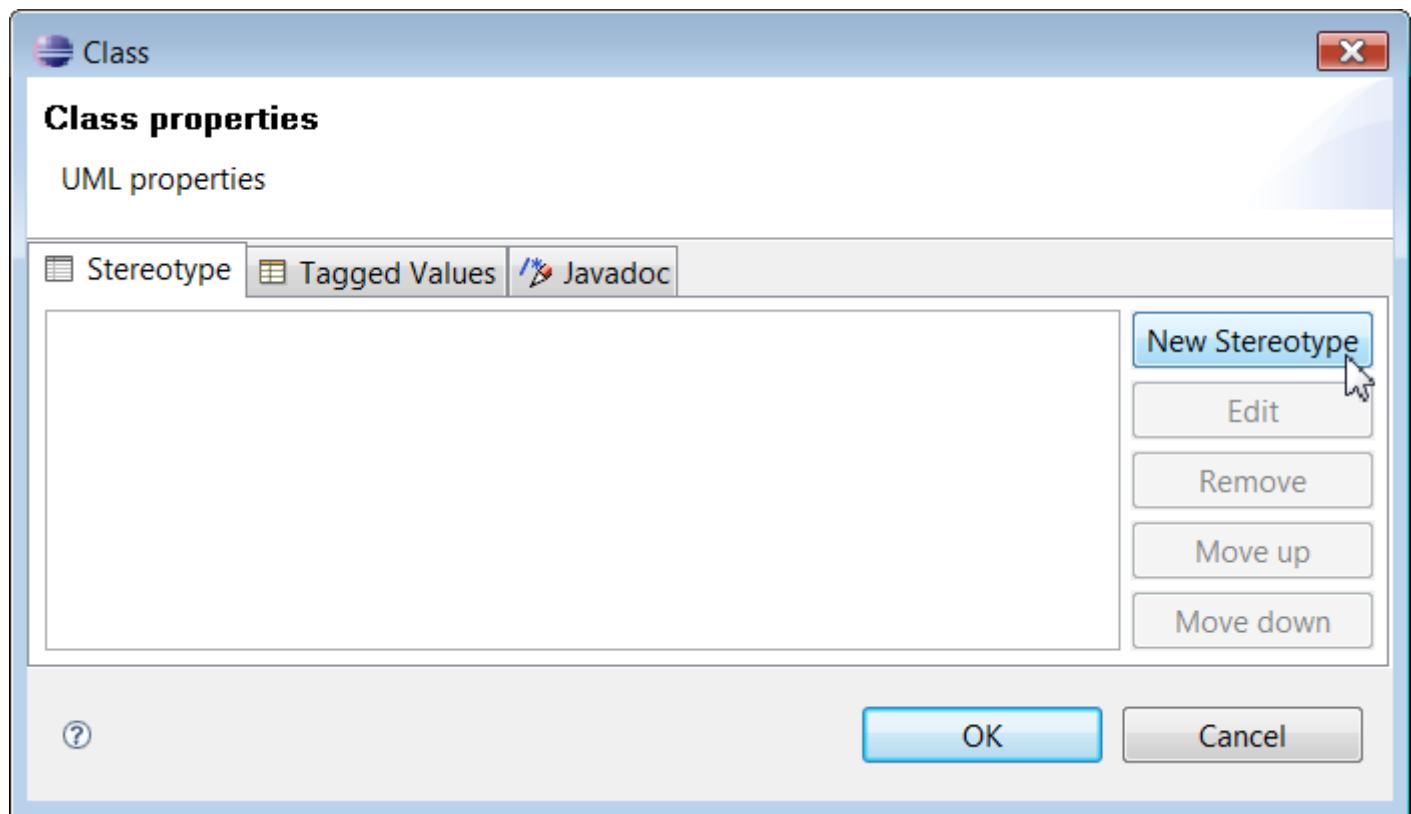
For example, click on class2 and add persistence property to this class.

Click on **Class2 > Properties**



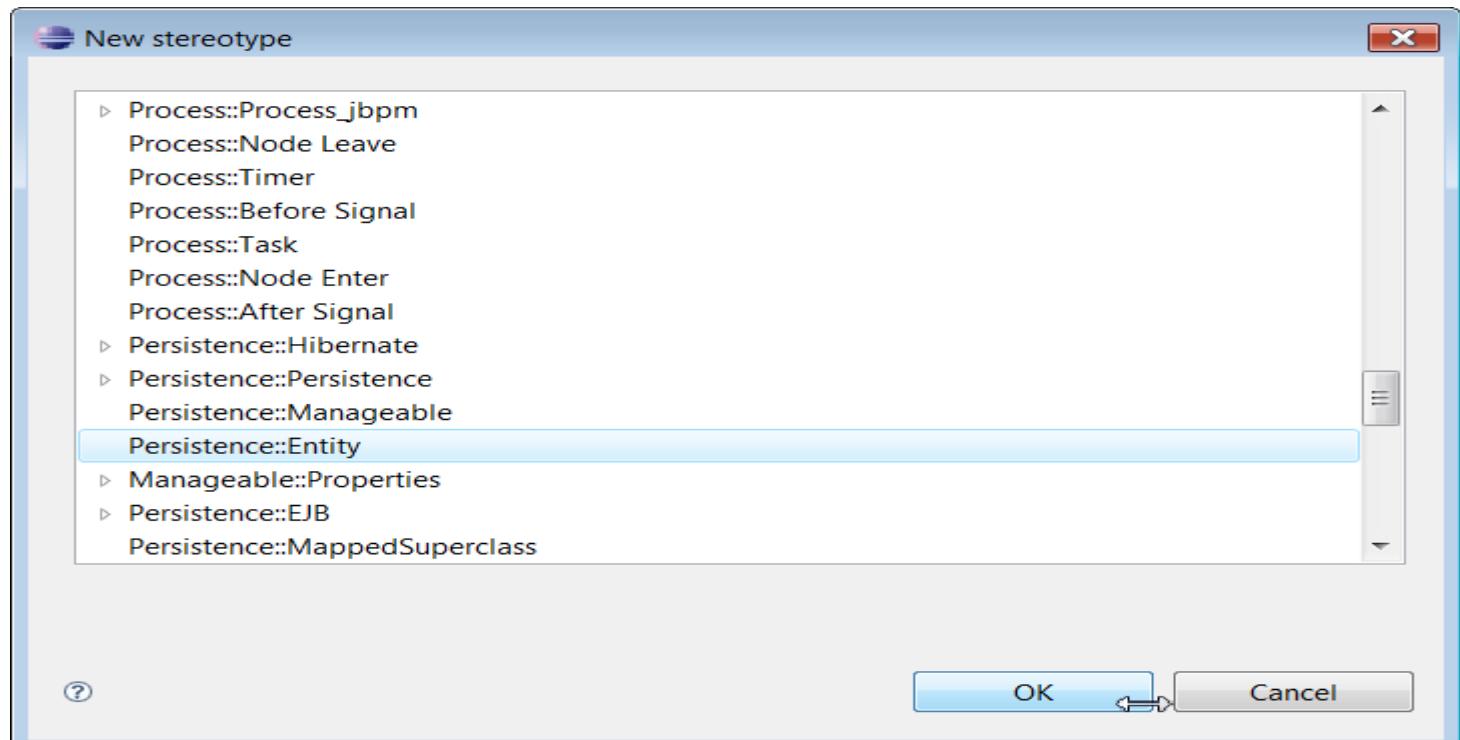
Add a new stereotype to your class.

For example add persistence properties to your class by selecting **New Stereotype**.



Select Persistence::Entity to map a Class and add persistence properties.

It means that this class will be saved as a table in your database.

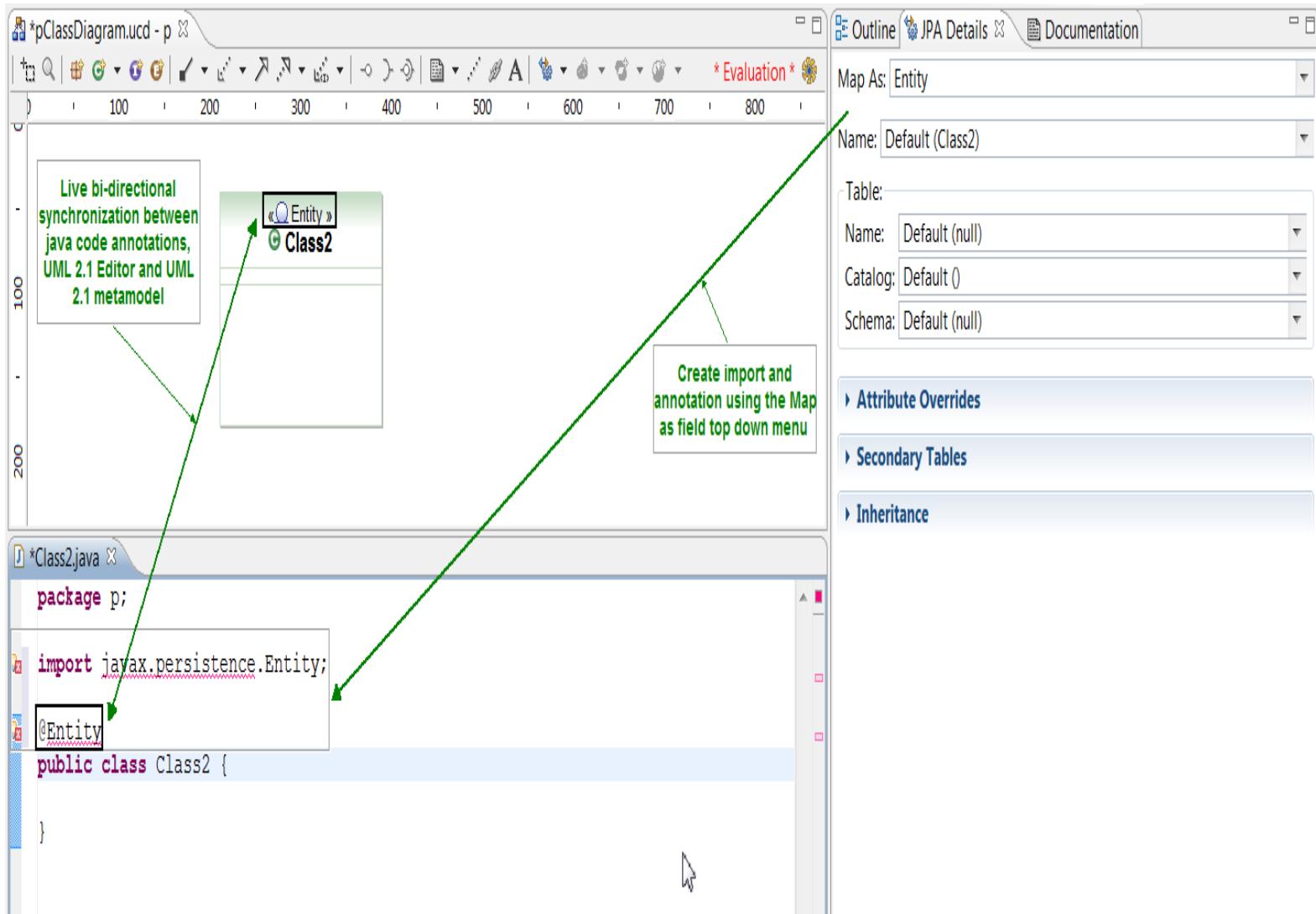


1. Using Dali plugin, or writing code to add annotation in your java code and getting immediate live from code to model synchronization.

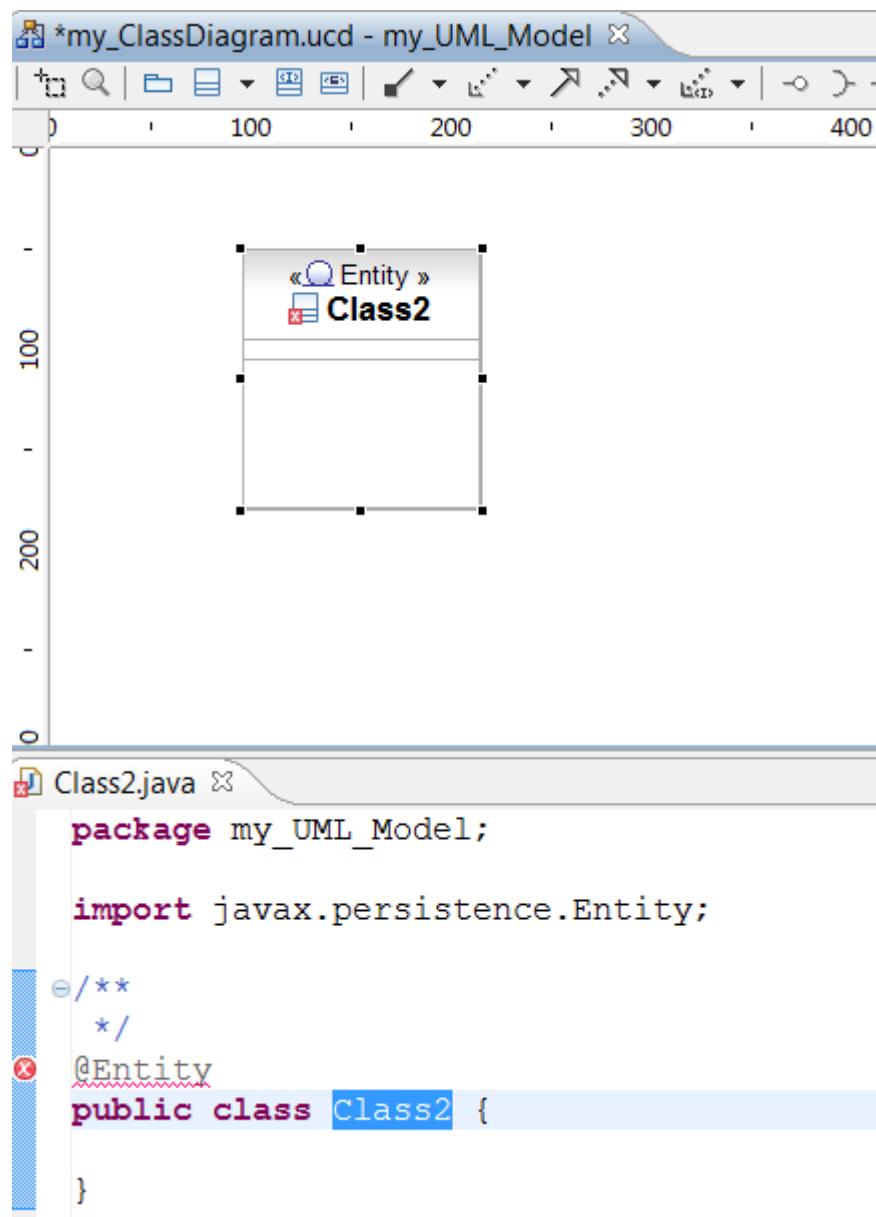
The Dali annotation will immediately be saved inside the UML 2.1 metamodel if you just add a JPA annotation in your code using the live code and model synchronization.

Create a class using the UML Editor or the Package Explorer menu. Note that the JPA details synchronization is using the Java Code Editor; therefore you need to select the Class either in the code, or double click in the class diagram on the Element.

- If you type @Entity directly in your code, the UML Editor and the UML Metamodel will be immediately updated
- If you use the Map As top down menu and add Entity, then this annotation will be added to your code and will update the UML Editor and the UML Metamodel.

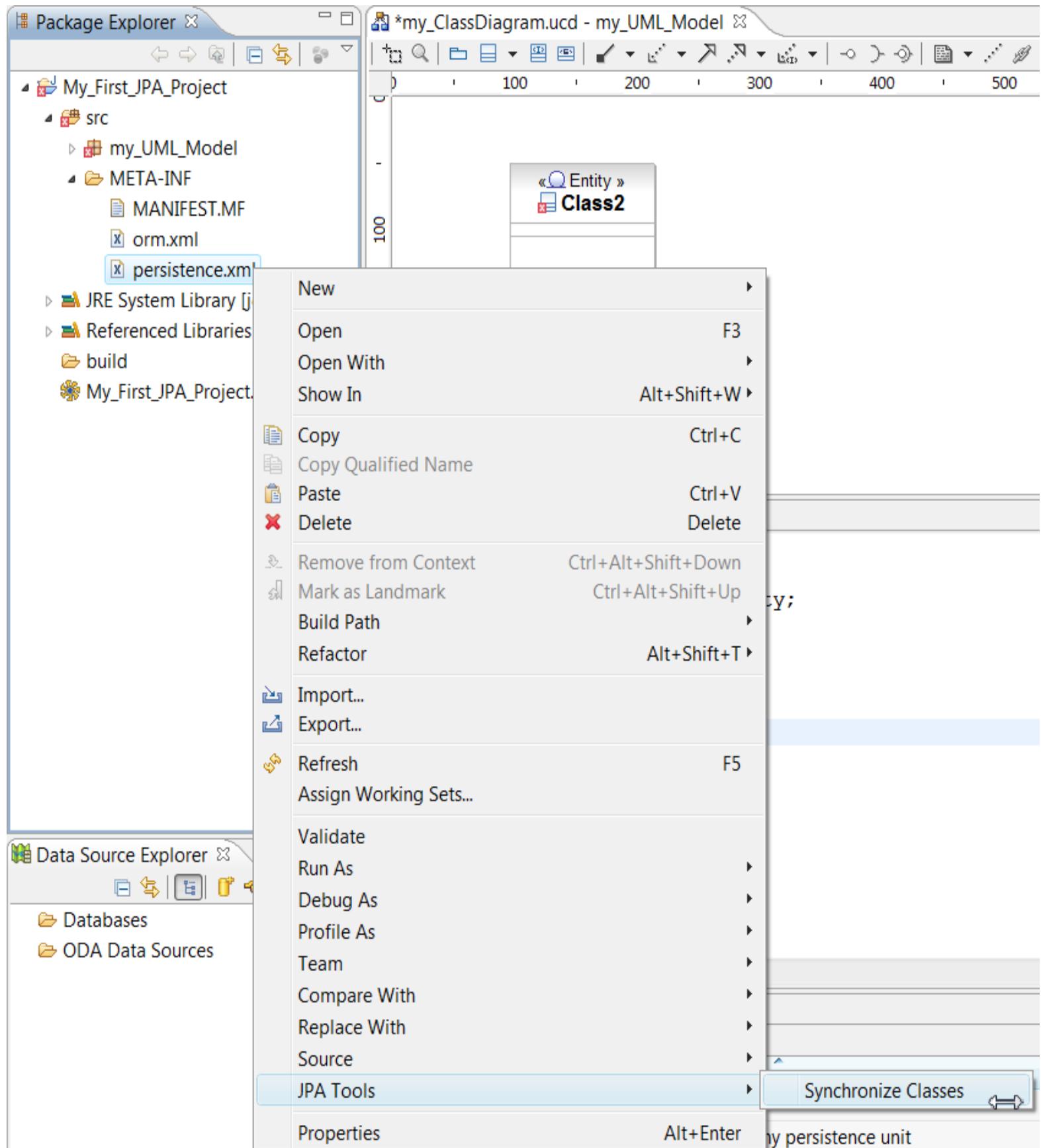


The Class2 has been updated.

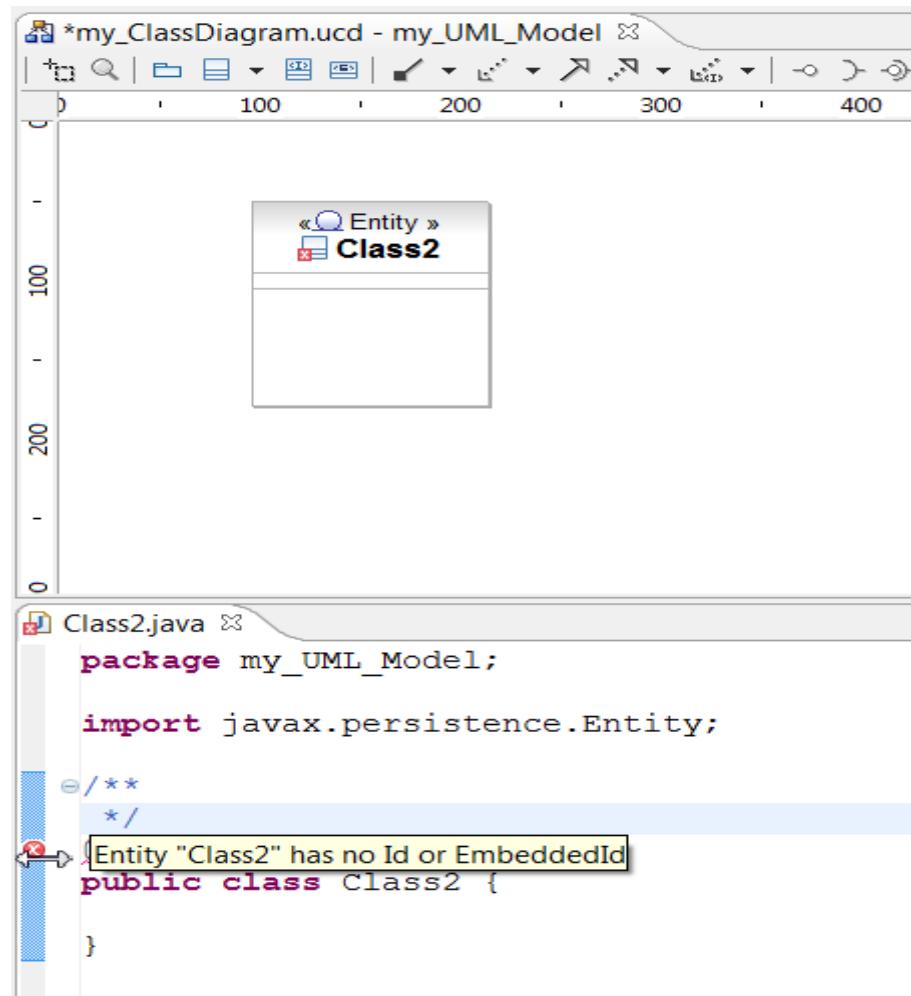


An error is in the code because the mapped class is not contained in any persistence unit.

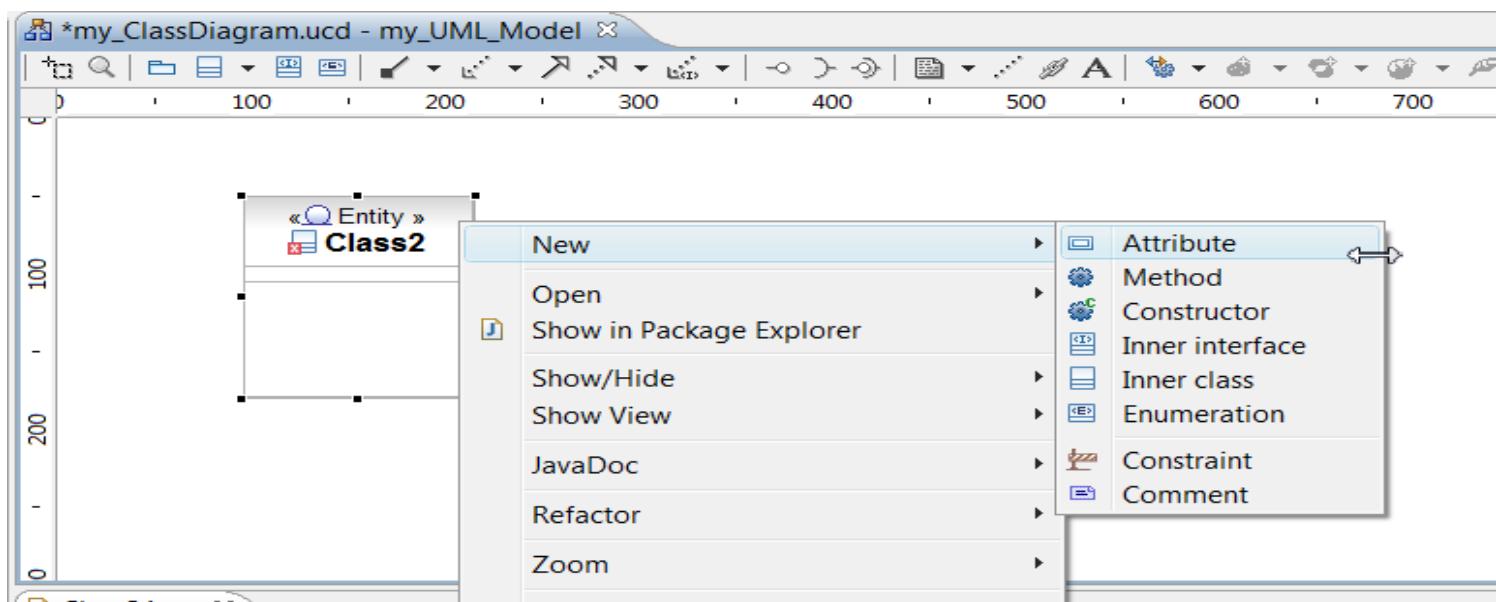
We are going to fix this error by selecting persistence.xml file in the **Package Explorer > JPA Tools > Synchronize**



There is still an error saying that the Entity " Class2" has no Id or EmbeddedId.
We are going to fix this error.



We need to create a new attribute Id.
Click on the **Class2 > New > Attribute**



The attribute has been created. Click on the attribute and select the attribute code (*the JPA Details is synchronize with the java code and the diagram is also synchronize with the java code. Don't close the java code, because there is no direct synchronization between the UML editor and the JPA details, except using the code editor...*)

Add a map property to the Attribute.

The screenshot shows the UML editor and Java code editor side-by-side, demonstrating the synchronization of JPA attribute mapping.

UML Editor: The top left window displays a ClassDiagram.ucd file. A class named "Class2" is shown with an attribute named "attribute". Below the class, two methods are listed: "getAttribute()" and "setAttribute()".

Java Editor: The bottom left window displays a Class2.java file. The code defines an Entity named "Class2" with an attribute "attribute" annotated with @Id.

JPA Details Editor: The rightmost window is titled "JPA Details". It shows the configuration for the "attribute" attribute. Under "Map As:", "Id" is selected. Other options include "Column", "Name", "Table", "Insertable", and "Updatable", each with their respective settings. Below this, sections for "Primary Key Generation" and "Table Generator" are visible.

The interface indicates that changes made in one editor (e.g., adding the @Id annotation) are reflected in the other, maintaining consistency between the UML model and the generated Java code.

The code is now clean with no error.

The screenshot shows a UML editor interface with two main panes. The top pane displays a Class diagram titled 'Class2' which is annotated as an Entity. It contains one attribute and two operations: 'getAttribute()' and 'setAttribute()'. The bottom pane shows the generated Java code for 'Class2.java'.

```
package my_UML_Model;

import javax.persistence.Entity;
import javax.persistence.Id;

/**
 */
@Entity
public class Class2 {

    /*
     * (non-Javadoc)
     */
    @Id
    private String attribute;
```

We are now going to use the Add JPA annotation menu.

