

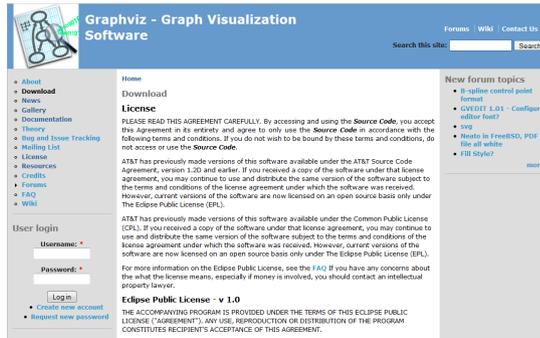


Tutorial

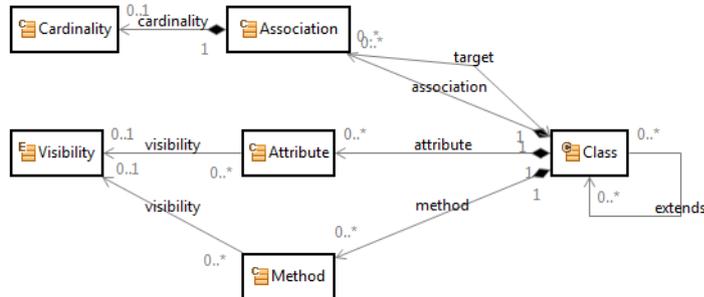
Diagram with Graphviz

| Tutorial | Actifsource Tutorial – Diagram with Graphviz |
|----------------|---|
| Required Time | <ul style="list-style-type: none"> • 60 Minutes |
| Prerequisites | <ul style="list-style-type: none"> • Actifsource Tutorial – Installing Actifsource • Actifsource Tutorial – Simple Service |
| Goal | <ul style="list-style-type: none"> • Write a template to generate a UML Class Diagram as Graphviz DOT file • Generate a SVG graphic using GraphvizBuildTask |
| Topics covered | <ul style="list-style-type: none"> • Setup Graphviz • Create an UML model • Create a graph template for Graphviz • Add Visibility to Methods and Attributes • Distinguish Association types |
| Notation | <ul style="list-style-type: none"> •  To do •  Information • Bold: Terms from actifsource or other technologies and tools • <u>Bold underlined</u>: actifsource Resources • <u>Underlined</u>: User Resources • <u><i>UnderlinedItalics</i></u>: Resource Functions • Monospaced: User input • <i>Italics</i>: Important terms in current situation |
| Disclaimer | <p>The authors do not accept any liability arising out of the application or use of any information or equipment described herein. The information contained within this document is by its very nature incomplete. Therefore the authors accept no responsibility for the precise accuracy of the documentation contained herein. It should be used rather as a guide and starting point.</p> |
| Contact | <p>actifsource GmbH Täfernstrasse 37 5405 Baden-Dättwil Switzerland www.actifsource.com</p> |
| Trademark | <p>actifsource is a registered trademark of actifsource GmbH in Switzerland, the EU, USA, and China. Other names appearing on the site may be trademarks of their respective owners.</p> |

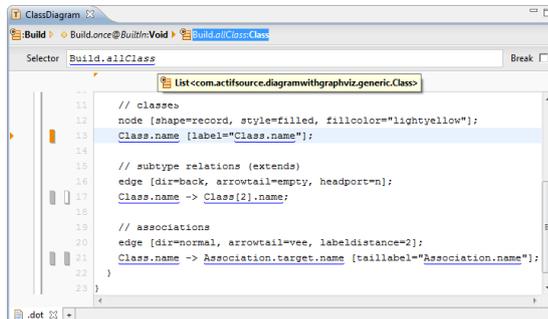
- Setup Graphviz



- Create a UML model



- Create a graph template for Graphviz



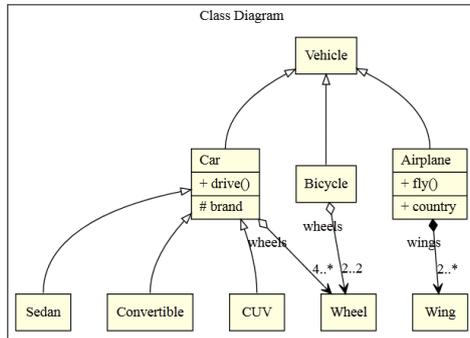
- Add more UML features

```

11 // classes
12 node [shape=record, style=filled, fillcolor="lightyellow"];
13 Class.name [label="{Class.name}\l|\
14 Visibility.umlSymbol Method.name ()\l\
15 |\\
16 Visibility.umlSymbol Attribute.name\l\
17 }"];
18 Class.name [label="Class.name"];

```

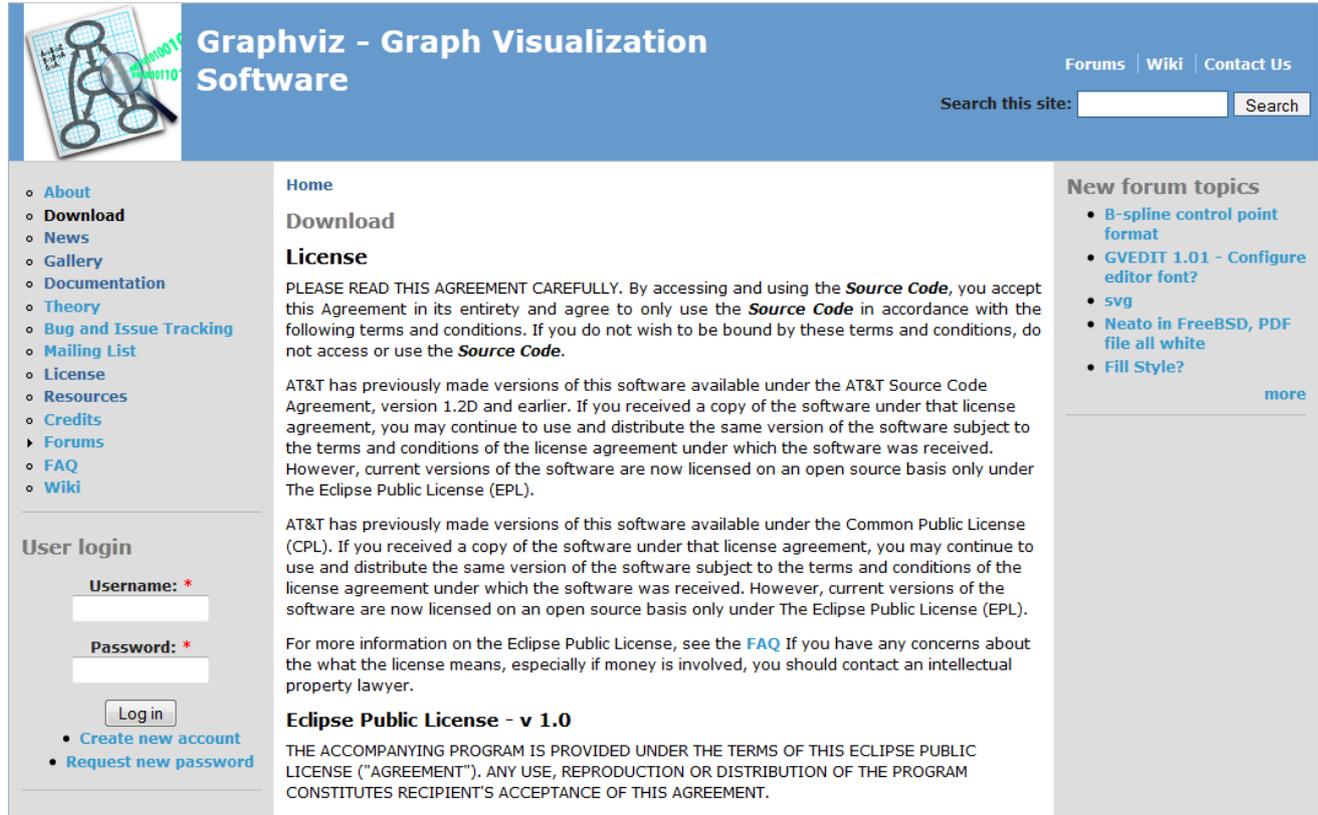
Class.method union Class.attribute:NamedResource



Preparation

- ① Setup the Graphviz software.
- ① Set Built-in Dependencies of an actifsource Project for graph generation.

- ① Before you can generate UML diagram, you need a running environment of Graphviz. If you already have the Graphviz commands working from command line (with the PATH environment variable set), you can skip the following setup and proceed with the actifsource Project settings.
- ① Graphviz is an open source graph visualization software. Previously developed by AT&T it is now available under Eclipse Public Licence.
- ↪ Start your internet browser and open the address www.graphviz.org.



Graphviz - Graph Visualization Software

Forums | Wiki | Contact Us

Search this site:

- o [About](#)
- o [Download](#)
- o [News](#)
- o [Gallery](#)
- o [Documentation](#)
- o [Theory](#)
- o [Bug and Issue Tracking](#)
- o [Mailing List](#)
- o [License](#)
- o [Resources](#)
- o [Credits](#)
- [Forums](#)
- o [FAQ](#)
- o [Wiki](#)

User login

Username: *

Password: *

- [Create new account](#)
- [Request new password](#)

Home

Download

License

PLEASE READ THIS AGREEMENT CAREFULLY. By accessing and using the **Source Code**, you accept this Agreement in its entirety and agree to only use the **Source Code** in accordance with the following terms and conditions. If you do not wish to be bound by these terms and conditions, do not access or use the **Source Code**.