Please note that the JPA Annotation sub menus will only be activated if the Class is already an Entity.

Diagram View (Top Left): A UML Class Diagram showing a class named "Class2". The class has an attribute "attribute" and two methods: "getAttribute" and "setAttribute". The class is annotated with "«Entity»".

Code View (Bottom Left): An editor showing the generated Java code for "Class2.java". The code includes Javadoc comments for the attribute and methods, and annotations like @Entity and @Id.

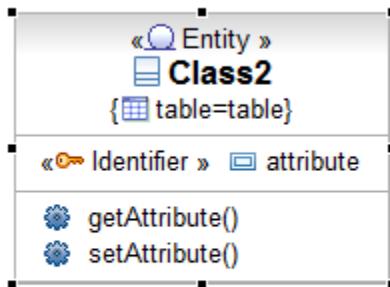
Context Menu (Right Side): A context menu is open over the "Class2" element. The menu items include:

- New
- Open
- Show in Package Explorer
- Show/Hide
- Show View
- JavaDoc
- Refactor
- Zoom
- Sorting
- Package Information
- View selector...
- Dependencies
- Inheritance
- Associations
- Resize Shape
- Undo Creating element
- Redo
- Delete from Diagram
- Delete from Model and Source
- Links
- Preferences
- Add JPA Annotation
- Properties

Sub-menu for "Add JPA Annotation" (highlighted):

- Discriminator
- Entity
- inheritance
- PK_JoinColumn
- Secondary Table
- catalog...
- idClass...
- schema...
- table...

The JPA table annotation has been added in your UML Editor diagram, in your Java Code and in your UML Model.



5. Why my new JPA Class is in error?

The JPA Class could be in error for one of the following reasons:

- Did you add the needed external jar. [See more information](#)
- Did you synchronize your data. [See more information](#)
- Did you add an Id to your Class. [See more information](#)

Note that if you select Discover annotated classes automatically at the first time project creation you will not have to synchronize your persistence file each time you create a new class :-).

Add JPA properties

This section is about how to use the JPA menu inside the Class diagram:

1. [Class](#)
2. [Attributes](#)

Don't forget that to activate the JPA properties on class diagram elements you need to:

1. Activate the Persistence JPA Class Diagram contextual menu

Select the Class Diagram background > **Persistence Development > JPA**

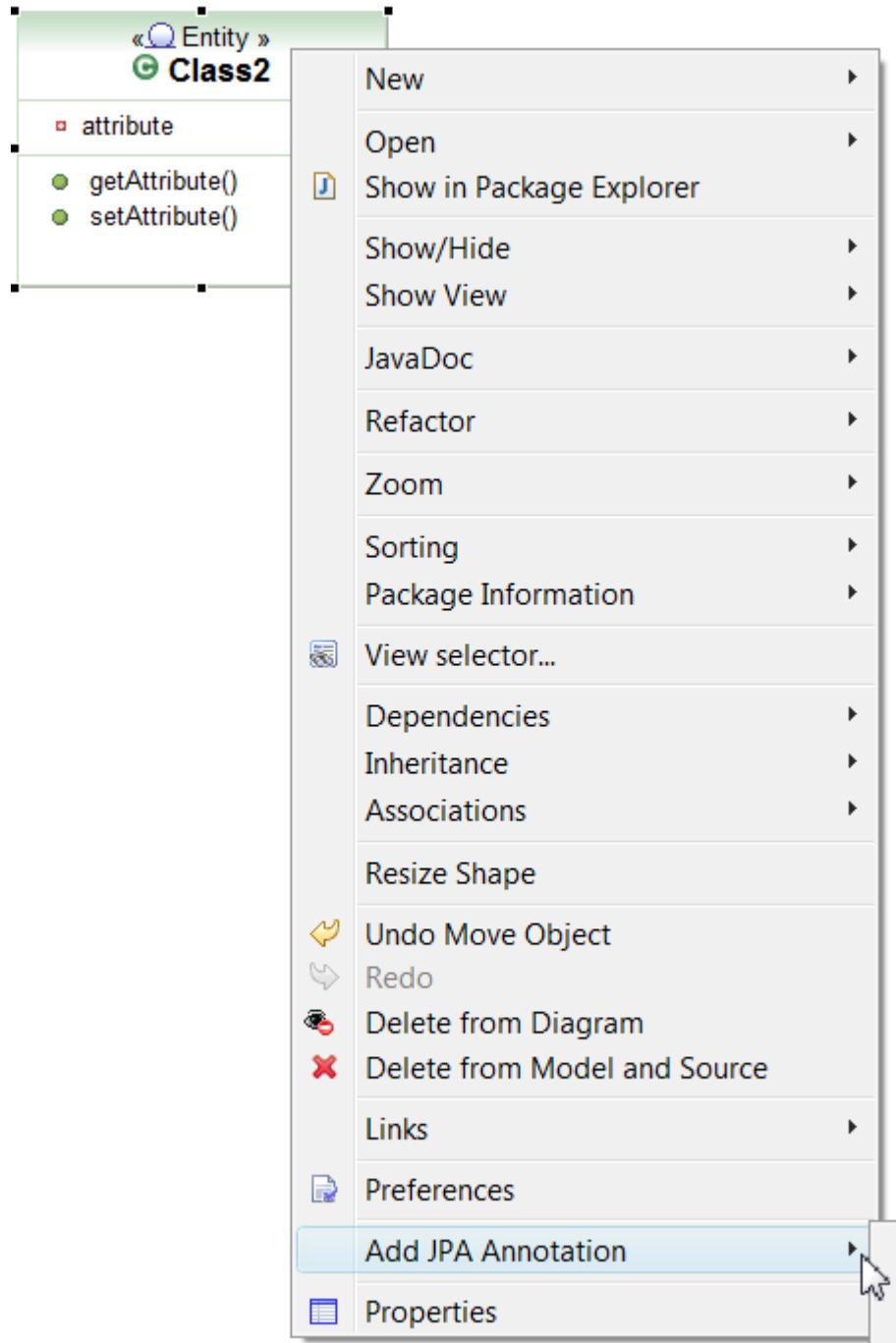
2. Add Entity stereotype on the UML Class or in the Java Code @Entity

Select the **Class > Properties > New Stereotype > Persistence::Entity** to add entity stereotype using the UML Editor

Select the **Class code in the java editor and type @Entity** to add Entity Stereotype using the Java Editor

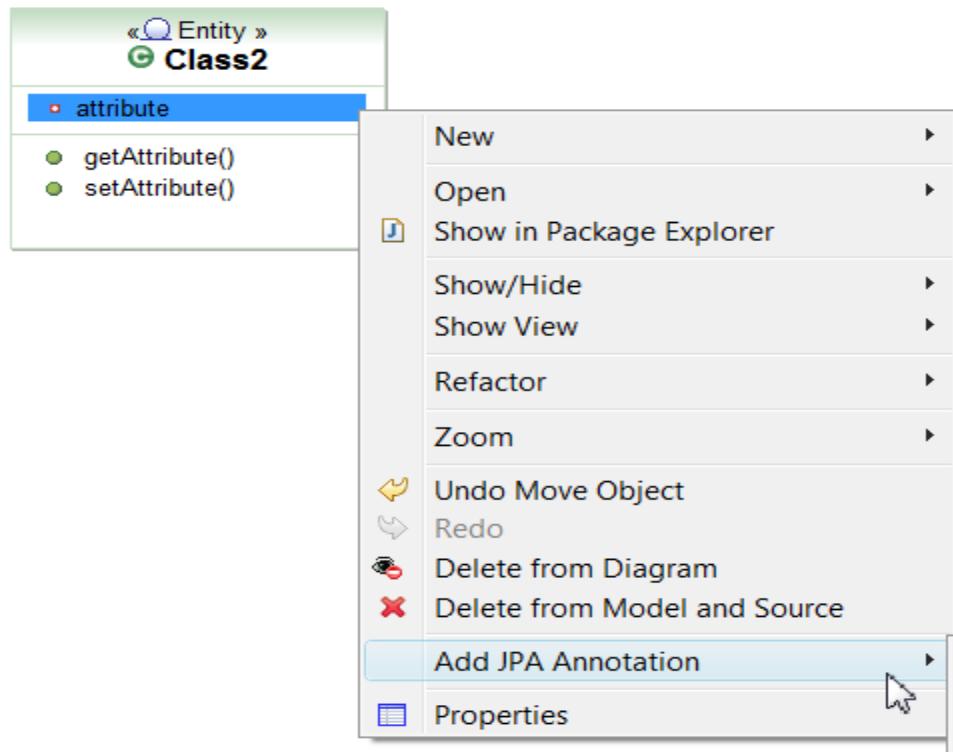
1. Add JPA Properties on a Class using the UML Editor

Click on a **Class** > **Add JPA Annotation**



2. Add JPA Properties on an Attribute

Click on an Attribute inside the **Class** > **Add JPA Annotation**

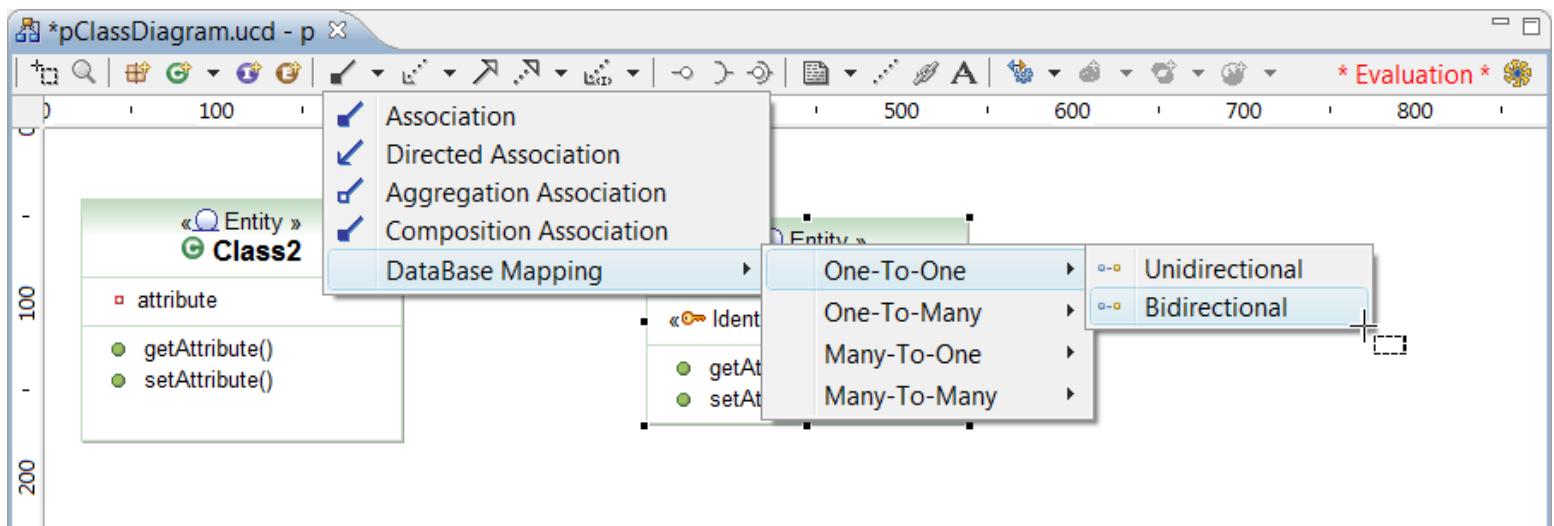


Create Associations between Classes

EclipseUML allows creating the following mapping associations:

1. [One to One](#)
2. [One to Many](#)
3. [Many to One](#)
4. [Many to Many](#)

The Database Mapping menu is available in the Class diagram Association top down menu
Database Mapping >....
Click on the arrow to open the top down menu.



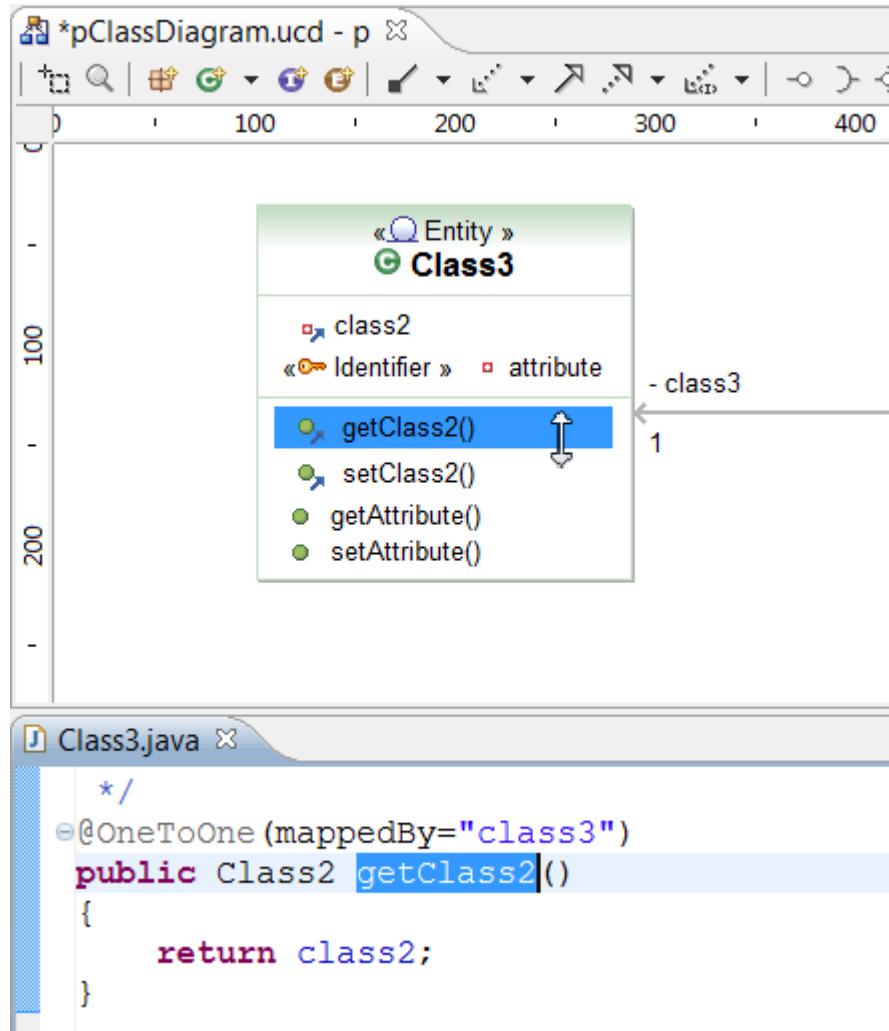
1. One to One Mapping

Click on the **association arrow in the toolbar > Database Mapping > One to One > Bidirectional**

Then move your mouse on the first class and click.

Finally move the link to the second class and click.

The annotation has been added on the getter and the UML editor has been updated.



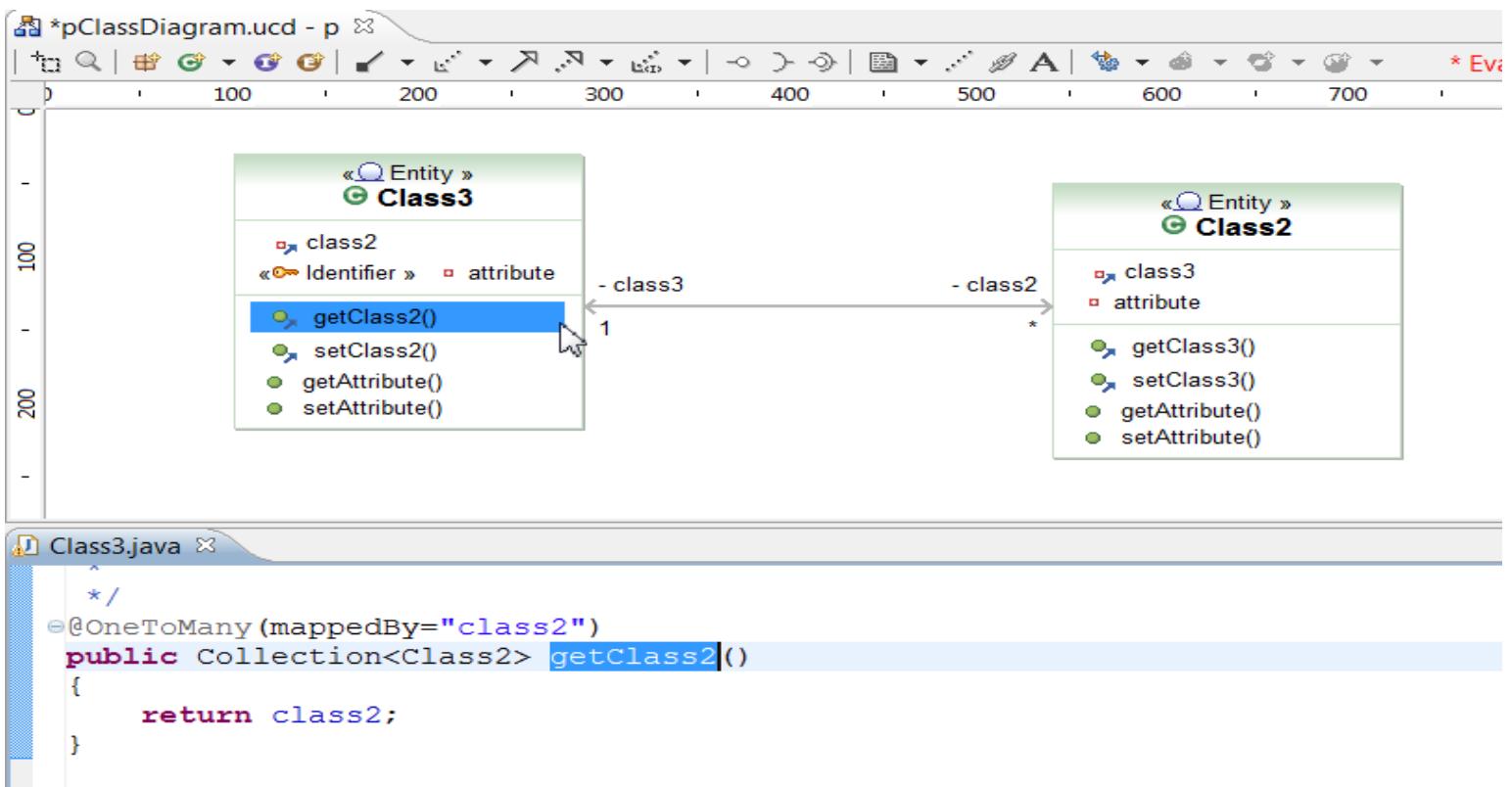
2. One to Many Mapping

Click on the **association arrow in the toolbar > Database Mapping > One to Many > Bidirectional**

Then move your mouse on the first class and click.

Finally move the link to the second class and click.

The annotation has been added on the getter and the UML editor has been updated.



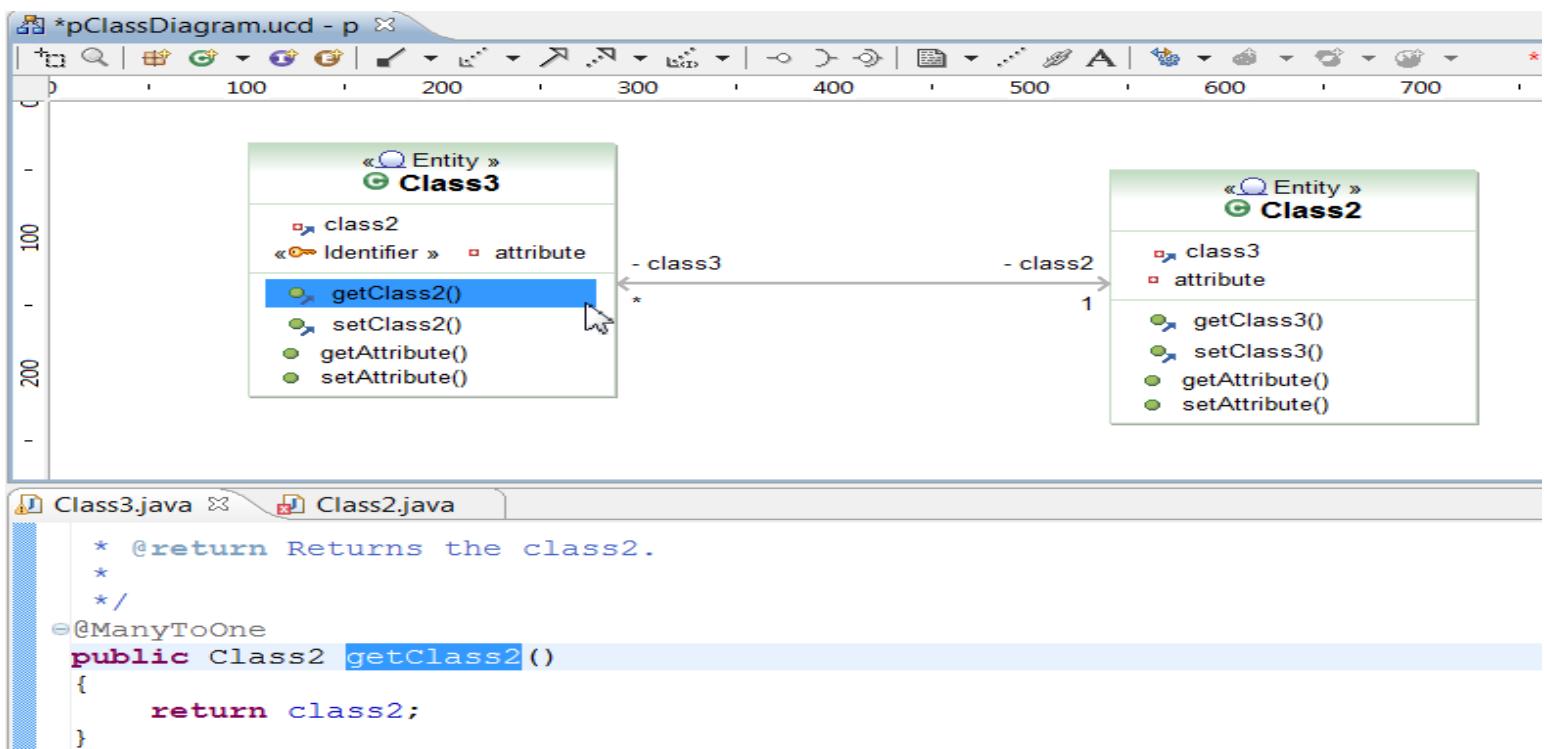
3. Many to One Mapping

Click on the **association arrow** in the toolbar > Database Mapping > Many to One > **Bi-directional**

Then move your mouse on the first class and click.

Finally move the link to the second class and click.

The annotation has been added on the getter and the UML editor has been updated.



4. Many to Many Mapping

Click on the **association arrow in the toolbar > Database Mapping > Many to Many > Bidirectional**

Then move your mouse on the first class and click.
Finally move the link to the second class and click.

The annotation has been added on the getter and the UML editor has been updated.

The screenshot shows the Eclipse UML editor interface. At the top is the toolbar with various icons. Below it is the UML Class Diagram workspace. Two classes are shown: 'Class3' on the left and 'Class2' on the right. A bidirectional association line connects them, labeled '- class3' on the Class3 side and '- class2' on the Class2 side. Both ends of the association have multiplicity '*' indicated below the line. The 'Class3' class has an attribute 'class2' and a method 'getClass2()'. The 'Class2' class has an attribute 'class3' and a method 'getClass3()'. In the code editor at the bottom, the Java file 'Class3.java' contains the following code:

```
*  
*/  
@ManyToMany  
public Collection<Class2> getClass2()  
{  
    return class2;  
}
```

Please note EclipseUML will only create annotation on getters.

It a different choice than Dali using attributes, but we think this is a better way of coding.

Reverse an existing Database into the Class Diagram

To reverse an existing Database into the Class diagram you need to use both Dali and EclipseUML plugins.

It is not possible to reverse an existing database if you don't use the EclipseUML JEE build including Eclipse 3.4 Ganymede.

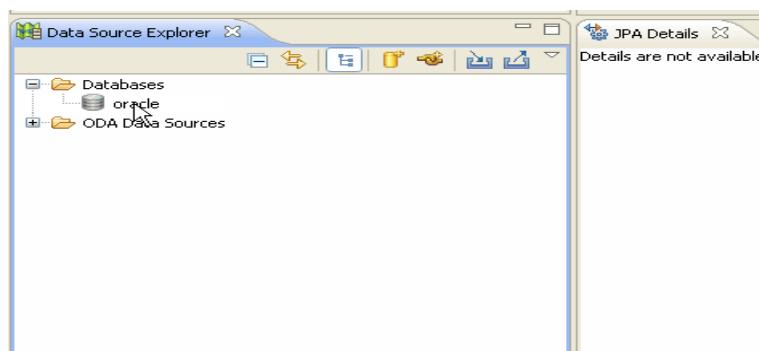
This section is composed by an Oracle Database example:

1. [Connect your database](#)
2. [Create a JPA Project](#)
3. [Generate Java code from Database](#)
4. [Reverse Oracle Database into a class Diagram](#)

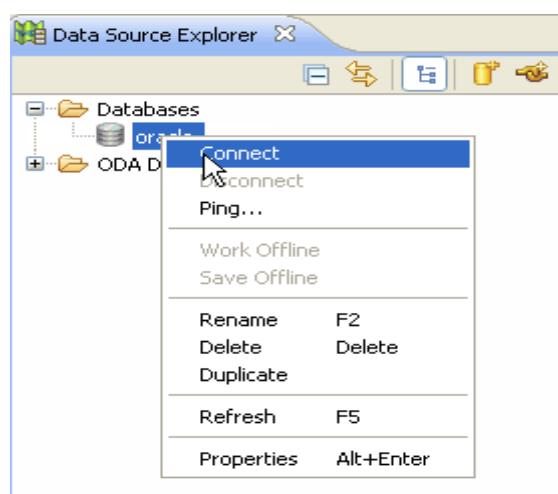
1. How to connect to my Database.

You need to switch to JPA Perspective.

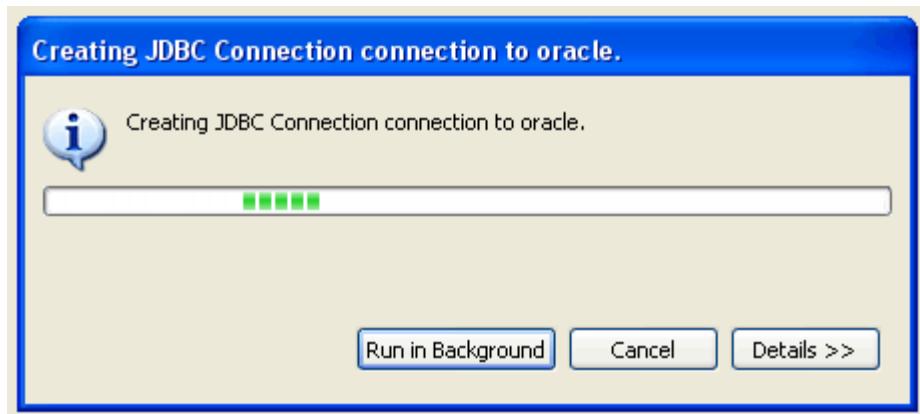
Go to the **Database Source Explorer > Database > Oracle**



Click on Oracle in the Database Explorer > **Open the contextual menu > Connect**

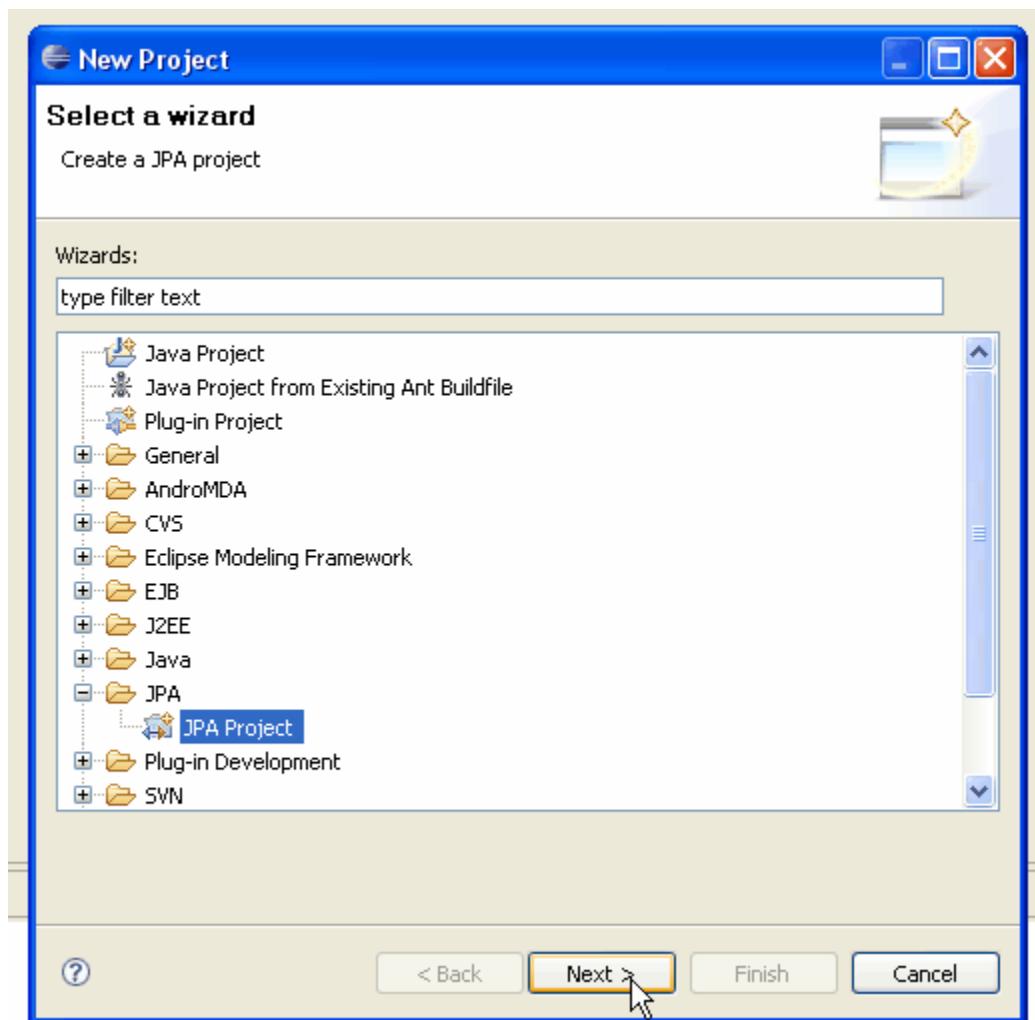


The JDBC connection will be created

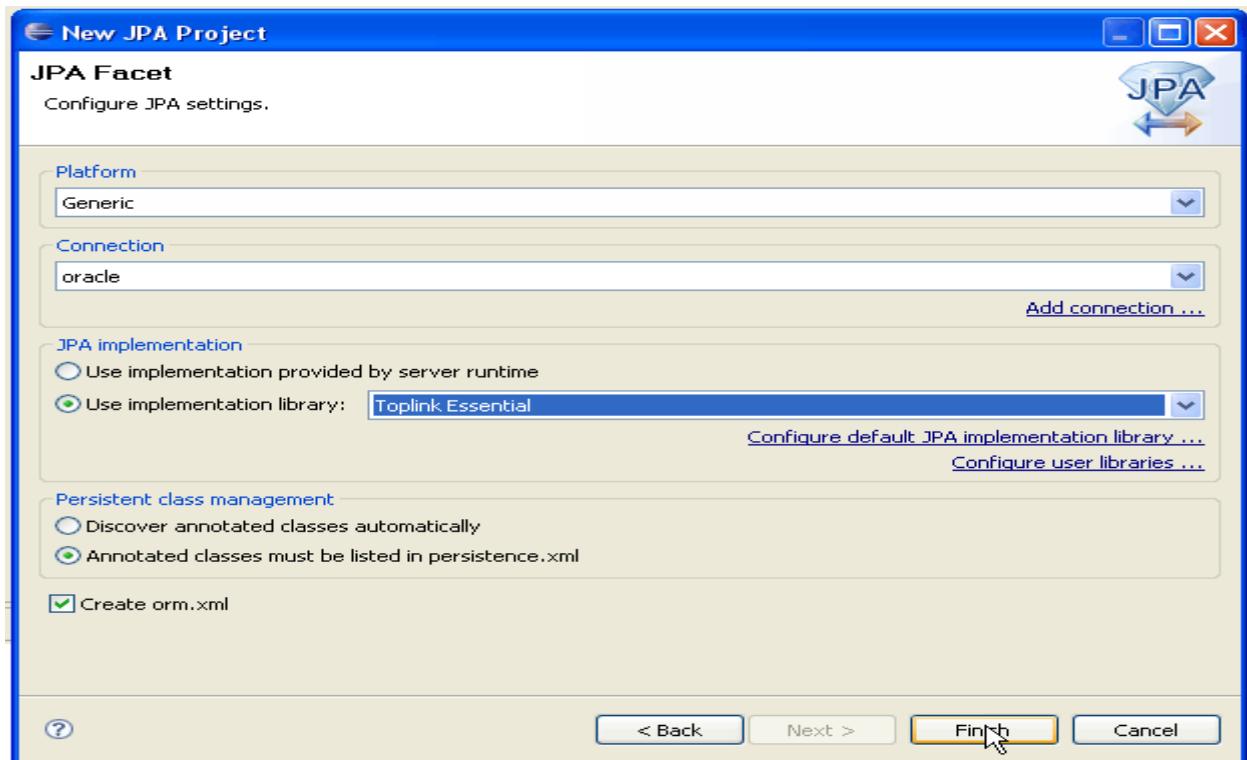


2. Create a JPA Project

Click in the **Package Explorer** > **New** > **Other** > **JPA** > **JPA Development** > **Next**

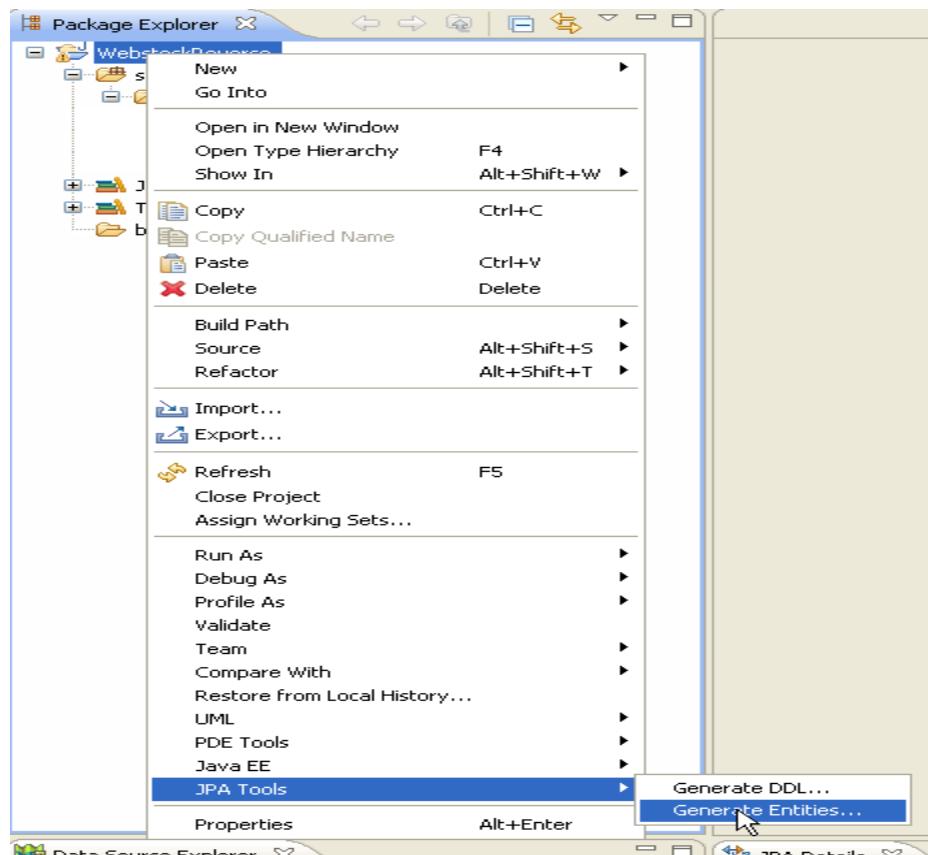


Select your Connection (e.g. Oracle) and your implementation library (e.g. Toplink)



3. Generate Java Code from your database

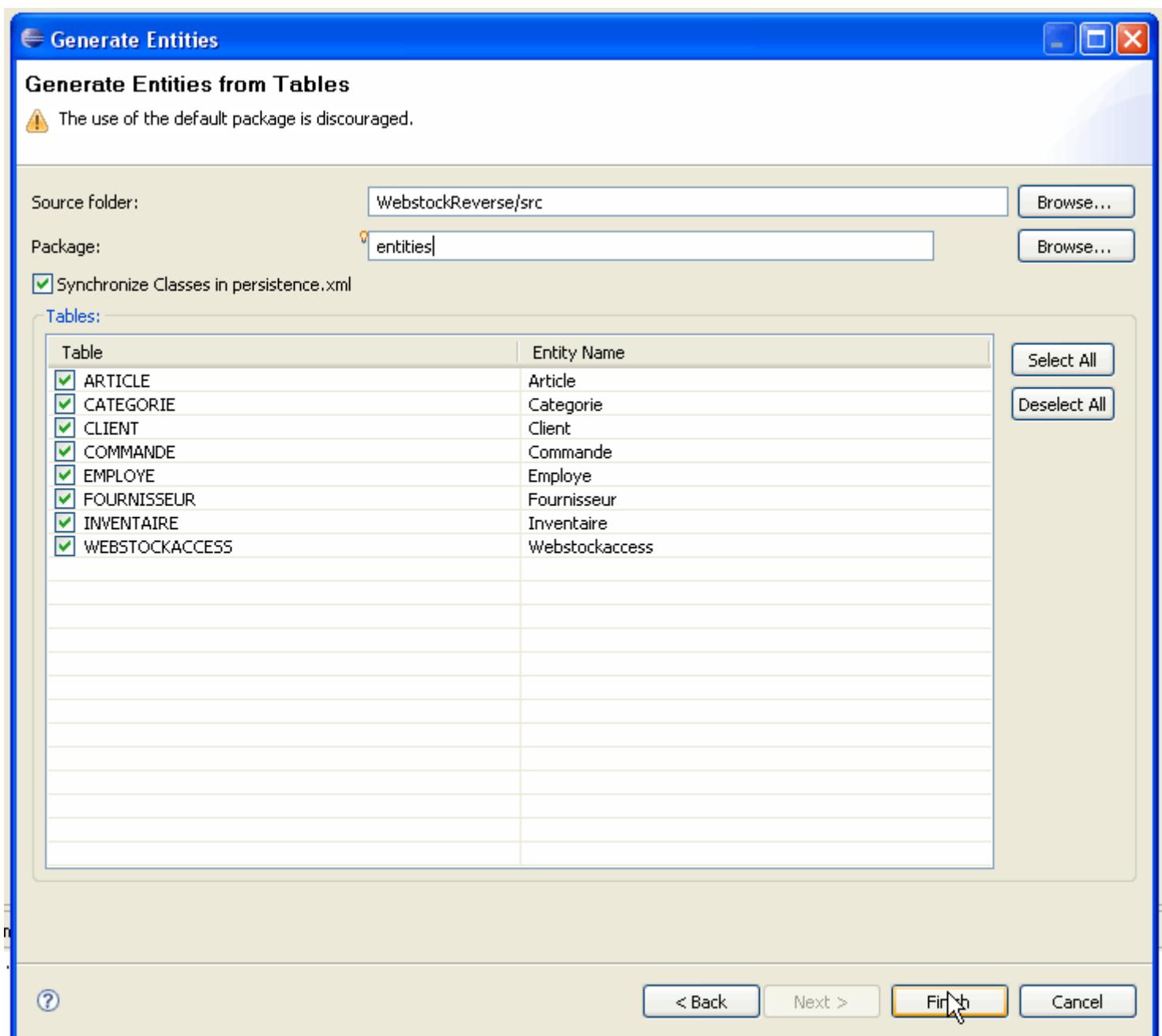
Click on your JPA project in your **Package Explorer** > **JPA tools** > **Generate Entities**



Select the source folder and the table you want to transform into Java Classes (e.g. Article, Categorize etc...)

Create a package in which your code will be generated (e.g. entities)

Click on the finish button.



Your previously selected tables have been created as Entity Classes in your Package Explorer in the package entities.

The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer view displays a Java project named 'WebstockReverse'. The 'src' folder contains a package named 'entities' which includes several Java files: Article.java, Categorie.java, Client.java, Commande.java, Employe.java, Fournisseur.java, Inventaire.java, and Webstockaccess.java. It also contains a 'META-INF' folder with MANIFEST.MF and orm.xml files, and a 'persistence.xml' file. Other project components like JRE System Library and Toplink Essential are also listed. On the right, the Article.java editor shows the code for the Article entity class. The code includes imports for Serializable and Entity annotations, defines fields for articleId, description, imageUrl, nomarticle, and poids, and uses @ManyToOne and @JoinColumn annotations to map to Categorie and Fournisseur entities. It also includes a @OneToMany annotation for inventaireCollection and a constructor.

```
package entities;

import java.io.Serializable;

@Entity
public class Article implements Serializable {
    @Id
    private String articleid;

    private String description;

    @Column(name="IMAGE_URL")
    private String imageUrl;

    private String nomarticle;

    private BigDecimal poids;

    @ManyToOne
    @JoinColumn(name="ARTICLECATEGORIEID")
    private Categorie articlecategorieid;

    @ManyToOne
    @JoinColumn(name="FOURNISSEURID")
    private Fournisseur fournisseurid;

    @OneToMany(mappedBy="articleid")
    private Set<Inventaire> inventaireCollection;

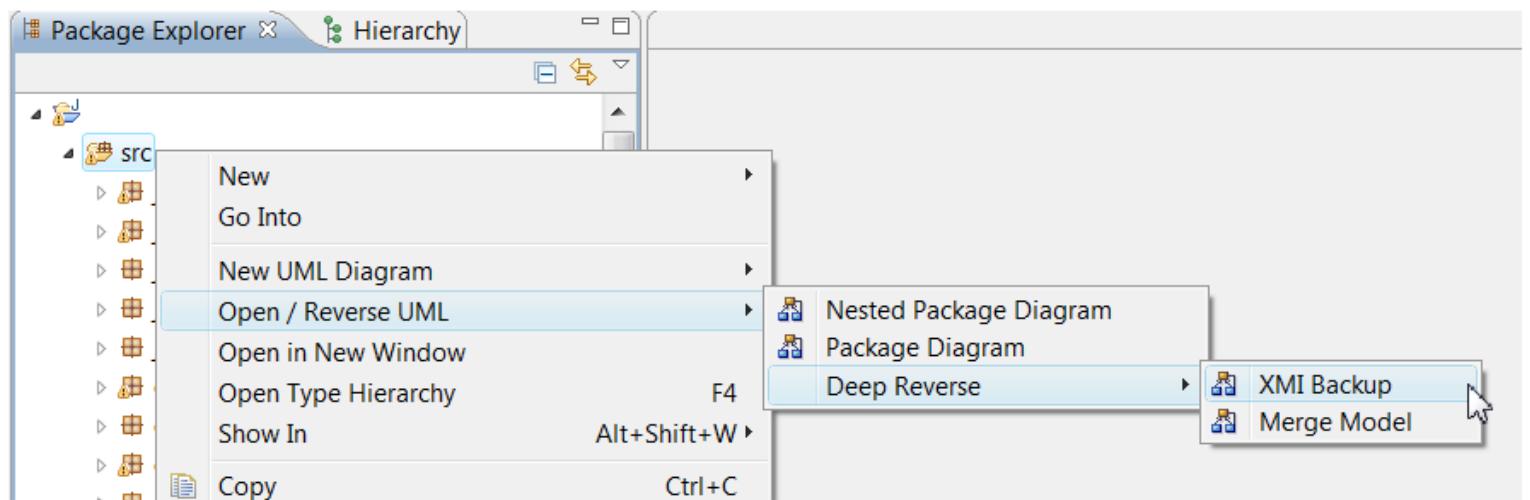
    private static final long serialVersionUID = 1L;

    public Article() {
        super();
    }
}
```

4. Reverse Oracle Database into a UML 2.2 model

For a deep reverse click on the src of your JPA project in your **Package Explorer > Open /Reverse UML > Deep Reverse > Xmi Backup**.

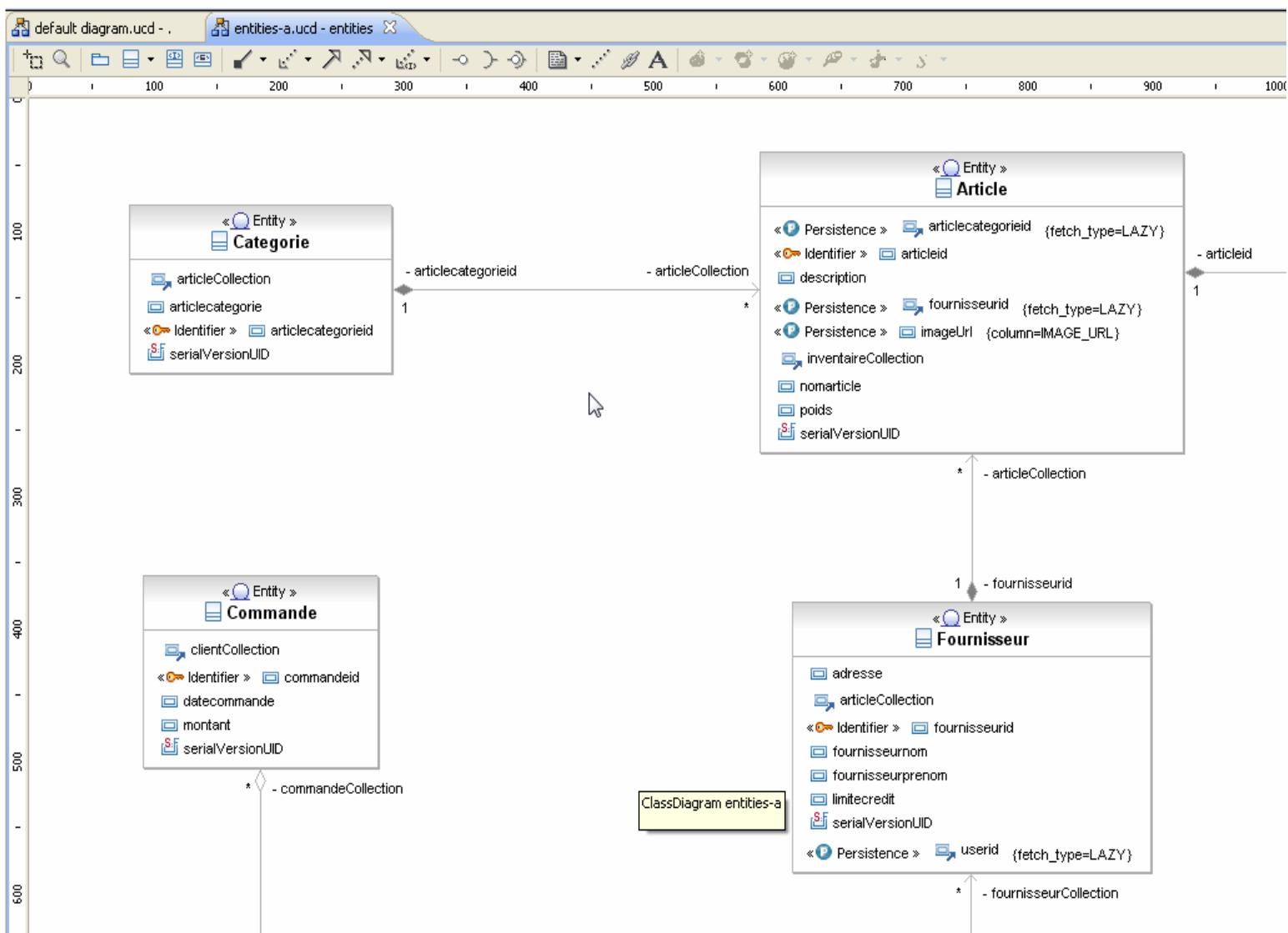
This will create a new UML 2.2 Metamodel from your Java Code Entities.



Your project has been updated to UML 2.2 metamodel.

Click on your project and select **src > entities > Open Reverse UML > Class Diagram > select association and click on the OK button.**

Here is the class diagram which will be displayed inside Eclipse 3.4 in the JPA Perspective.

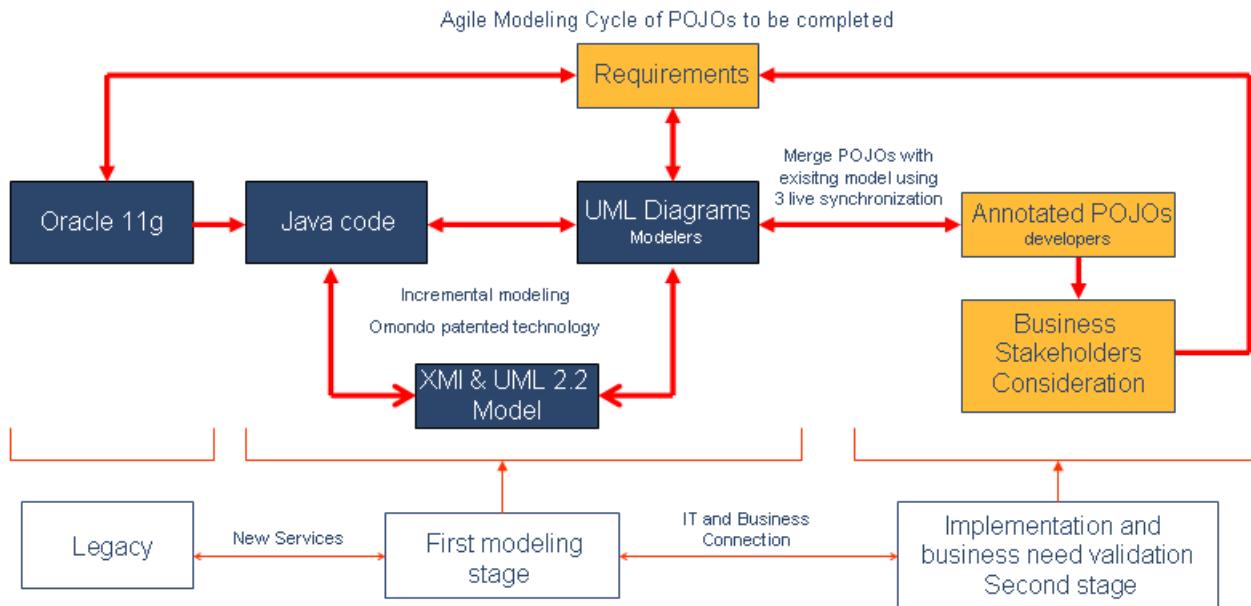


Use Oracle® Enterprise Pack Eclipse 11g with EclipseUML

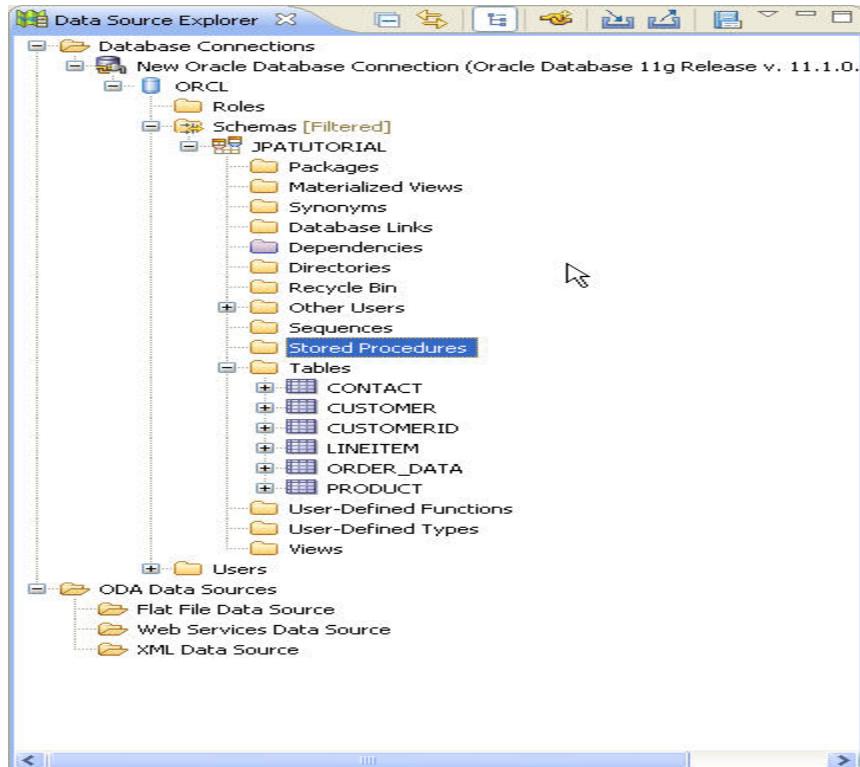
EclipseUML could be directly activated inside Oracle® Enterprise Pack Eclipse 11g for **Eclipse 3.4 Ganymede**.

The EclipseUML integration is available using the live code and model synchronization.