AT&T has previously made versions of this software available under the AT&T Source Code Agreement, version 1.2D and earlier. If you received a copy of the software under that license agreement, you may continue to use and distribute the same version of the software subject to the terms and conditions of the license agreement under which the software was received. However, current versions of the software are now licensed on an open source basis only under The Eclipse Public License (EPL).

AT&T has previously made versions of this software available under the Common Public License (CPL). If you received a copy of the software under that license agreement, you may continue to use and distribute the same version of the software subject to the terms and conditions of the license agreement under which the software was received. However, current versions of the software are now licensed on an open source basis only under The Eclipse Public License (EPL).

For more information on the Eclipse Public License, see the [FAQ](#) If you have any concerns about the what the license means, especially if money is involved, you should contact an intellectual property lawyer.

Eclipse Public License - v 1.0

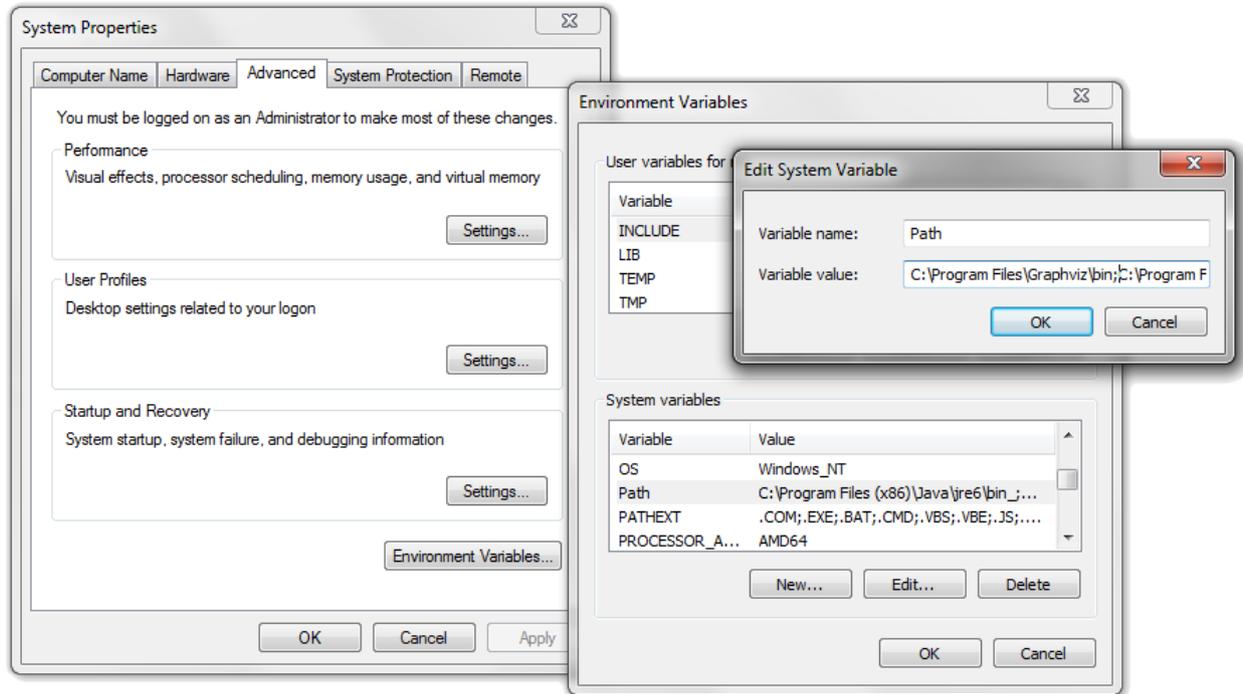
THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

New forum topics

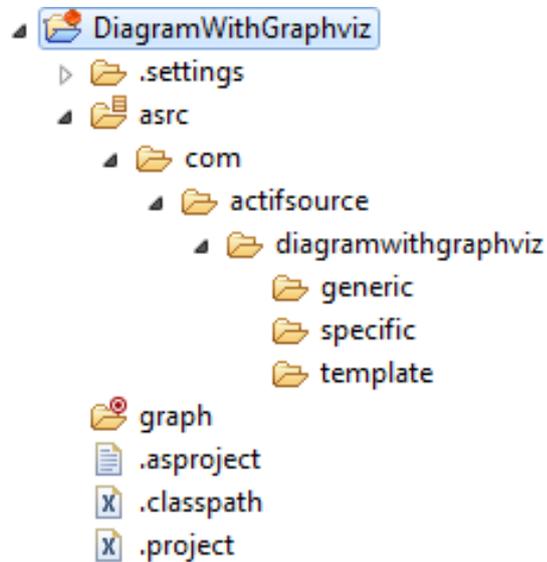
- [B-spline control point format](#)
- [GVEDIT 1.01 - Configure editor font?](#)
- [svg](#)
- [Neato in FreeBSD, PDF file all white](#)
- [Fill Style?](#)