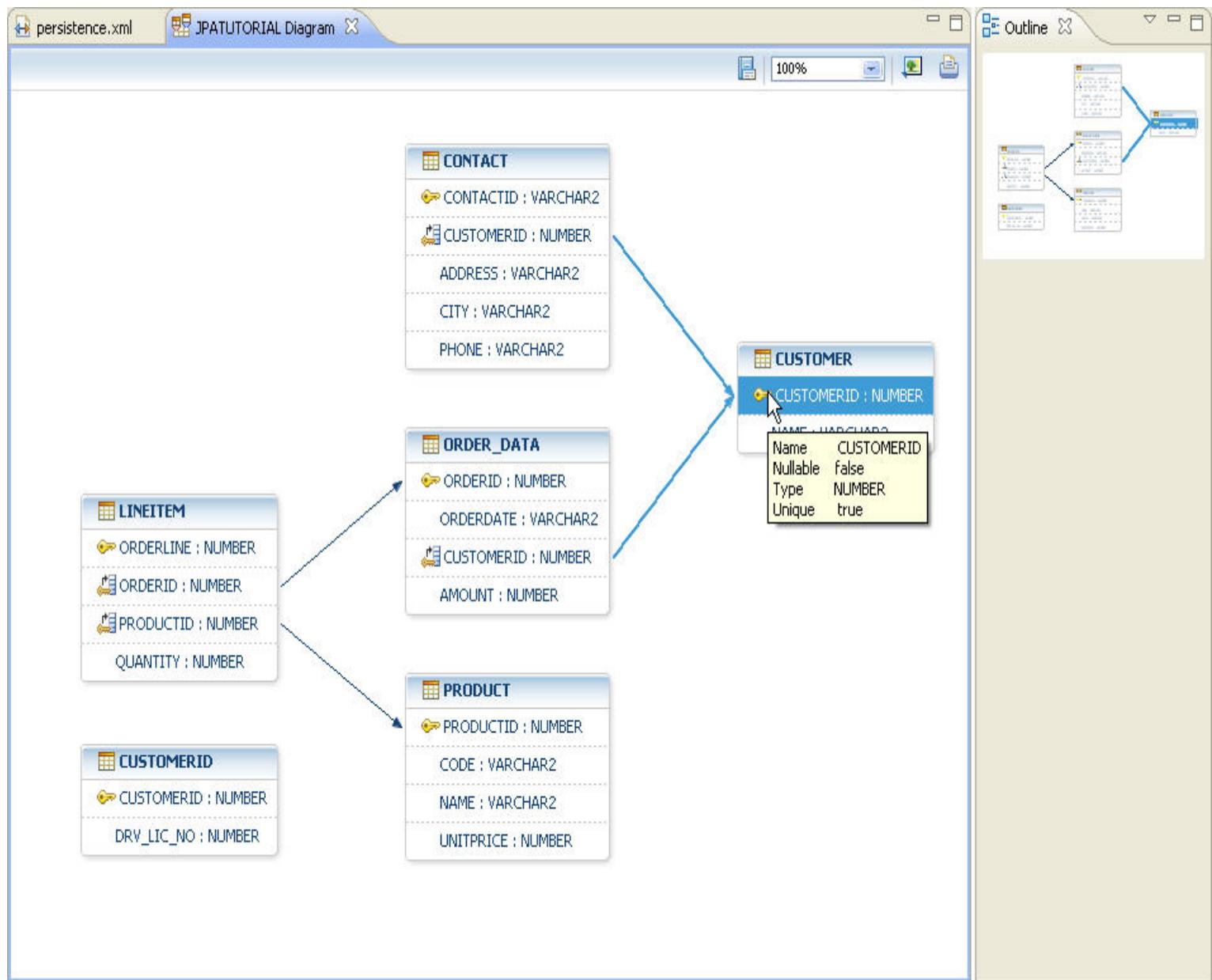
The picture below shows how EclipseUML and the Oracle® Enterprise Pack Eclipse 11g can cover the full POJOs modeling cycle:



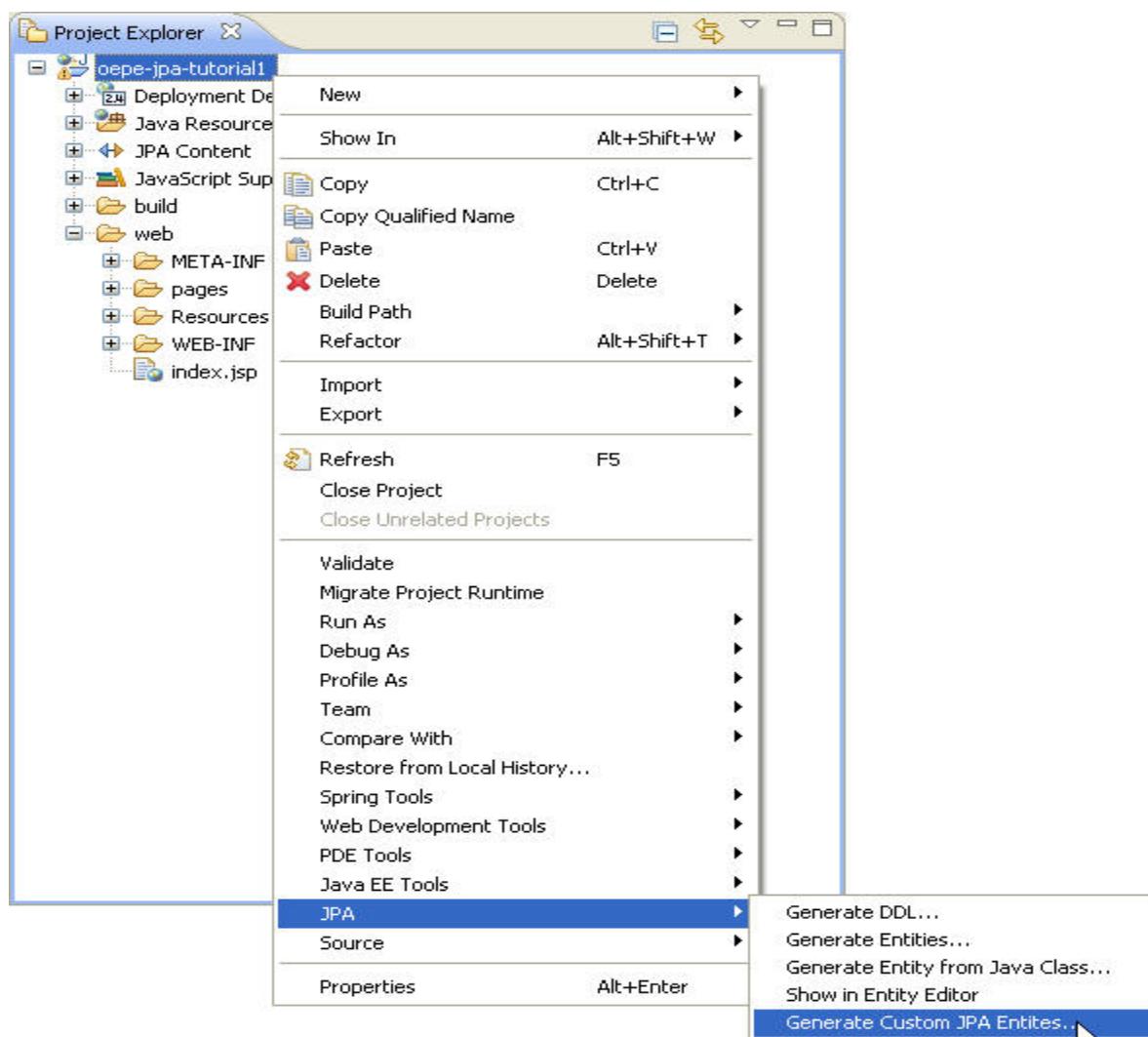
Using the Spring, ORM Oracle® features allow you to navigate in the Data Source Explorer



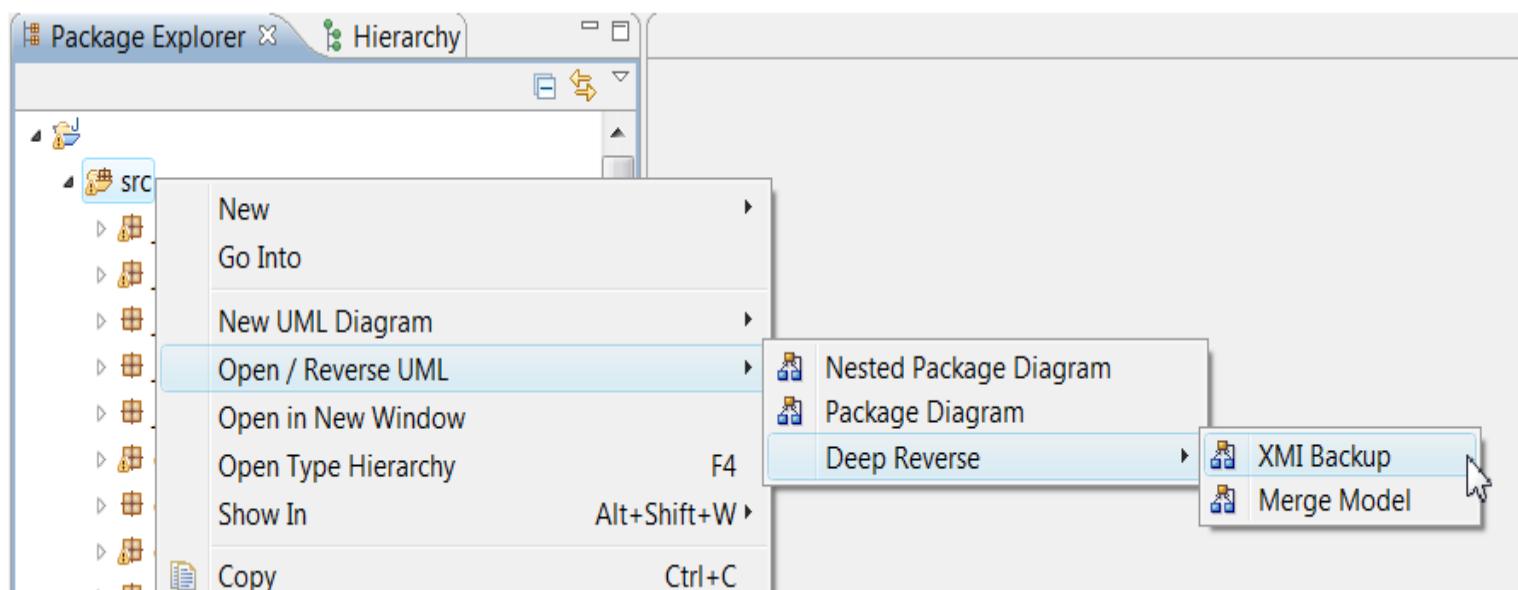
and show a schema View



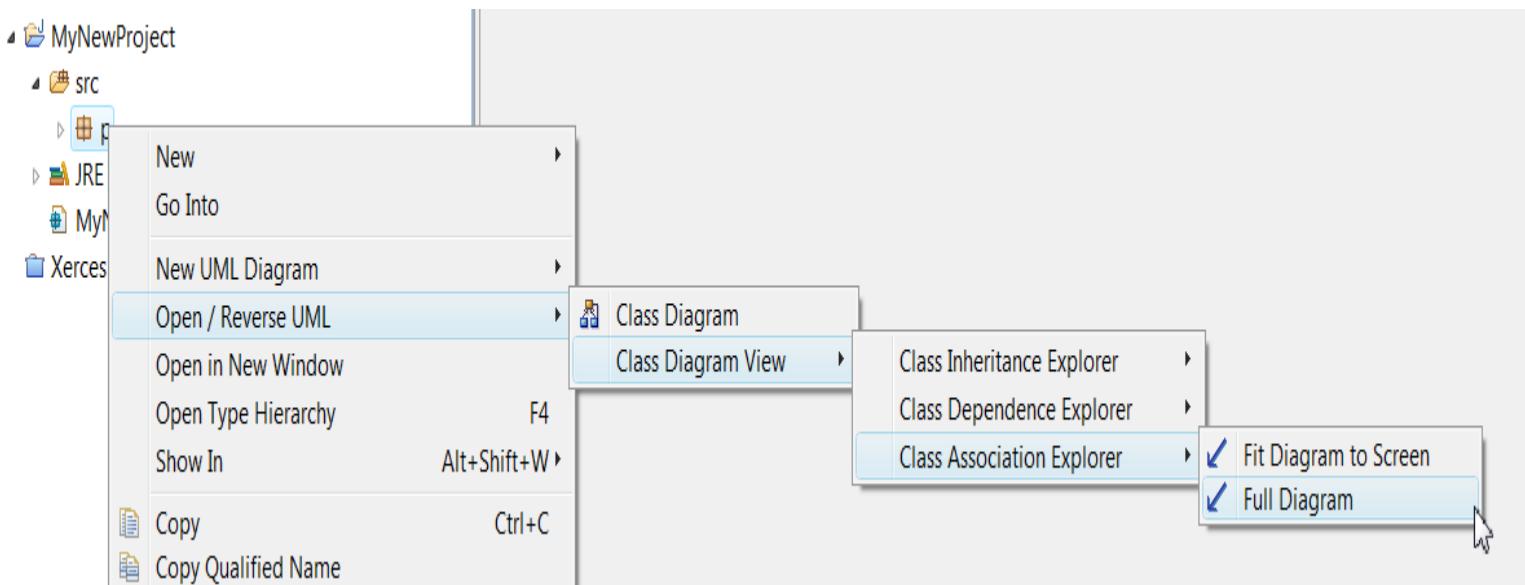
And finally generate Annotated POJOs from a schema



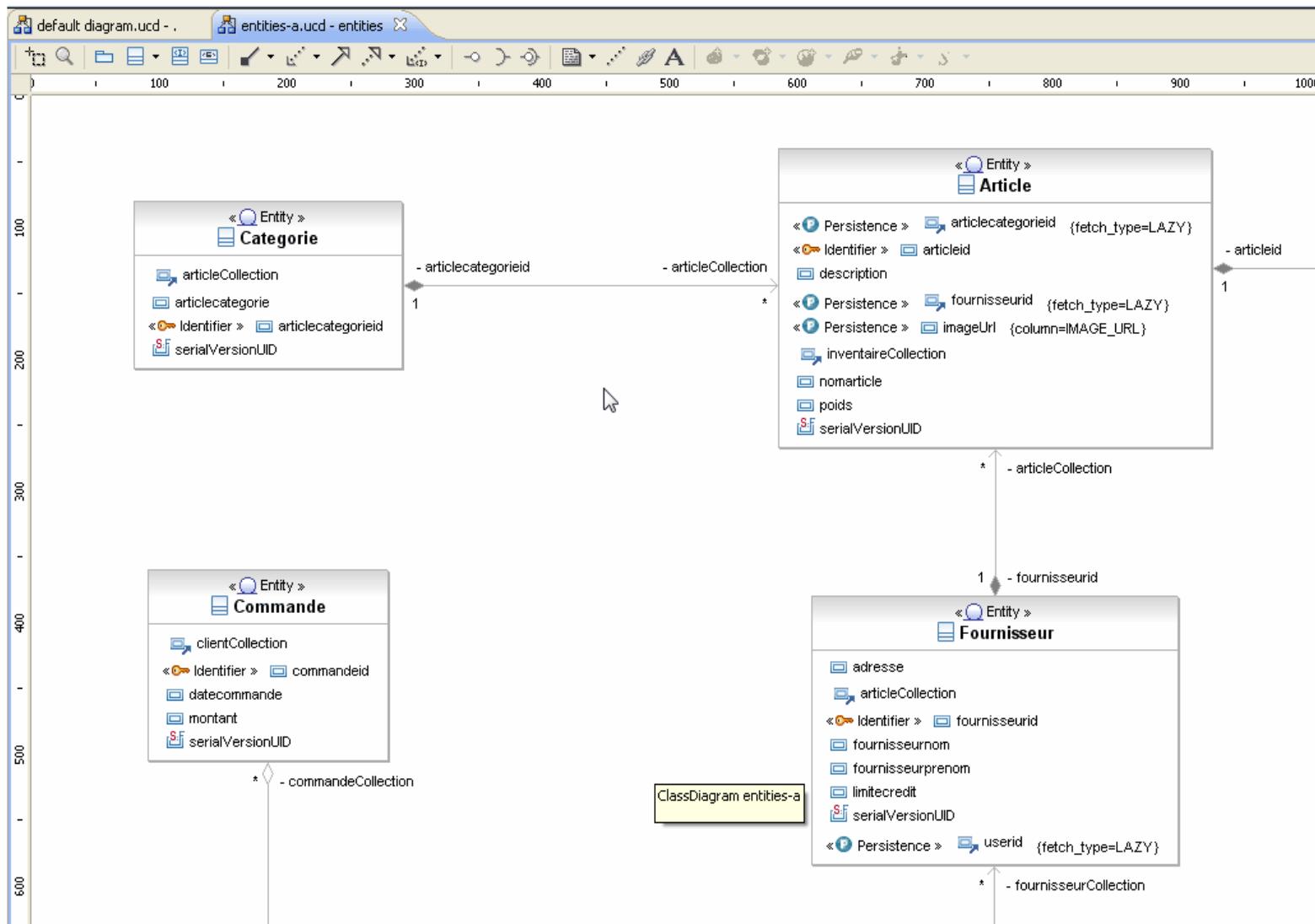
Once the Java code has been generated click on the src in which your source code has been created and upload your java code (*e.g. code including annotations*) inside your Class diagram. This will map your entire Database to a standard UML 2.2 model.



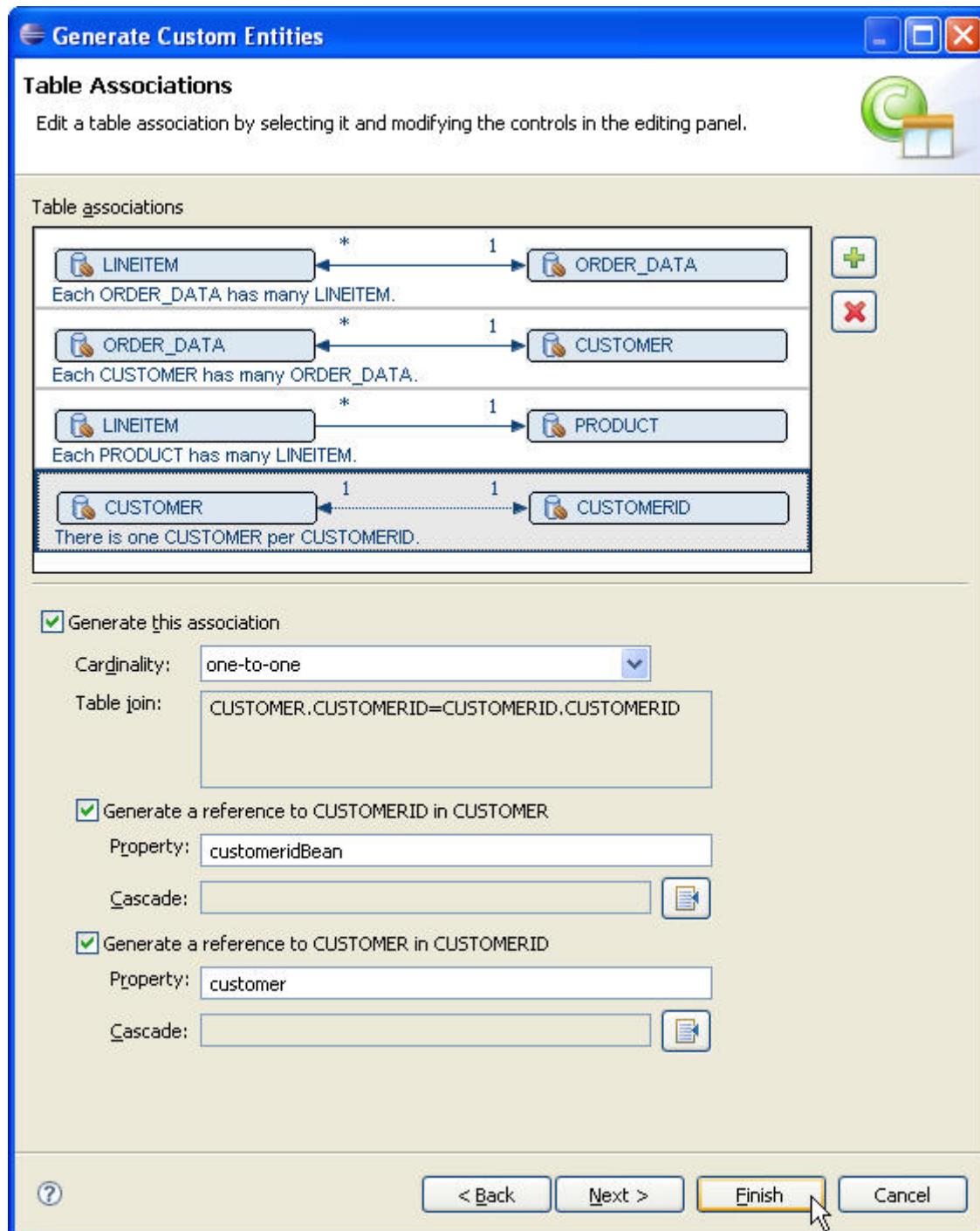
If no src then click on the package which contains the source code



The following diagram is an UML 2.2 compliant Class Diagram example created from an Oracle Database.

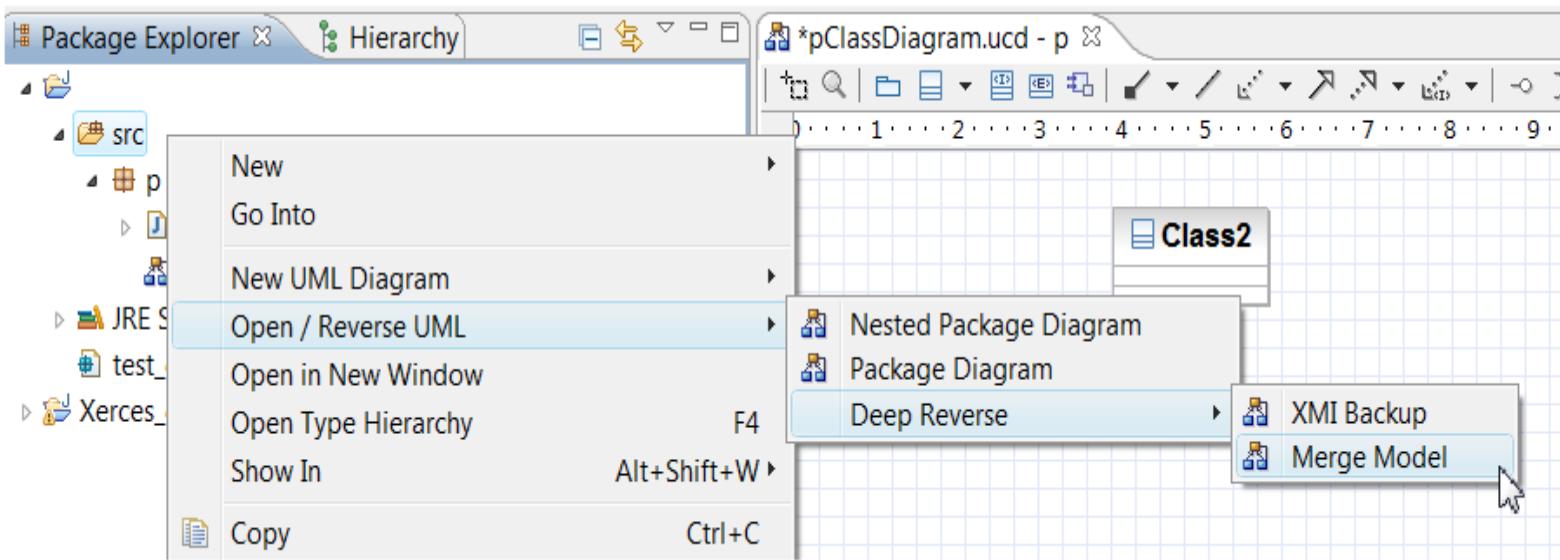


Please note that you can add, change or delete any information using Eclipse & the Enterprise Pack for Eclipse 11g because all your diagrams and model will be automatically updated. You can have 100 000 diagrams using a view of a POJO and update all your diagrams by just changing the table association information :-)



You can also deploy and test your application. After this first stage you can add new requirements and POJOs at object level and never loose Database UML 2.2 modeling compliance using the advanced merge feature. The merge option is an important concept if the java project is evolving with the creation of new UML diagrams. The concept is to reverse the full project at the beginning of the modeling stage, then to create diagrams and close EclipseUML. After few weeks of coding another iteration stage is needed and the model merge will change the refactored classes and add all detected new packages, classifiers and connectors to the existing model.

The merge option is available if you select the **src > Open / Reverse UML > Deep Reverse > Merge Model**.



The merge mechanism is working at XMI level and is respecting the following rules:

- Existing classifiers and connectors are never erased from the model.
- Each new package or classifiers are added to the model.
- Pojos annotations are updated
- Existing Classifiers which doesn't exist anymore in the java code are automatically transform into model classifiers and not deleted from diagrams or from the model.
- Existing classifiers which are refactored are also immediately renamed/moved in the model and in each UML diagram when you first time open the editor.
- If attributes or methods are added or deleted or rename then it is immediately mapped with the model (e.g. *it means that deleted attributes or methods from the java code are deleted from the model at this stage*).

Please note that if you copy and paste diagrams into another project then diagrams will not anymore be synchronized with Java or the model.

If you move diagrams inside your project then diagrams will keep the same properties and will always be synchronized.

TeamWork

This section is composed by:

1. [Multiple models consolidation](#)
2. [TeamWork solutions](#)
 1. [Java Code synchronization](#)
 2. [Diagrams creation with no Java code](#)
3. [Team work example using SVN inside Eclipse](#)

EclipseUML is fully integrated with SVN and CVS Eclipse solutions.

You can also use EclipseUML with any other SCC tool but you will not be able to use the model compare feature before the commit.

It is important when you decide to start a project to select which team mode to use.

You can decide to work in a team by selecting:

- either multiple models approach ([see more information...](#))
- or use immediately a teamWork solution

It is not recommended to:

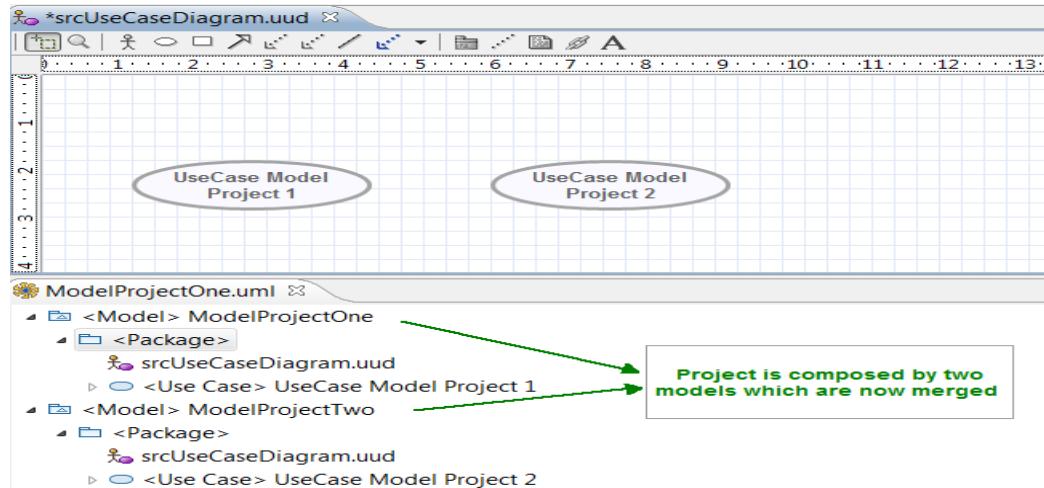
- *start a project using different projects on each desktop, even if the project have the same name and then try to merge these models. This is a dynamic Id creation limitation. This will not work!!*
- *use txt editor to manipulate the .uml model.*

1. Multiple models approach limitations

You can model different pieces of the architecture and then just unify your models at the end of the modeling stage. This is an easy way to work. Please note that the unify model is not incremental. It means that once models have been unified, then you can not merge your single model with the unified model if you change it after.

The only solution is to unify again all models inside another single model. This is a static approach which is used if the modeling project is finished. You can also from this single unify model use team work for the second stage of modeling.

See multiple models inside a larger model in the picture below:



It is not possible to split java code between multiple models because each diagram uses Java Id and create UML Id dynamically.

If you use java code, then your UML Ids will be different even for the same element used inside the same project name which makes no sense.

Java synchronization and multiple models are not therefore possible together.

2. TeamWork solutions

You first need to create a single project on SVN or CVS.

Then to ask each member of your team to use this server and commit any change they make.

Finally you need to define each member commit permissions at the server level and not to try to do this administration in the EclipseUML tool.

When working in a team you should consider two options with EclipseUML:

- use eclipseuml live java code synchronization
- do not use java code synchronization

2.1 Java Code synchronization

If you use this live code and model option with your class or sequence diagrams then after each change you need to commit the following files:

- java code
- graphical diagram (e.g. .ucd, .usd, .uud etc..file)
- model file (e.g. .uml file)

2.2 Diagrams creation with no Java code

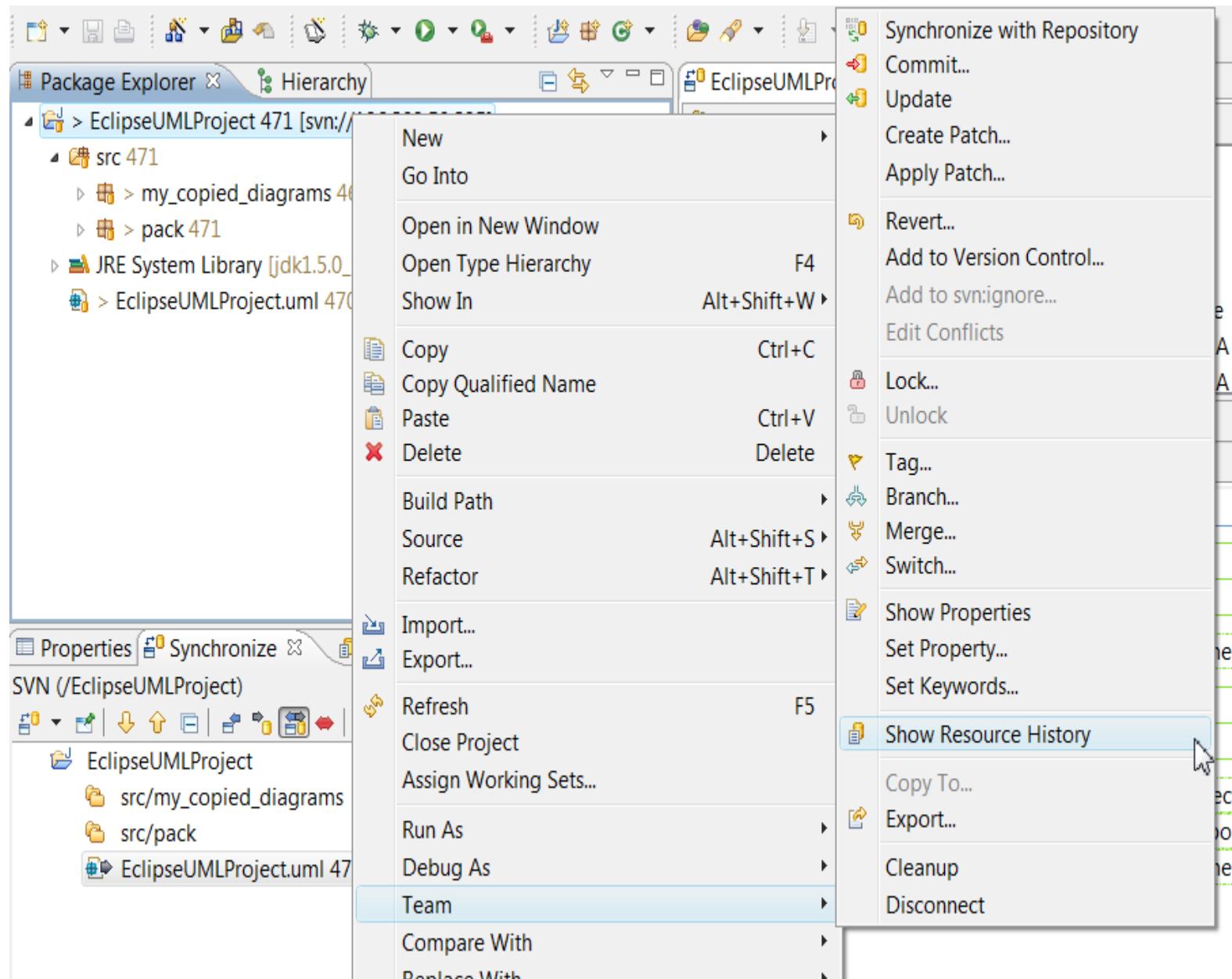
If you don't use the live code and model synchronization then after each change you need to commit the following files:

- graphical diagram (e.g. .ucd, .usd, .uud etc..file)
- model file (e.g. .uml file)

3. Team work example using SVN inside Eclipse

This is an example of using EclipseUML with SVN and Eclipse.

If you want for example to know what is the stage of your code compare with other commits.
Click on the Project inside the Package Explorer > **Team > Show Resource History**.



We see that Vladimir Varnica was the last to commit. You can now decide to update our code and merge it inside our desktop.

Click in the SVN History view and select the **latest commit > Update To**.

Svn Resource History for (EclipseUMLProject)

Revision	Date	Ch...	Author	Comment
472	Nov 14, 2008 1:42:20 PM	1	vladimir.varnica	new usecase diagram
*471	Nov 14, 2008 12:58:12 ...	1	vladimir.varnica	usecase diagram modification
470	Nov 14, 2008 12:57:51 PM	1	vladimir.varnica	change model
469	Nov 14, 2008 12:28:58 PM	1	vladimir.varnica	update UML Model after usecase change
468	Nov 14, 2008 12:28:18 PM	1	vladimir.varnica	change usecase diagram
467	Nov 14, 2008 12:15:21 PM	1	vladimir.varnica	bug SVN disant Malformed network - commit class diag
466	Nov 14, 2008 12:14:28 PM	1	vladimir.varnica	bug commit dit non committé mais en réalité commité
465	Nov 14, 2008 12:08:56 PM	1	vladimir.varnica	sequence diagram modification
464	Nov 14, 2008 10:40:17 AM	1	vladimir.varnica	Test commit ajout d'une class java
463	Nov 13, 2008 9:54:57 PM	2	vladimir.varnica	new Component diagram comit
462	Nov 13, 2008 4:53:25 PM	3	bacem.souissi	ajout du use case Bacem
461	Nov 13, 2008 4:44:29 PM	3	bacem.souissi	remove class2
460	Nov 13, 2008 4:16:54 PM	2	vladimir.varnica	new sequence diagram

new usecase diagram

ROOT

Name	Path	Copied Fr...	Copied Fr...
UseCase.uud	EclipseUMLProj...		

If you want to commit our new usecase which has been added directly in the XMI using the [XMI Editor](#) on the server in order to make it available for the other team members.

1. You can decide to either select all the project > **Team > Synchronize with Repository**.

Java - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Window Help

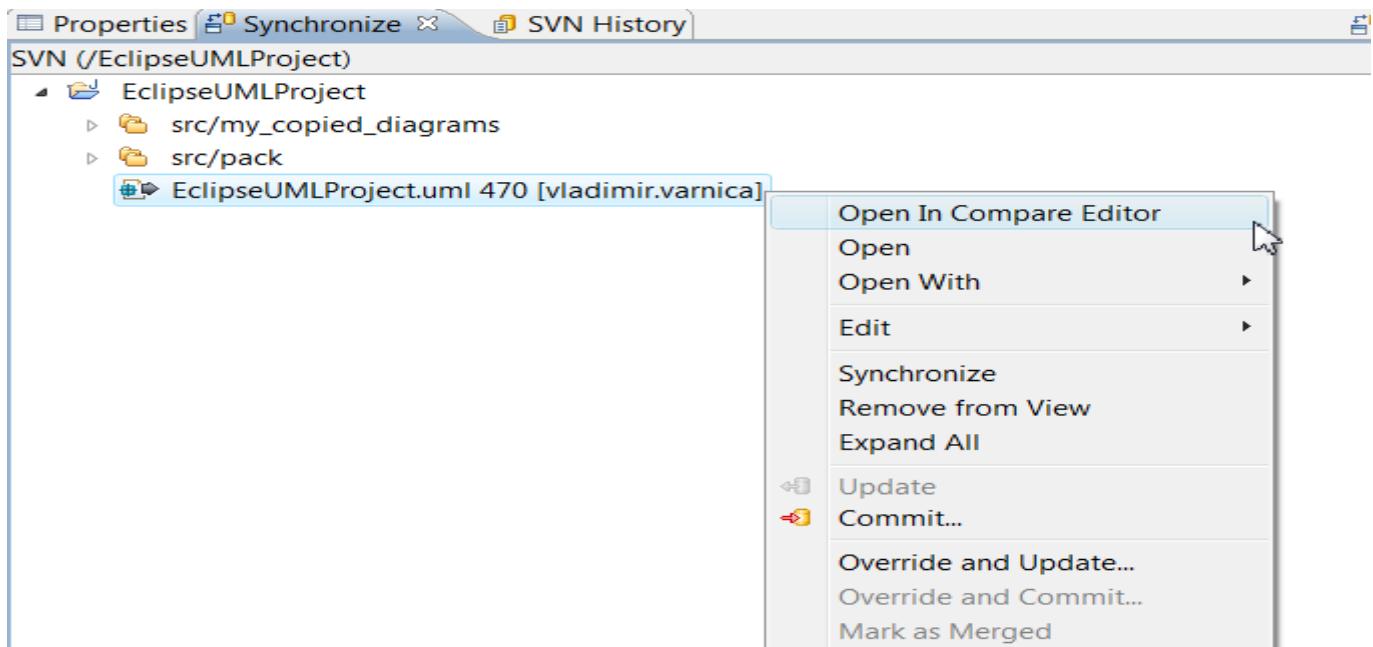
Package Explorer Hierarchy

- > EclipseUMLProject 471 [svn://196...
- src 471
 - > my_copied_diagrams 465
 - Enumeration1.java
 - accountingClasses.ucd

New Go Into Open in New Window Open Type Hierarchy F4

Synchronize with Repository Commit... Update Create Patch... Apply Patch... Revert... Add to Version Control...

Compare the model using the XMI Editor > **Open In Compare Editor**.



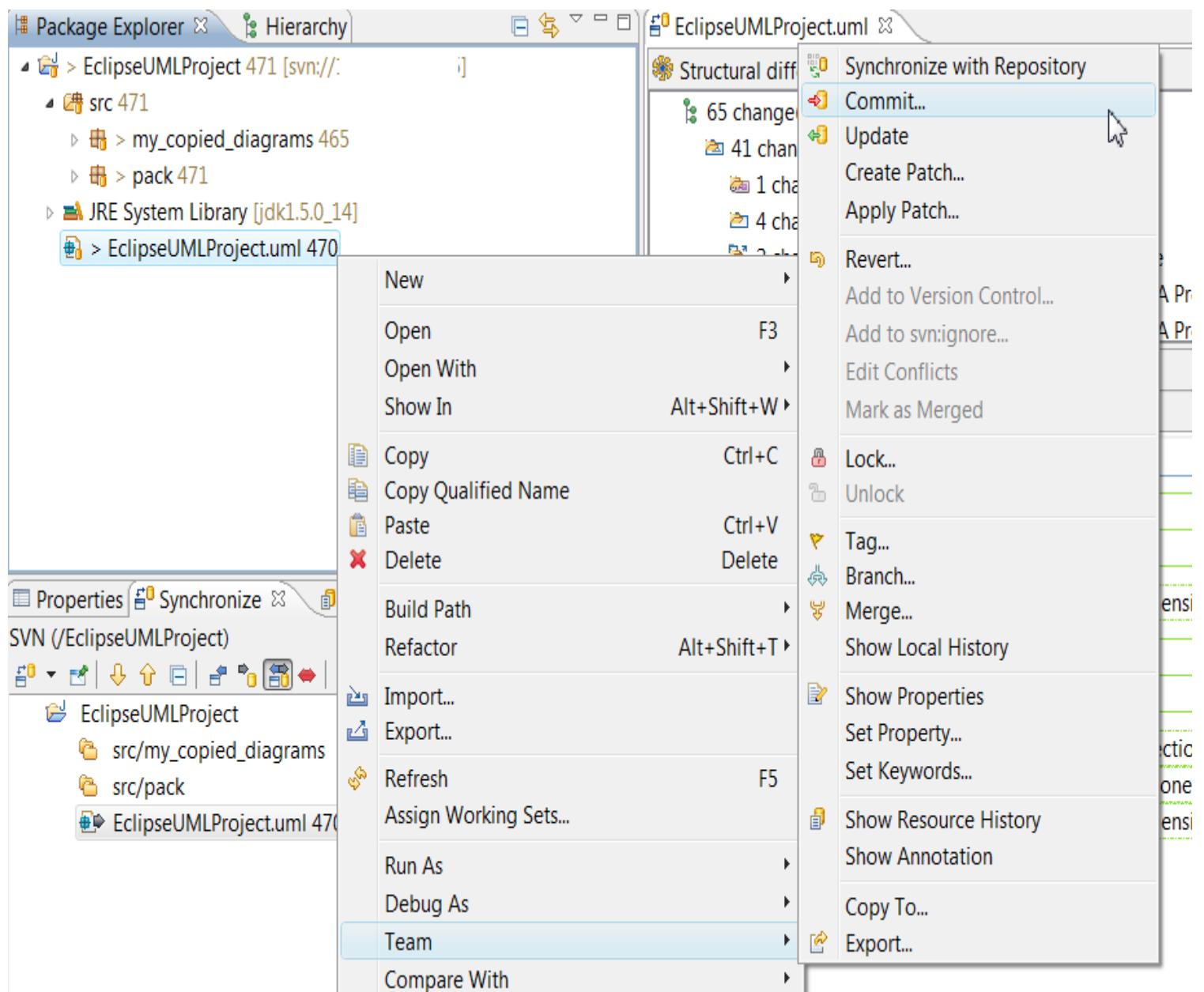
Have a look at changes before committing.

The screenshot shows the Structural Differences view. At the top left, it says 'EclipseUMLProject.uml X' and 'Structural differences'. Below that, it lists '65 change(s) in model' with a detailed breakdown:

- 41 change(s) in <Model> EclipseUMLProject
 - 1 change(s) in General
 - 4 change(s) in <Package> pack
 - 2 change(s) in <Profile Application> JavaProfile
 - 2 change(s) in <Profile Application> AndroMDA Profile Service
 - 2 change(s) in <Profile Application> AndroMDA Profile Webservice

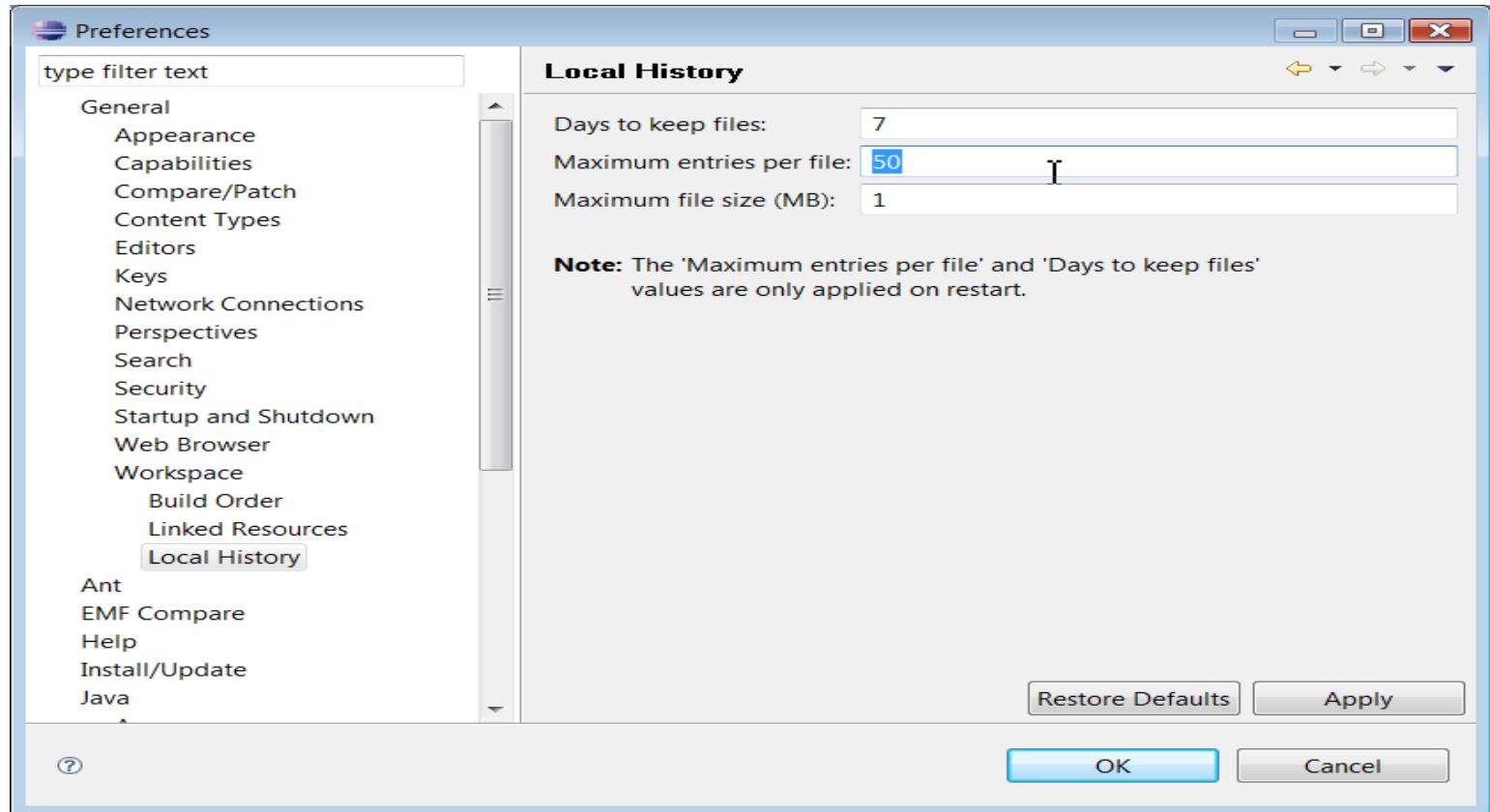
Below this, there is a 'Visualization of Structural Differences' section with two tabs: 'Remote Resource' and 'Local Resource'. Both tabs show a list of structural elements with green lines connecting them, indicating changes between the local and remote resources.

2. or to click on the file inside the Package Explorer and select files to be committed > **Team** > **Commit...**

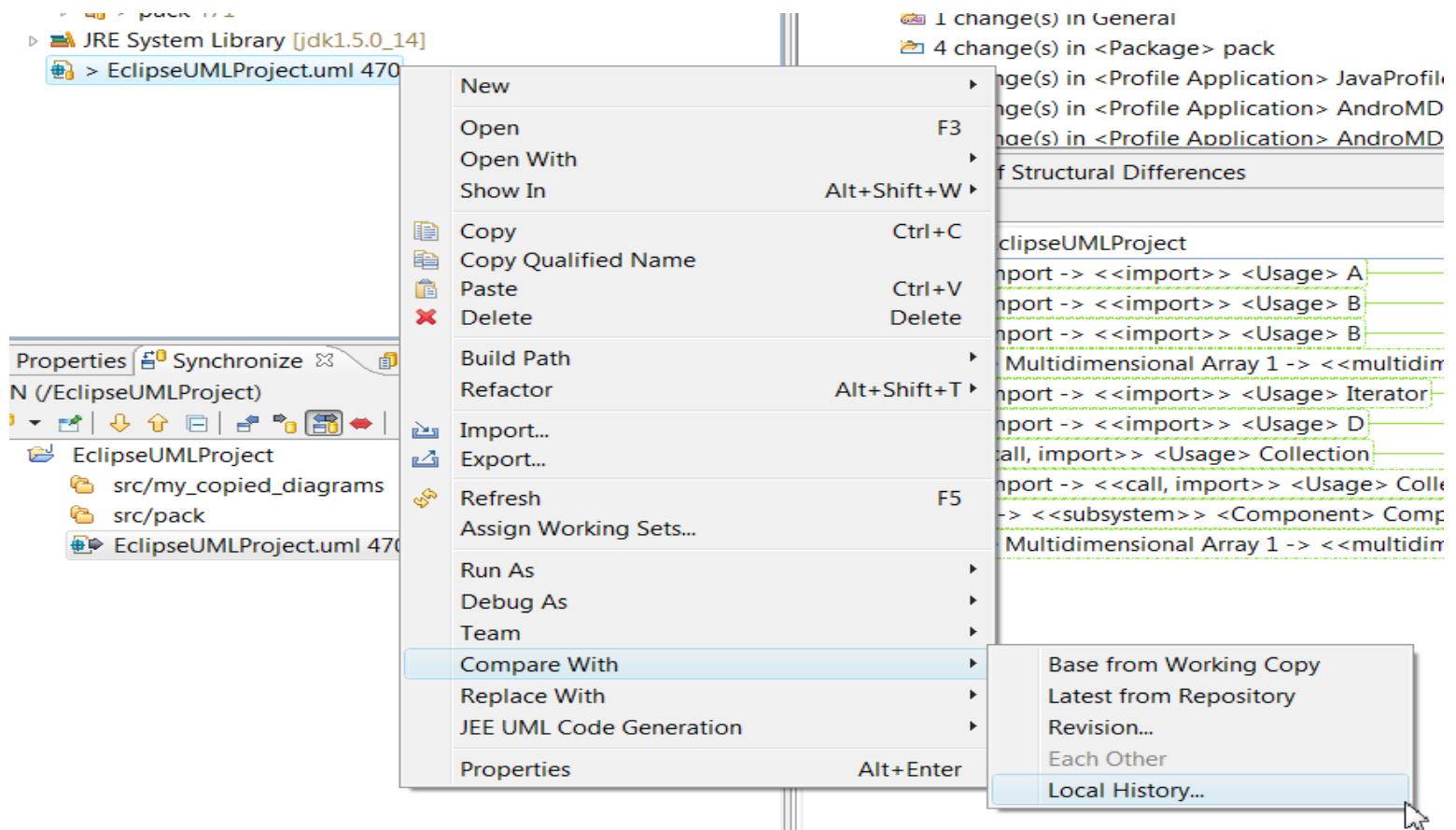


Omondo recommendation: SVN and teamWork are great tools but never forget to activate your local History backup.

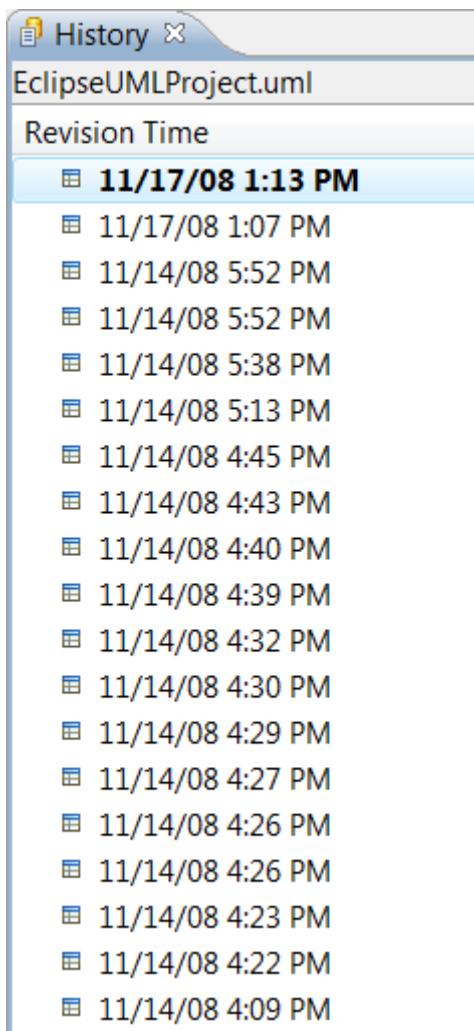
Click on **Window > Preferences > General > Workspace > Local History**.



Click in the Package Explorer on the file > **Compare With** > **Local History...**



You can revert at any time each file of your Package Explorer.



Tips & Tricks

It is important to tune EclipseUML in order to increase modeling productivity.
The following tips will help you:

1. [The UML editor is blocked or I don't see contextual menu anymore.](#)
2. [It seems that EclipseUML is frozen.](#)
3. [How to clean my UML Model.](#)

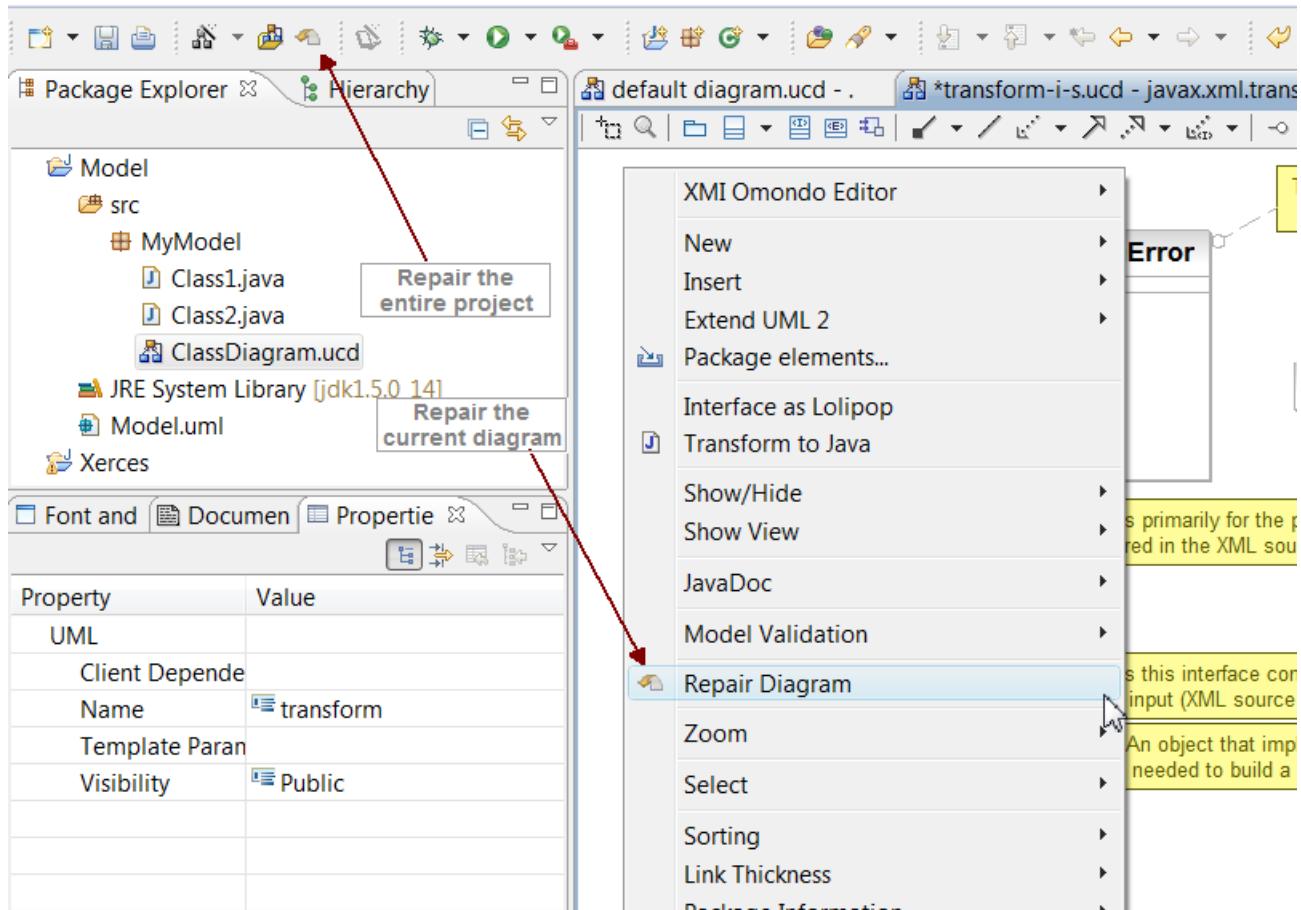
4. [I don't see my UML Model in the Package Explorer](#)
5. [It takes too much time to change diagram preferences.](#)
6. [My Class Diagram is too big after selecting arranges all.](#)
7. [My connectors \(e.g. association, dependencies, inheritance\) are not nice.](#)

1. The UML editor is blocked or I don't see contextual menu anymore.

You can at any time refresh your UML Editor or your full project.

This feature is important if you manipulate heavy diagrams and use live synchronization.

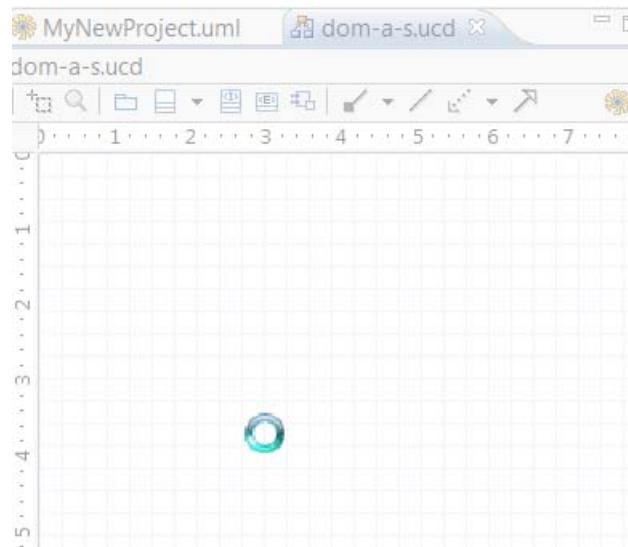
We recommend to immediately using this feature as soon as you feel that something is not perfect.
Doing that will clean everything and allow you to always have a clean model and project.



2. It seems that EclipseUML is frozen.

EclipseUML uses many progress monitors which are only activated at the process launch. We don't activate them anymore after in order not to be too intrusive. If you click on Eclipse during the process then the progress monitor is not anymore visible because the Eclipse window is on the top

of the process monitor. A good test to see if the process is still active is to click on the diagram and see if you have a running process such as below:

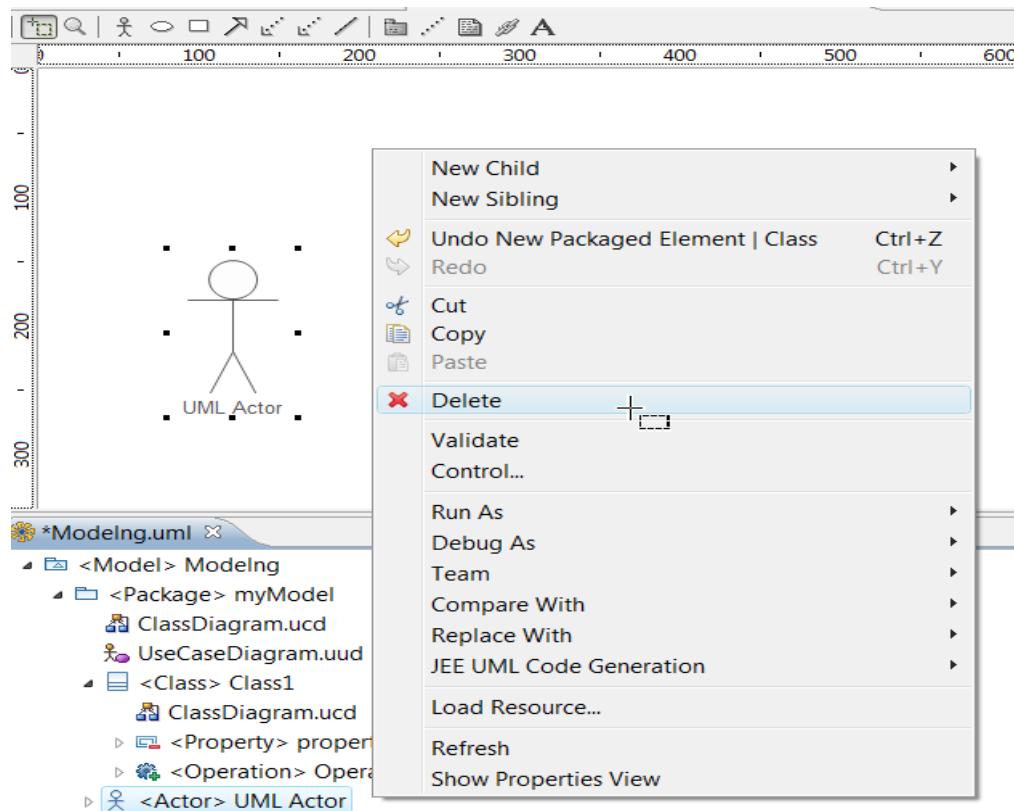


Every click will freeze Eclipse for an additional few seconds. Please don't stop any process before the end because this could damage the live synchronization.

3. How to clean my UML Model.

You can delete, change or add model element because our UML Editor is just a viewer of your UML superstructure.

Do not use Ultra Edit, Spyware or any xml editor because this could break the model integrity.
Only the Omundo XMI Editor or the Eclipse Model Explorer is allowed.



Please note that if you erase an element from your XMI superstructure then the associated UML Editor will still display this element.

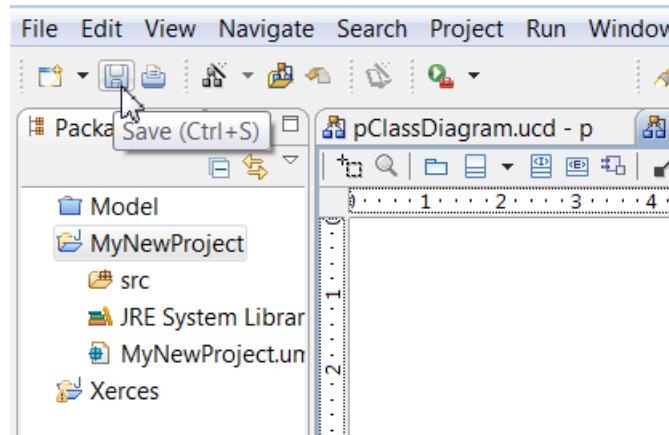
This element will therefore not be related to a model anymore. If you want to erase both UML editor and model then you need to delete the model from the UML Editor.

4. I don't see my UML Model in the Package Explorer

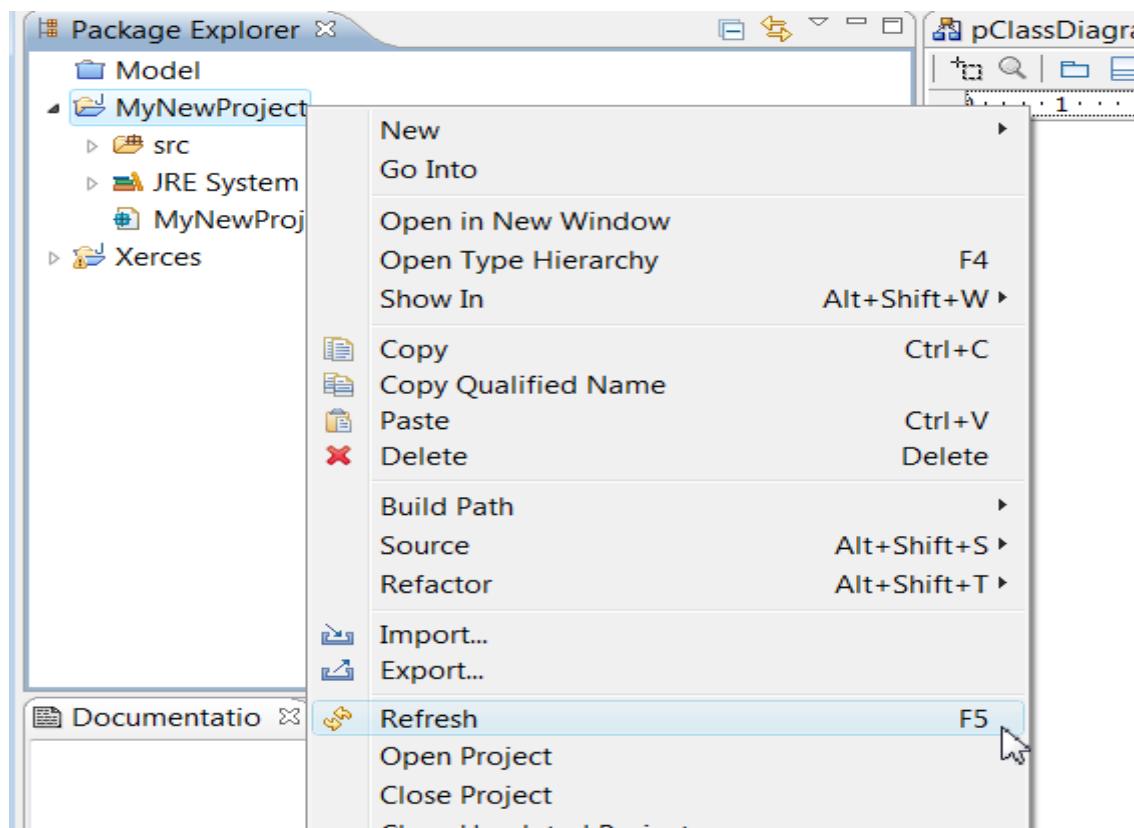
The model is only instantiated by the creation of any diagram if modeling non java elements and by creating a class diagram if java modeling.

It means that if you use reverse engineering features (*e.g. XMI Backup*) and you don't click on the save button then you will not see any model at the root of the project.

You need at least to create one class diagram and click on the save button.

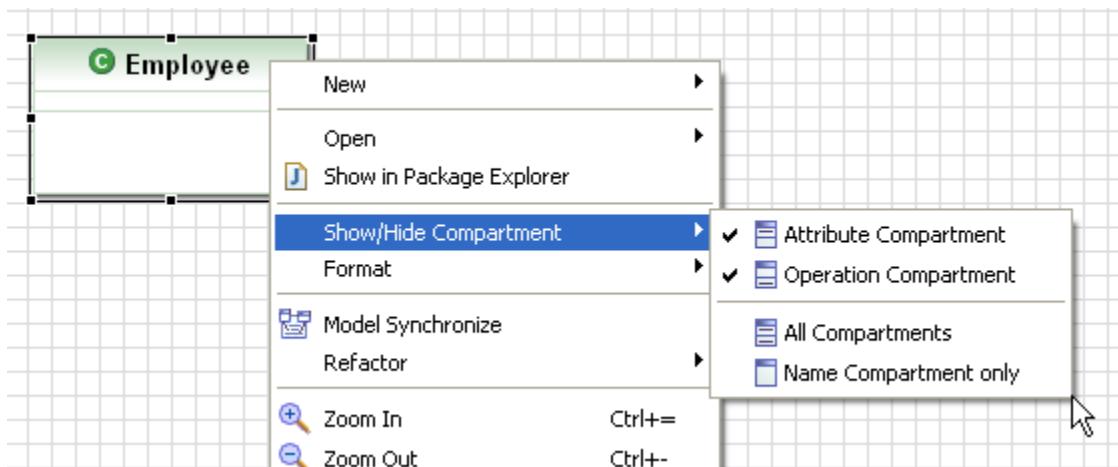


If you still don't see your model after saving a class diagram then just click on your project in **Package Explorer > Refresh**.



5. It takes too much time to change diagram preferences.

The most time consuming process inside EclipseUML is the Attributes and methods synchronization with the model. If you have 100 methods inside a class this is almost more time consuming than 100 classes with no compartment. Use the [Show hide compartment](#) to have a faster EclipseUML

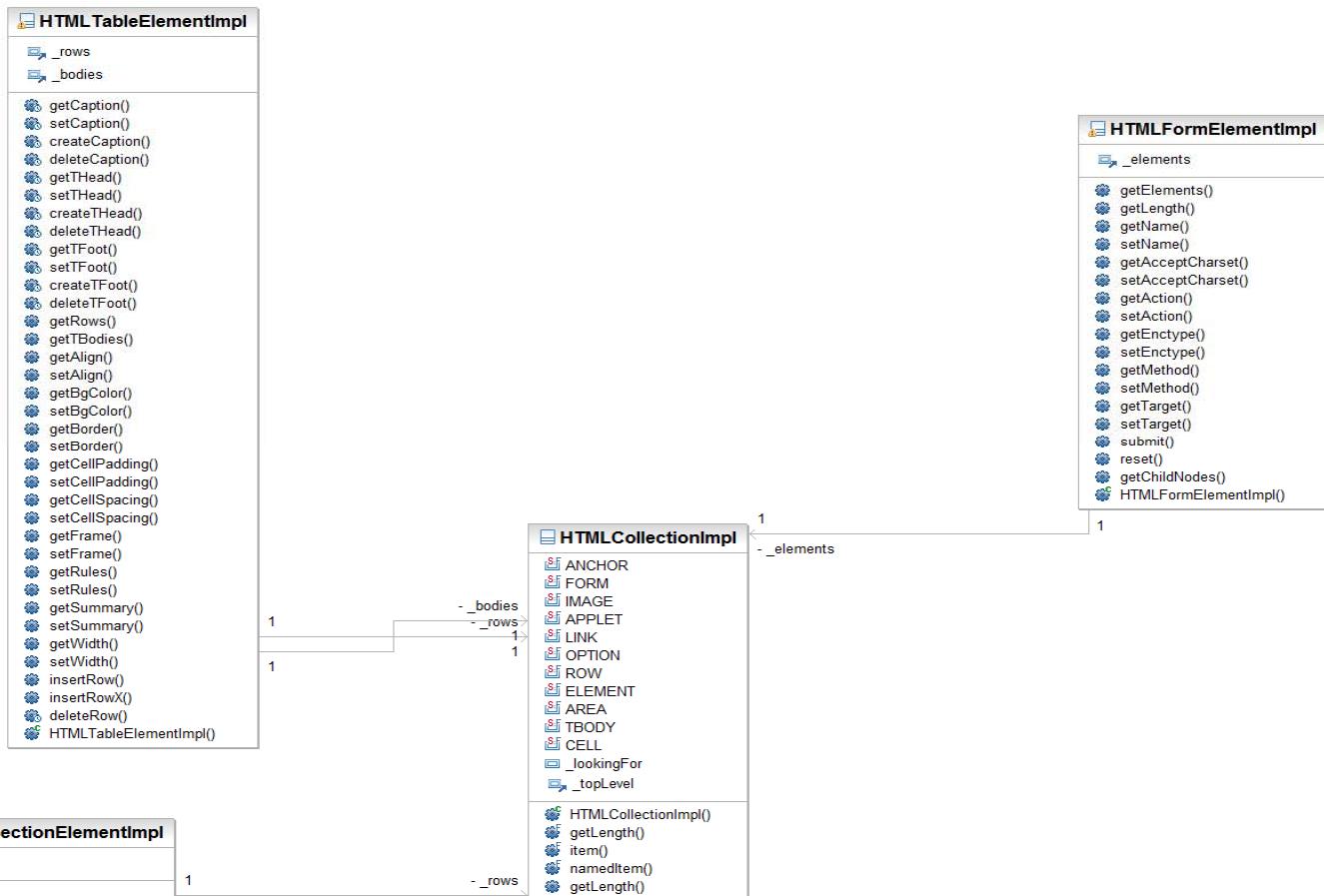


6. My Class Diagram is too big after selecting arranges all.

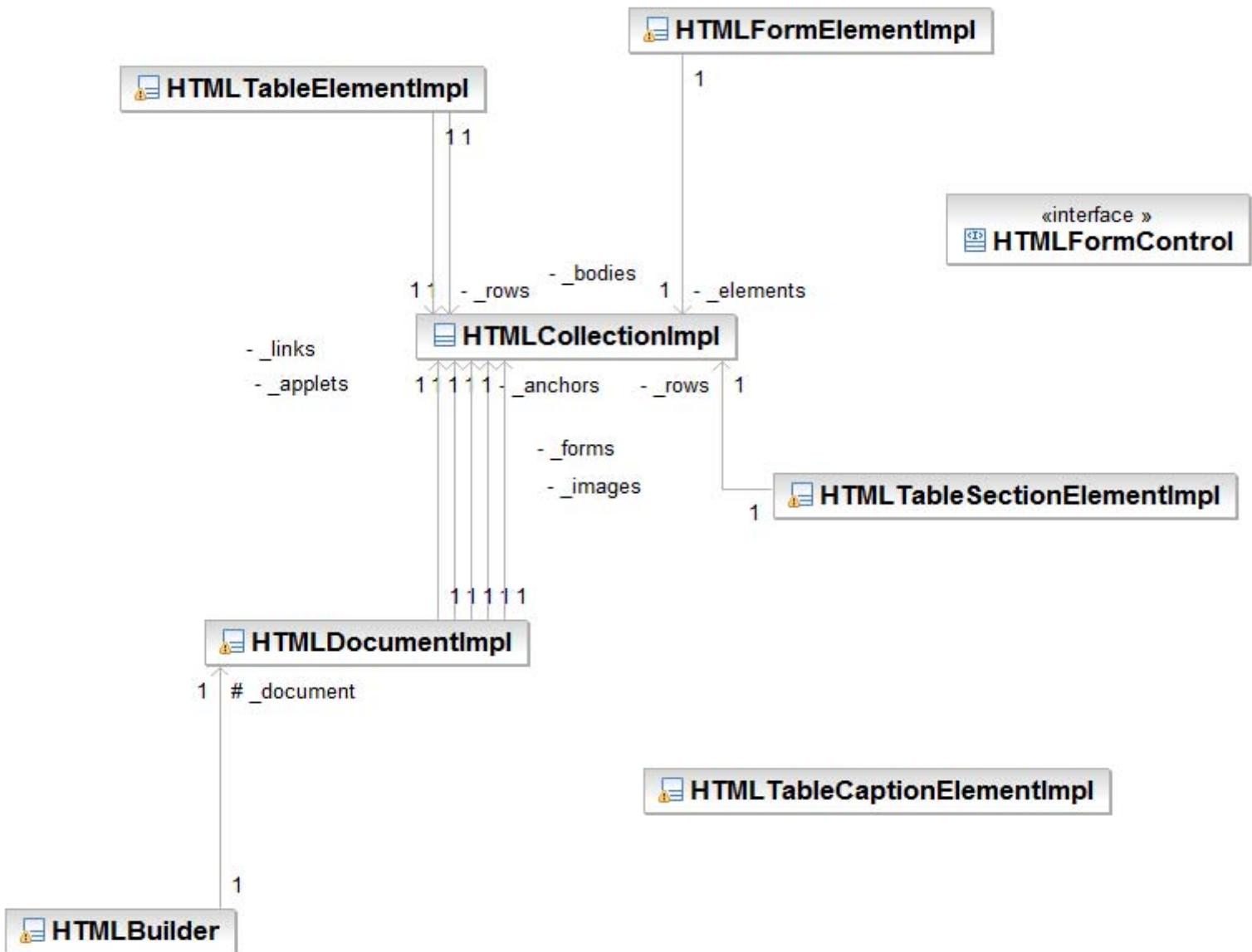
If you reverse a Java code and display associations then you can get a huge class diagram as below which is not really visible and too big for printing or documentation purposes.

The layout mechanism is based on adding classes inside the Class Diagram with no connector's links crossing classes. It means that the smaller is each class, the smaller is diagram.

You can reduce the size of each class by using the [show/hide menu](#) and the width by [hiding package information](#)

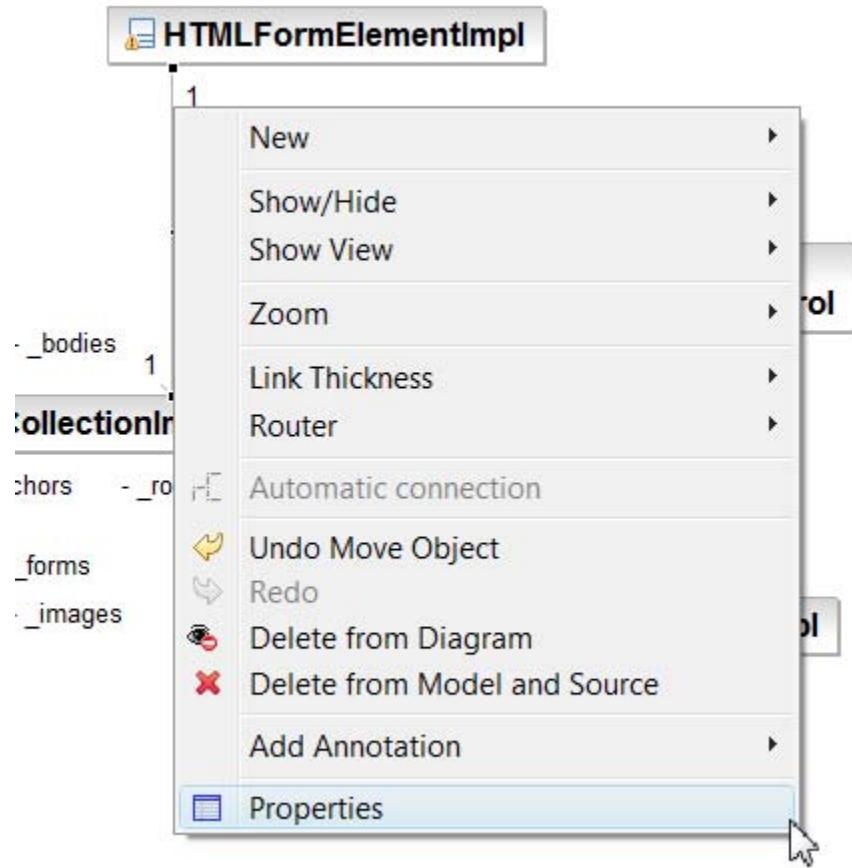


To get a smaller diagram we recommend using the two following features:
First click on the diagram background > **Show/Hide** > **Show/Hide Compartments** > **All Compartment**
Then click on the diagram background and [select > Arrange Diagram](#)

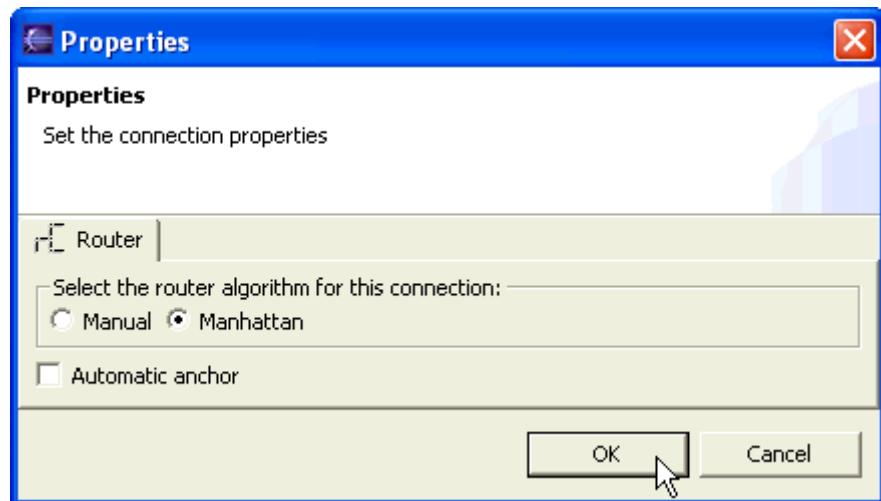


7. My connectors (e.g. association, dependencies, inheritance) are not nice.

You can customize Routers and anchors properties inside your diagram editor.
Select a group of links> **Link thickness** or **Router** to customize them.
Right Click on a link between two elements >**open the pop up menu>Properties** to change one link property

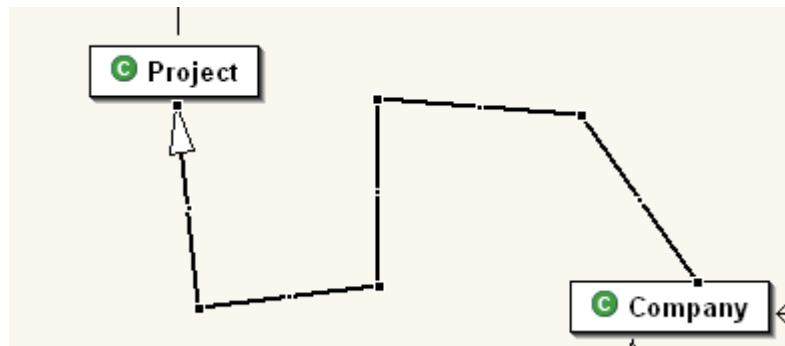


You can select the router algorithm and Anchor for each link



Router:

Manual allows moving the extremity of each link.

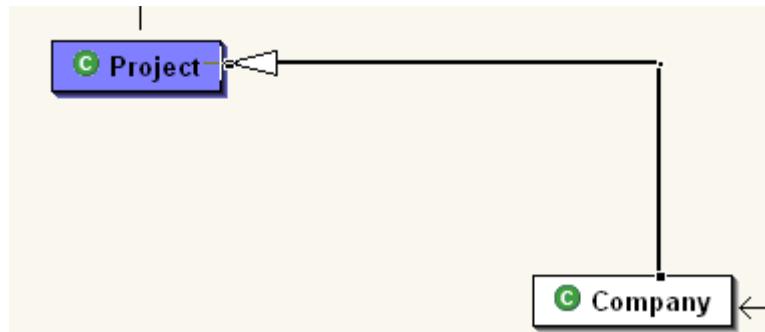


Manhattan uses automatic functions to design the link.

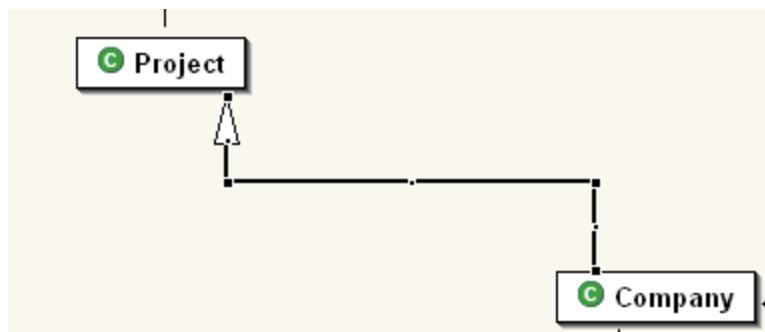


Anchor:

Unselecting Automatic anchor allows fixing the extremity anywhere. Move the anchor inside the element which will become blue when the anchor is properly fixed.



Automatic anchor uses algorithm to fix the anchor automatically (the anchor cannot be moved)



See a class diagram flash demo at <http://www.forum-omondo.com/viewtopic.php?f=29&t=1493>

See state diagram flash demo at: <http://www.forum-omondo.com/viewtopic.php?f=29&t=1482>

8. How to move manually my UML Elements in order to get a perfect shape

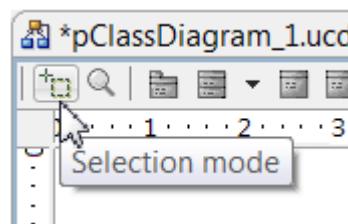
You can move any graphical element inside any diagram by selecting the element with the mouse left button and then using the keyboard **Ctrl + Alt+ keyboard Arrows** to move it each time you press on an arrow.

This feature is important when you want to have strait lines or design complex diagrams in order to get a perfect shape of your design.

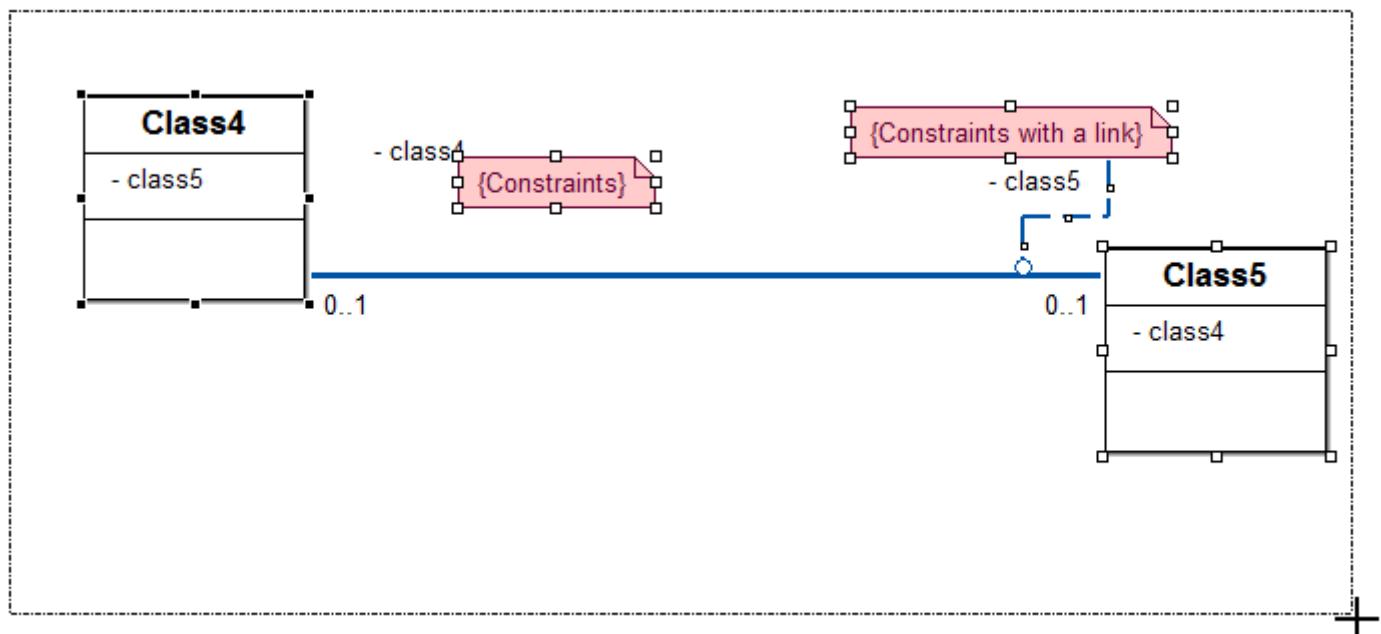
9. How to manually move a group of Elements at the same time

You first need to select a group of elements using:

1. The Selection mode icon



Drop the selection into the diagram and crop elements.



2. Using the keyboard. Click on an element using the mouse and keep the Ctrl key pressed.

You can select as many as needed elements to move them manually at the same time.

Tips and tricks sequence diagram example

The following sequence diagram example is coming from a customer who was blocked while modeling because manually moving up and down his messages.

The received message by our support team was "What I see happen is that as I move a message up then, yes, all messages below get pushed down. Some are pushed out of the frame. If I then extend the lifeline of the entity the focus of control blocks also become longer. the then the diagram keeps growing down. Additional messages are moved outside the frame. If I extend the frame again I cannot move those additional messages which no longer have end points attached to lifelines.

We provided him the tricks example below which helped him to redraw and repair the full diagram."

Our support team answer was: "The message is attached to the activation bar. If new messages are added on the top of existing messages then all messages are pushed down in order to include the new activation bar+message needed space."

The sequence diagram example is composed by:

1. [The initial diagram having layout problem](#)
2. [The recommended solution to fix the layout](#)
3. [The final diagram after using tips and tricks](#)

1. The initial diagram having layout problem

The following picture is the initial diagram which was sent to our support team.

You can upload and try to remodel this diagram

- xml file: [bad_diagram_layout.usd](#)
- zip file to unzip in the Eclipse project : [bad_diagram_layout.zip](#)

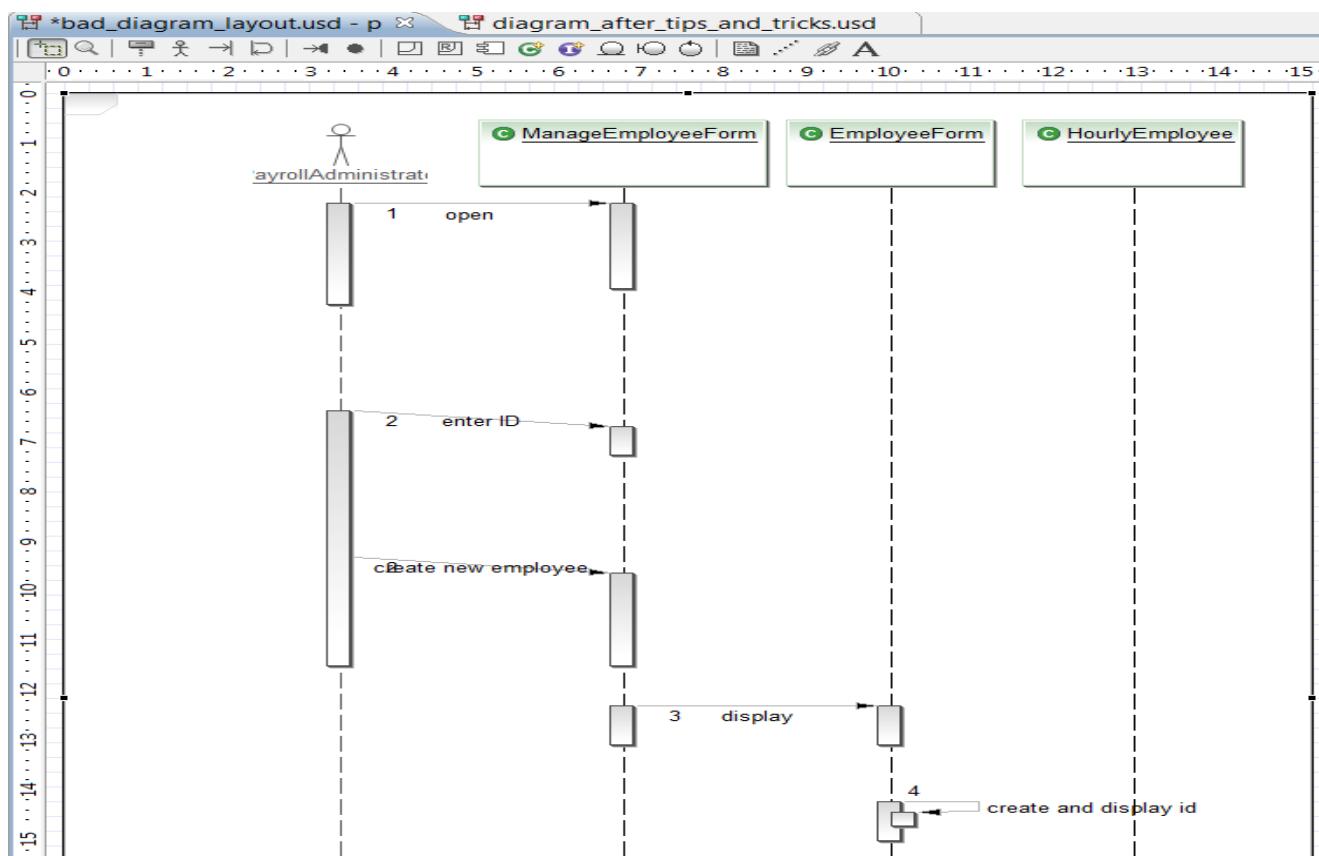
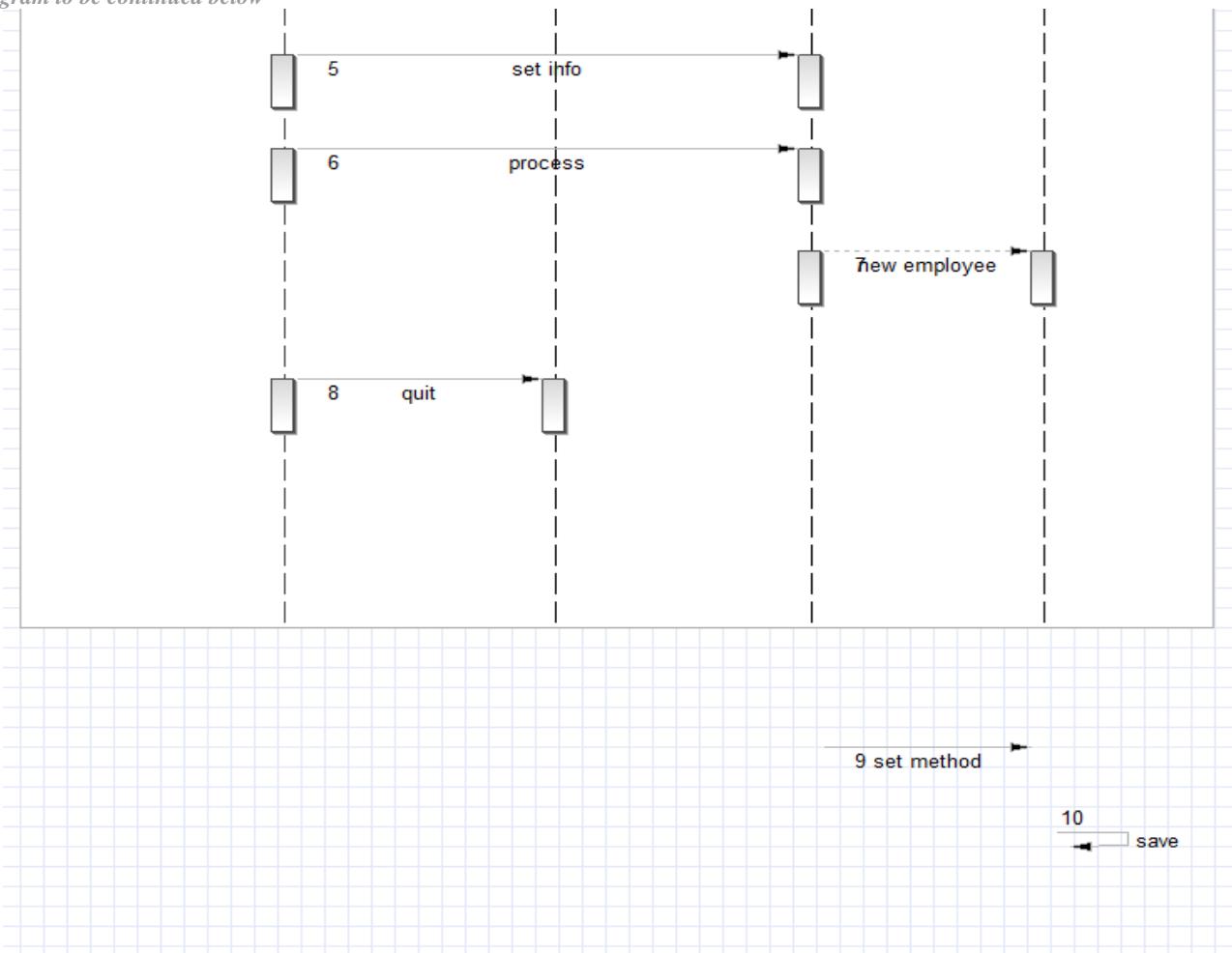


Diagram to be continued below

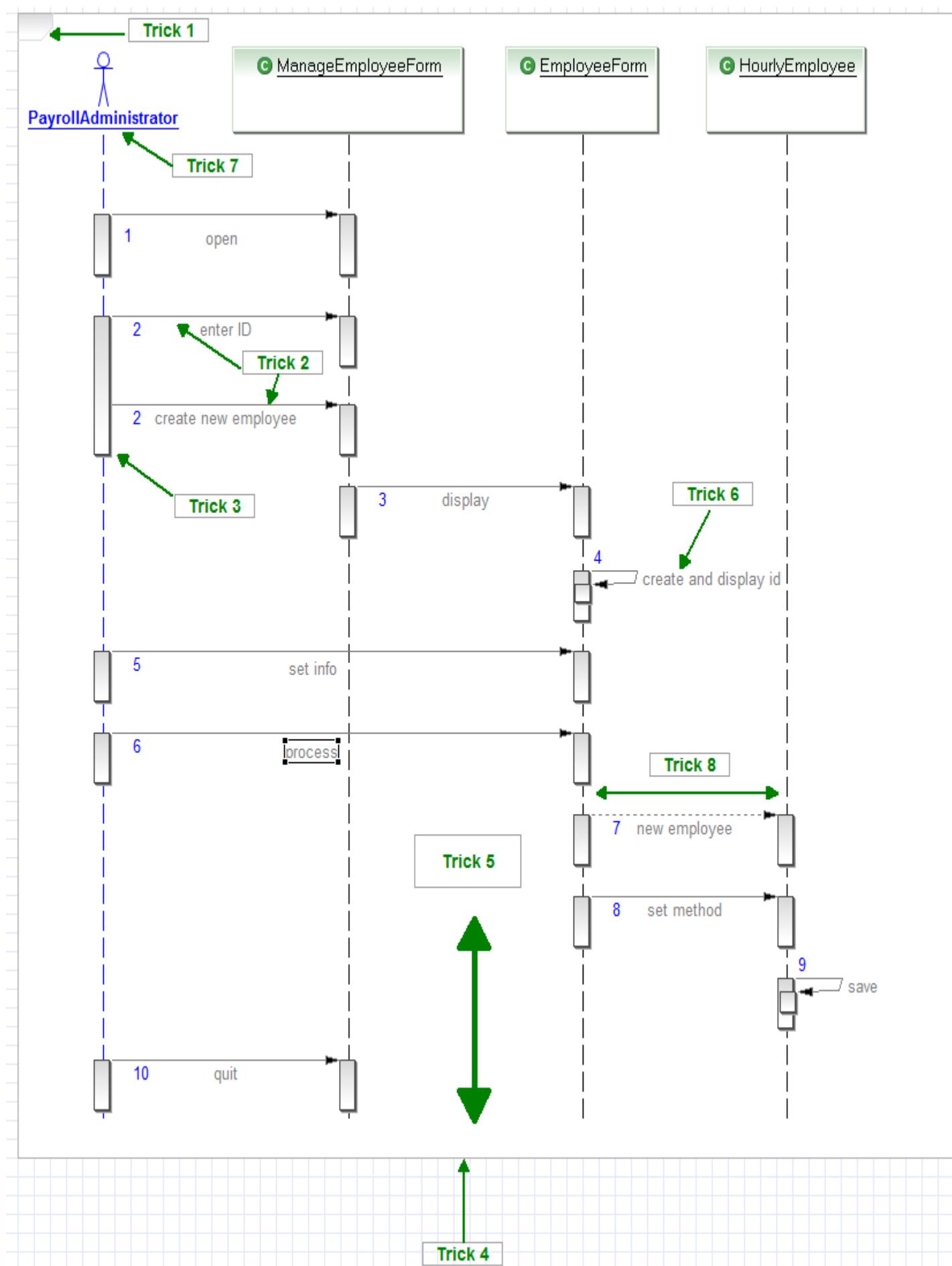


2. The recommended solution to fix the layout

Have a look at the tips_tricks_example.png picture below.

Each trick on the picture has a number which is explained below:

1. You can **add a name to your main frame**: [See more](#)
2. Have a **straight line message** between two lifeline : [See more](#)
3. Have a **smaller activation bar**. The activation bar should at least have the size of the target activation bar. You can therefore minimize the target activation bar using the mouse and then the other one. Don't forget that the message will be created at the mouse position click. [See where will be created your message and more](#)
4. Use the mouse to **enlarge the main frame manually**. Click on the frame, select the black point in the middle and drag it down then drop it in order to get a bigger frame.
5. Click on the diagram **background > Lifeline Layout** in order to **have messages the same size and going down to the bottom of the main frame**. [See more](#)
6. Click on the message or use the Sequence Diagram contextual menu > Select > All Connectors then use the Font and Colors View to **change the colors of your message**. [See more](#)
7. Click on the PayrollAdministrator Actor and use the mouse to **get a larger width in order to have visible name**.
8. Move each instance on the right or on the left (e.g. ManageEmployeeForm, HourlyEmployee etc....) with the mouse in order to **have more readable message name**



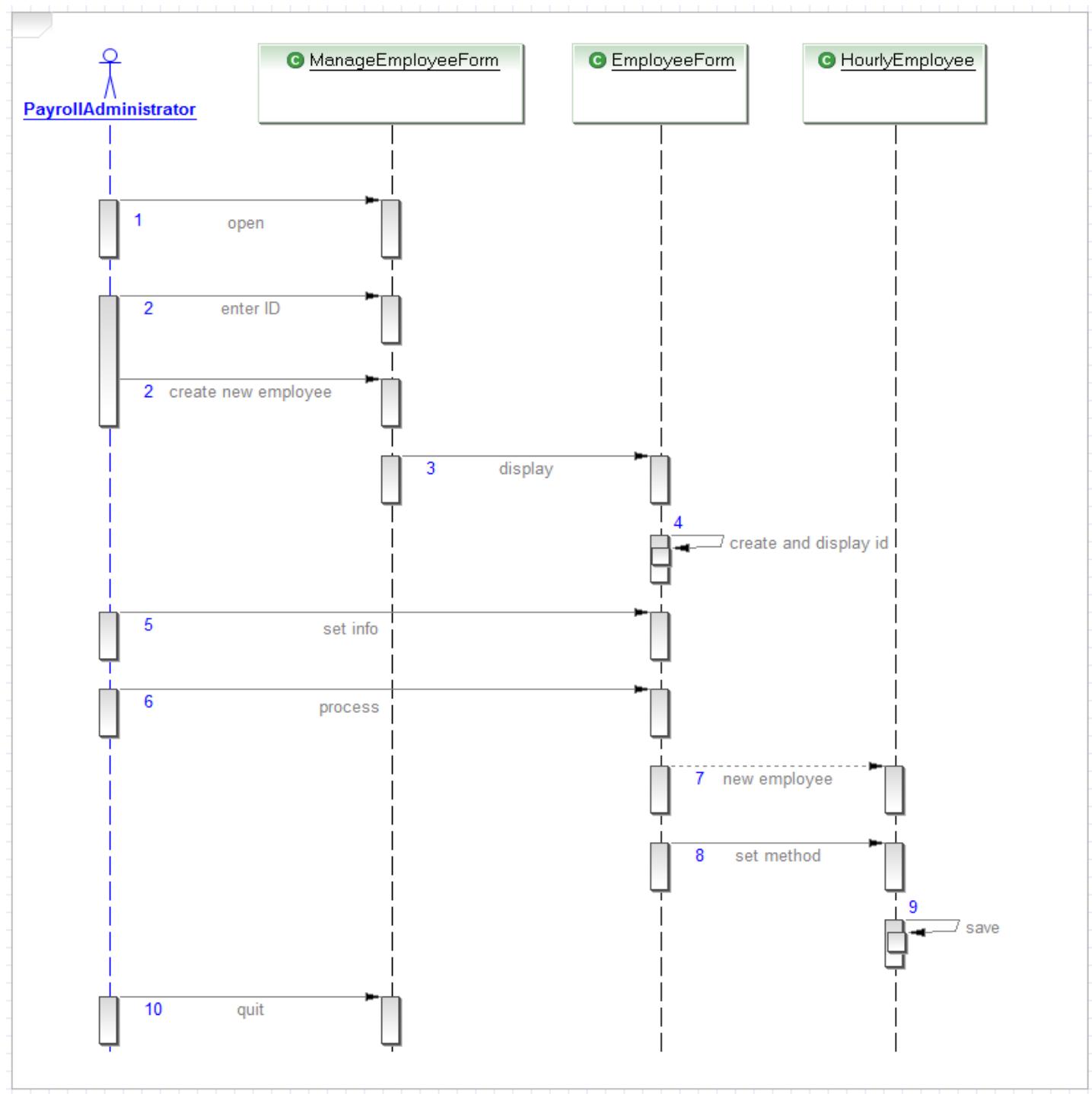
3. The final diagram after using tips and tricks

the following picture is the final diagram which was fixed by our support team.

The below diagram is only using contextual menus which are already available in the sequence diagram as features related to the tips and tricks documentation chapter.

You can upload and try to remodel this diagram:

- xml file:: [diagram_after_tips_and_tricks.usd](#) (no model just the graphical file like a .usd format)
- zip file to unzip in the Eclipse project : [diagram_after_tips_and_tricks.zip](#)



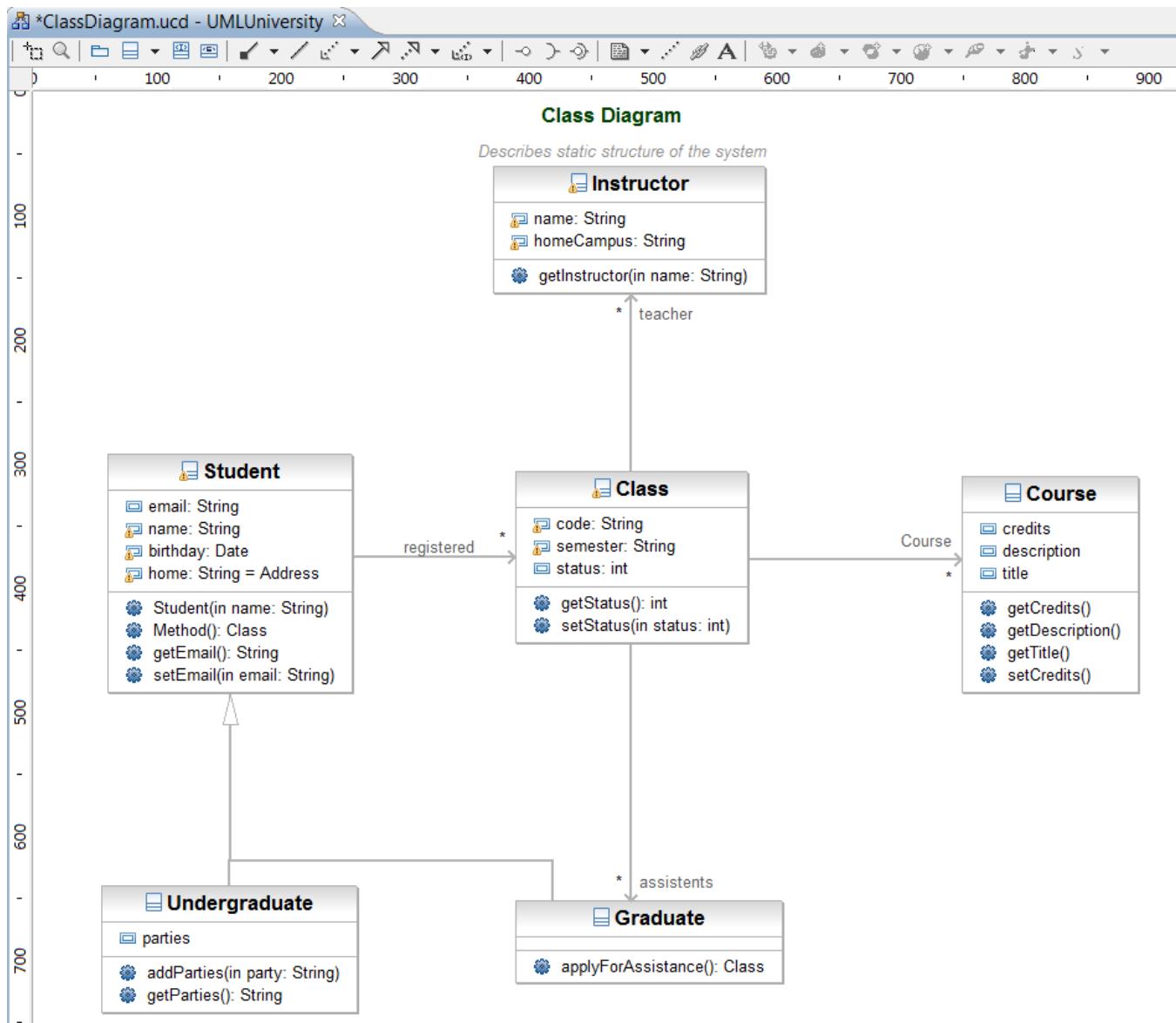
The New EclipseUML UML 2.2 Diagrams

EclipseUML is an intuitive, fully featured and very powerful tool which support all 13 UML 2.2 diagrams.

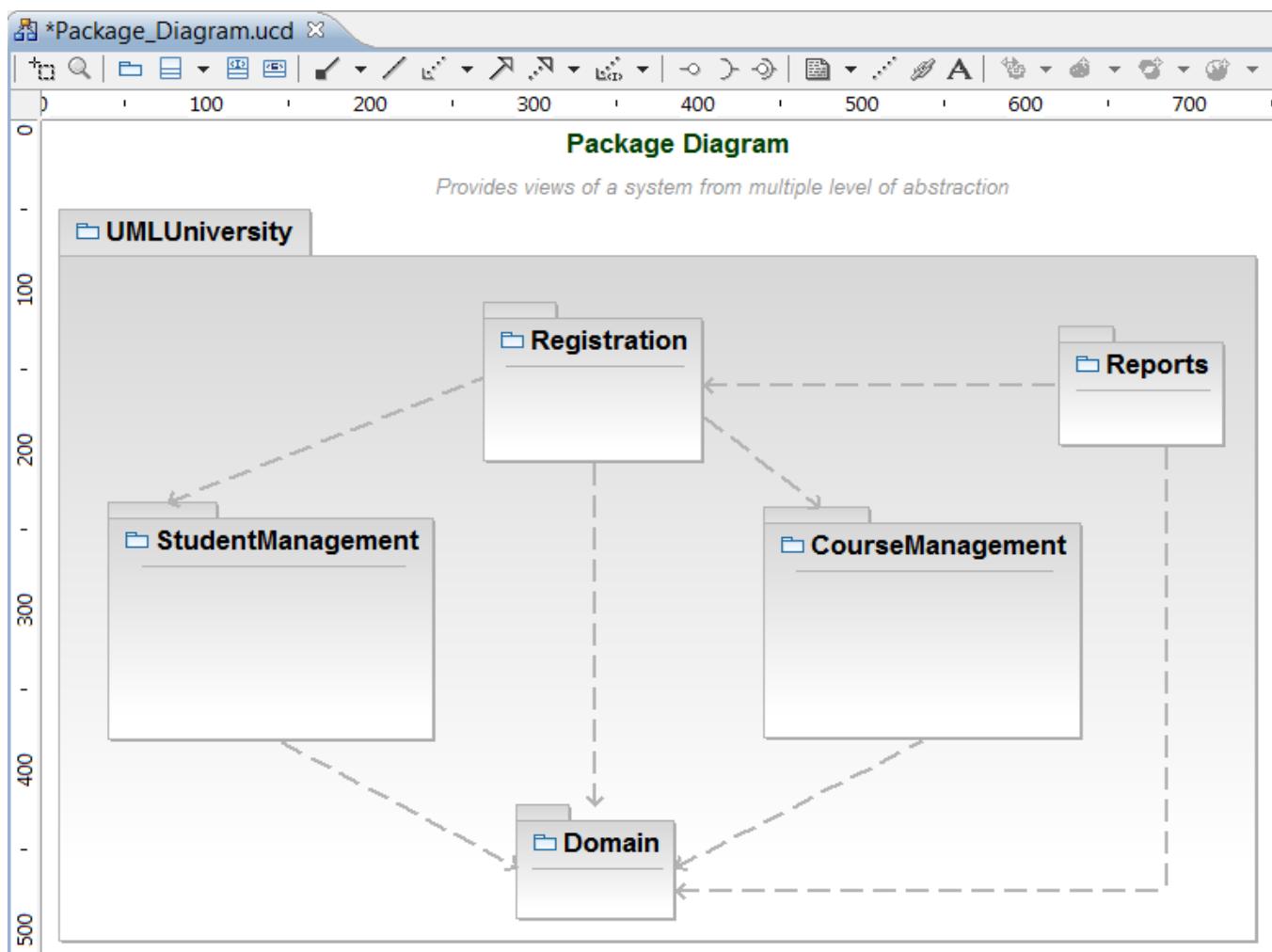
The UML 2.2 diagrams below show an example of student enrollment in a UML University:

- **Structural View:** Class, Package, Object Diagrams
- **Implementation :** Component Diagram
- **Behavioral View :** Sequence, State, Activity Diagrams
- **Environment View :** Deployment Diagram
- **Reverse Engineering** example is using the Composite Structure Diagram
-

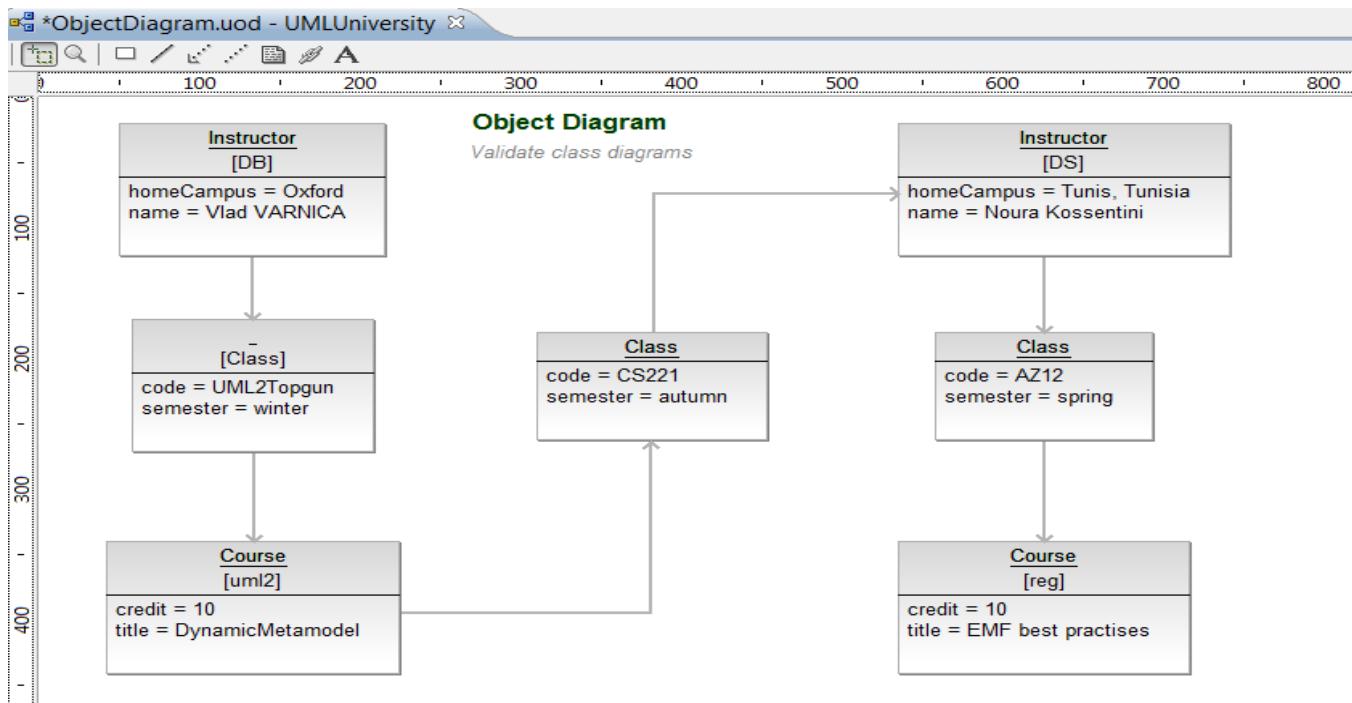
Class Diagram



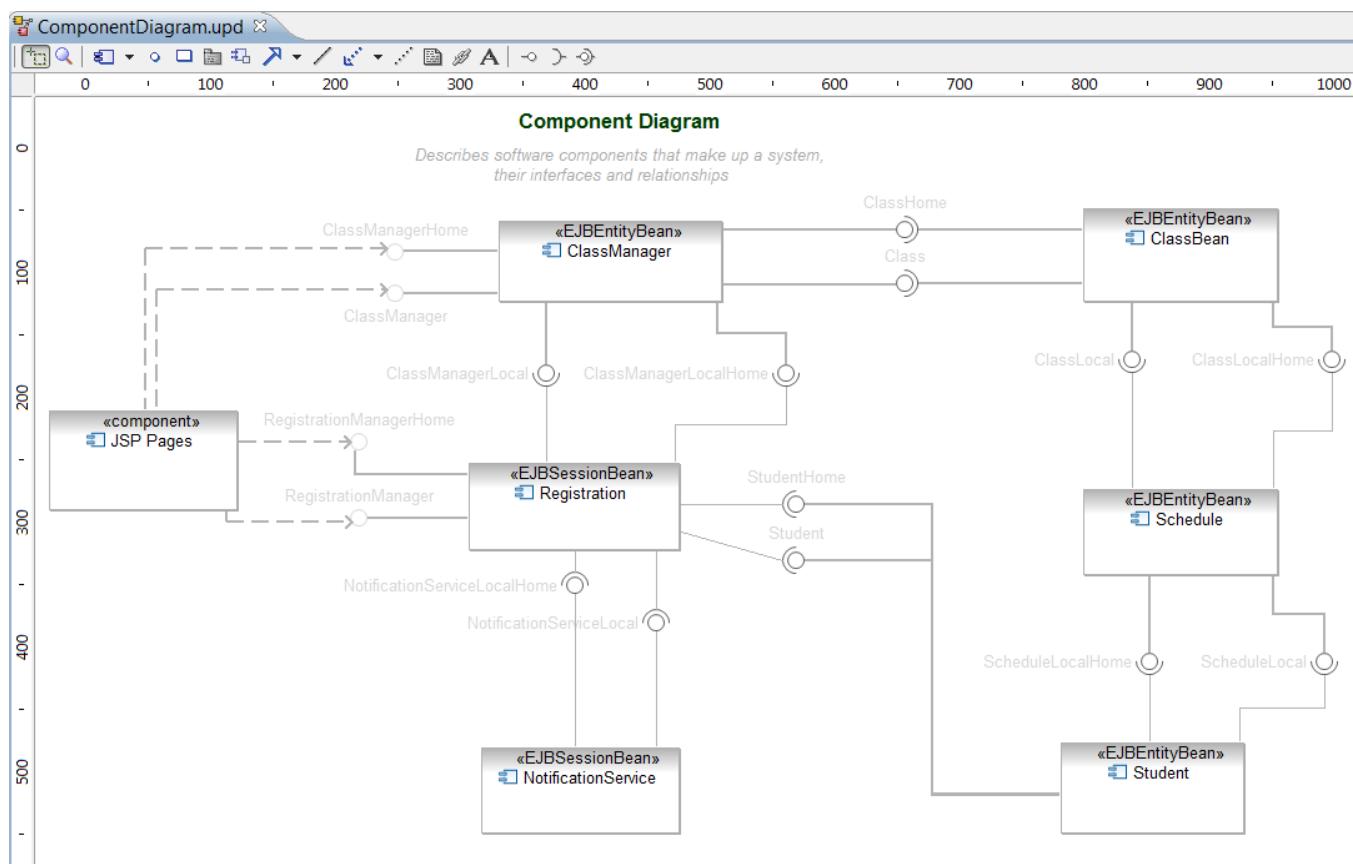
Package Diagram



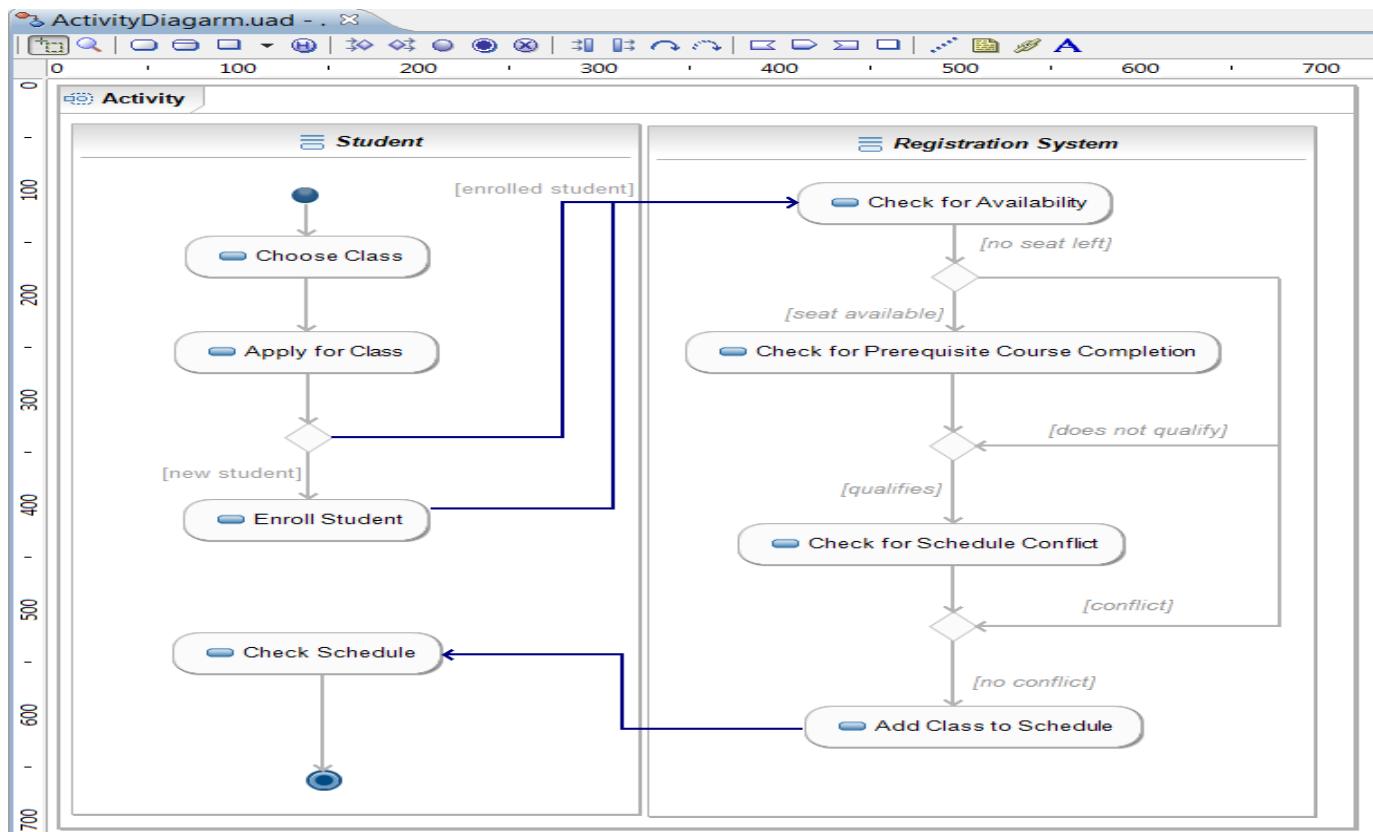
Object Diagram



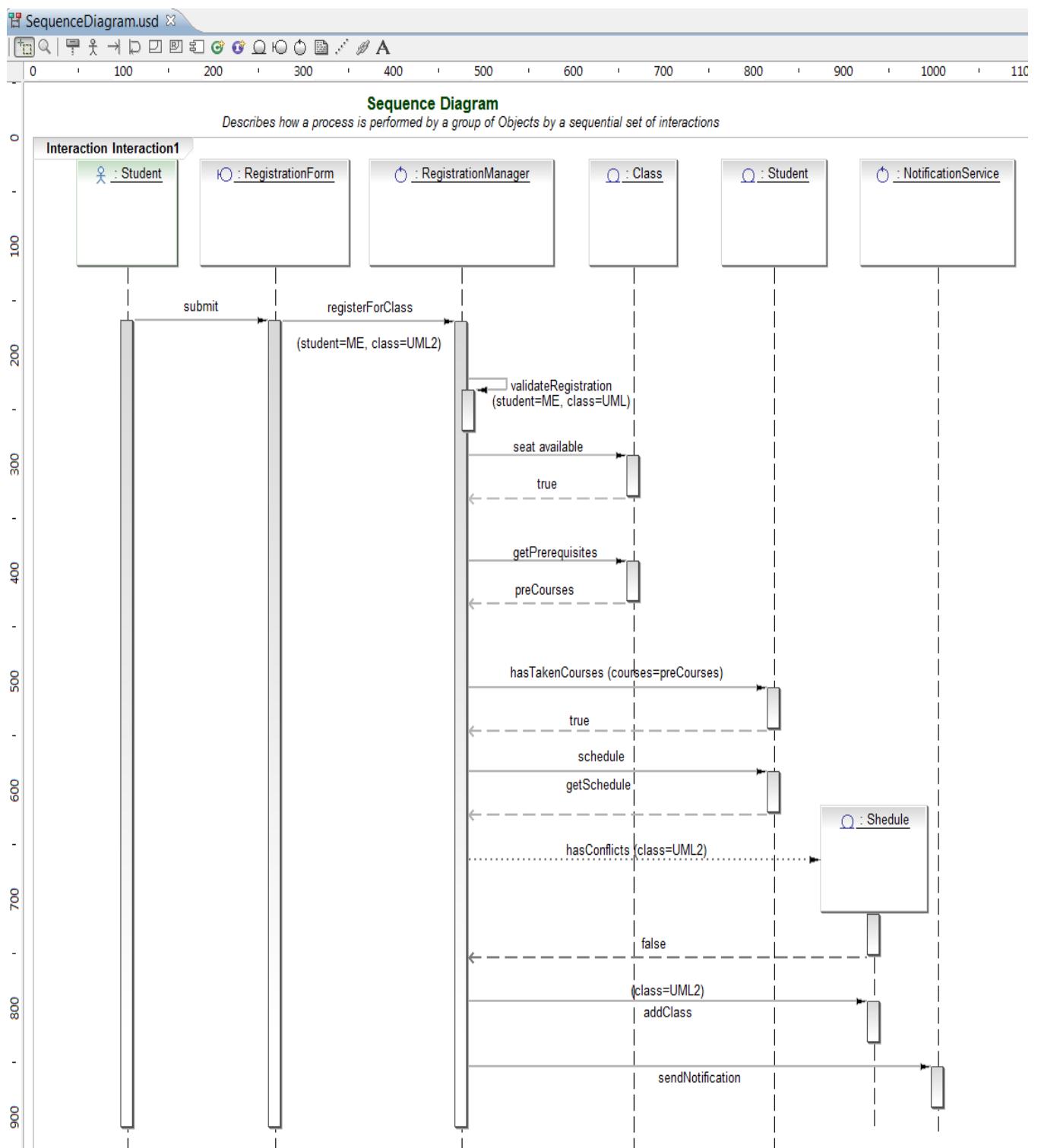
Component Diagram



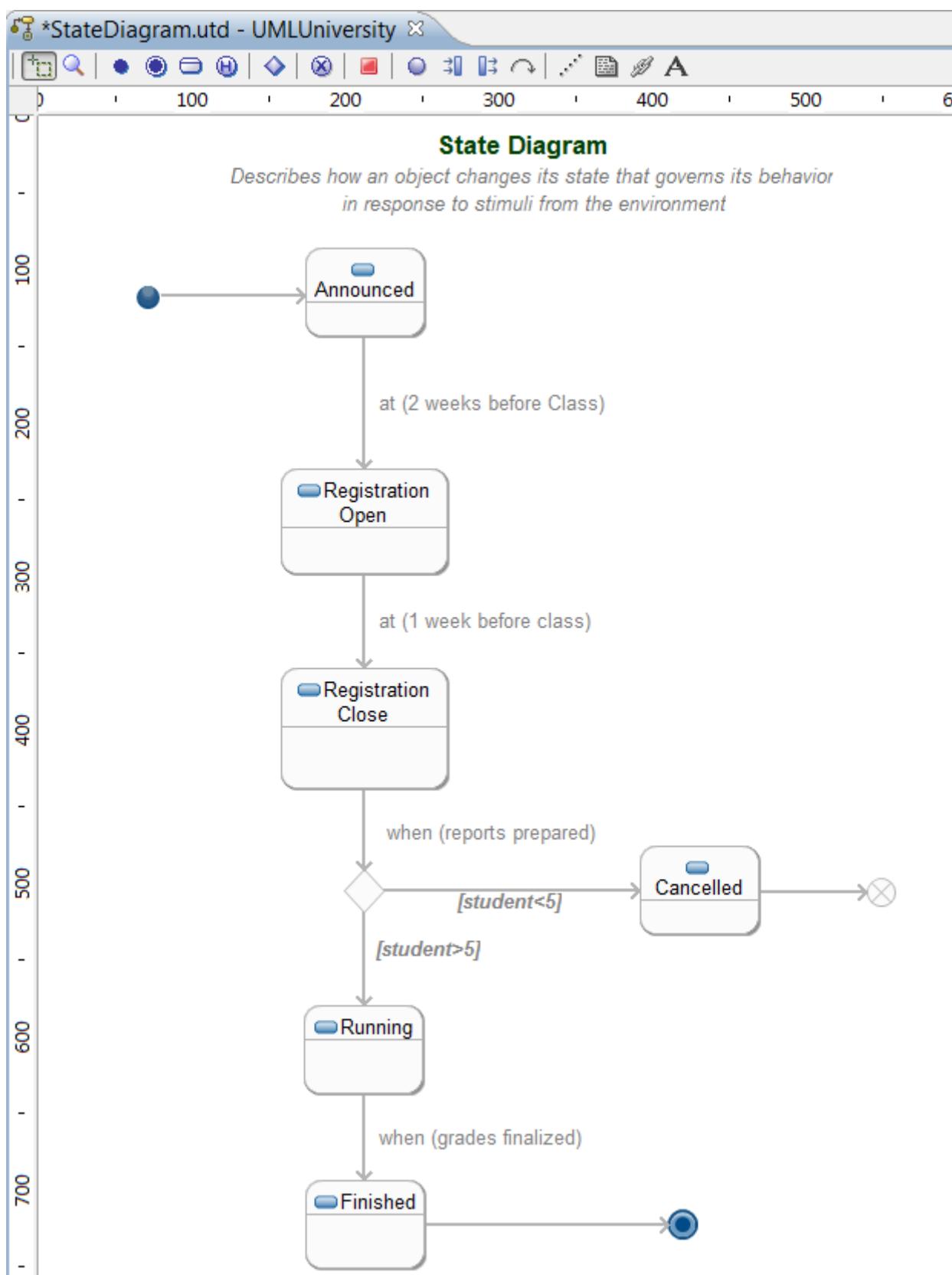
Activity Diagram



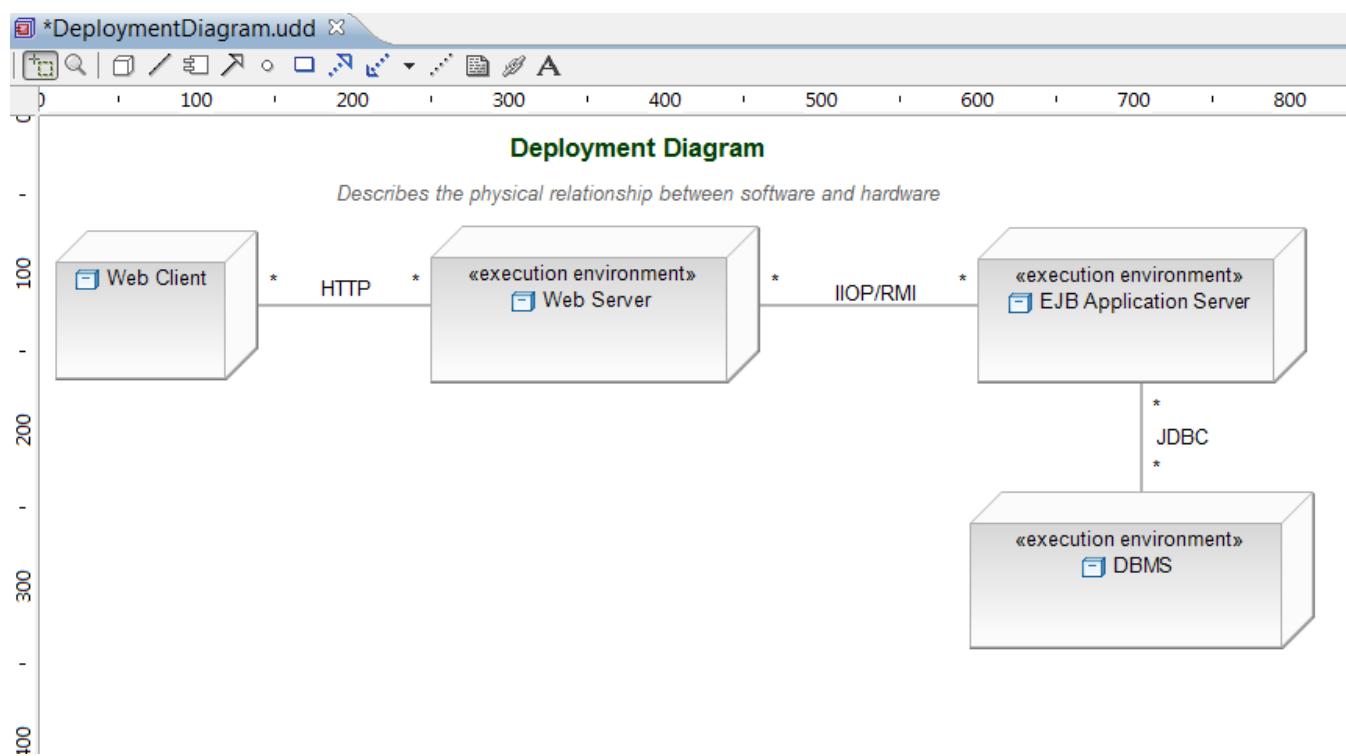
Sequence Diagram



State Diagram



Deployment Diagram



Composite Structure Diagram