[more](#)

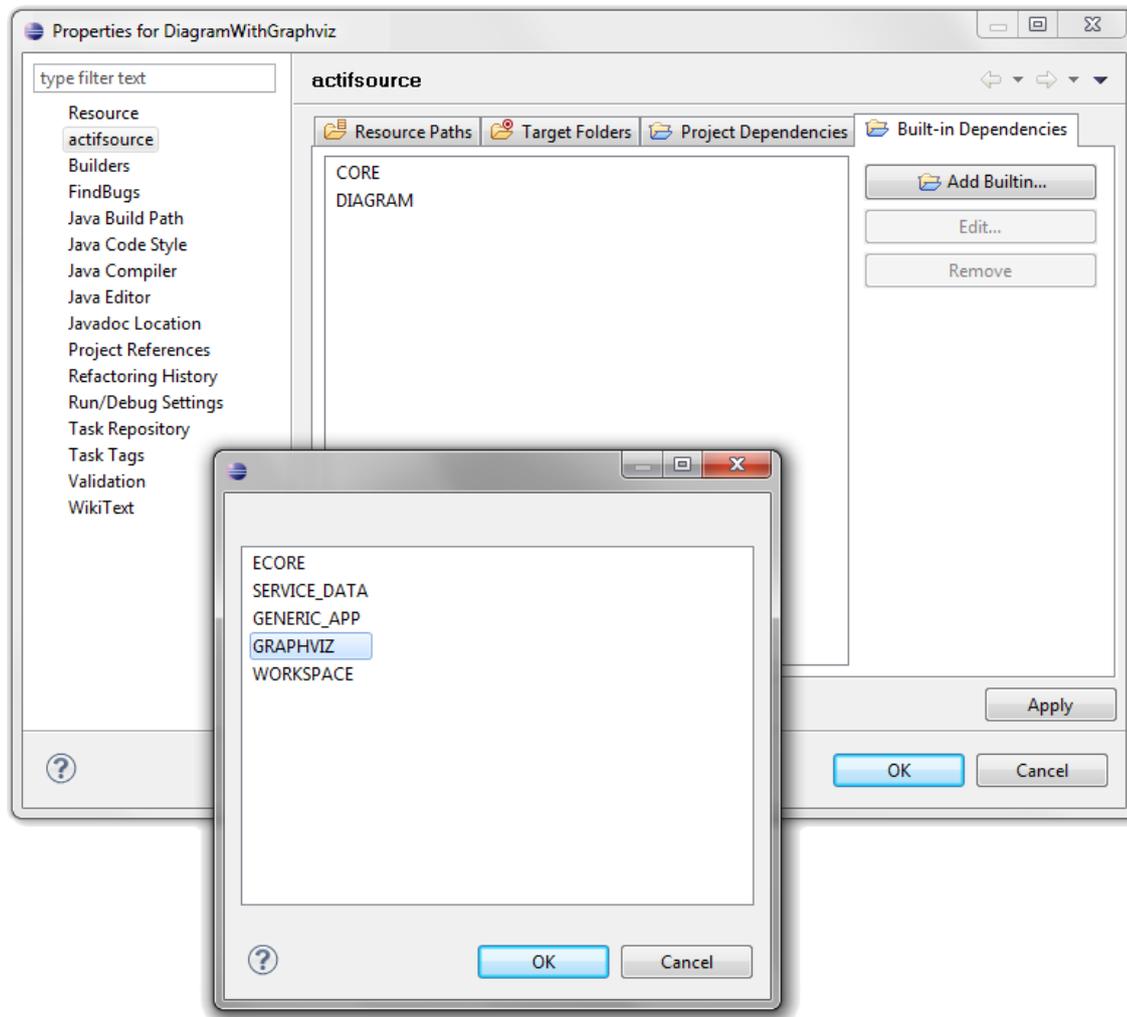
- ↪ Goto “Download” and check the licence agreement (Eclipse Public Licence).
- ↪ Select the executable package provided for your operating system, download the current release and run it.
- ↪ Follow the steps of the installation routine.
- ↪ Add the “bin” folder of the installation path (e.g. C:\Program Files\Graphviz\bin) to your PATH environment variable.



- ↪ Prepare a new **actifsource Project** named `DiagramWithGraphviz` as seen in the *Actifsource Tutorial – Simple Service*
 - Setup the Target Folder *graph*
- ↪ Use the following package structure



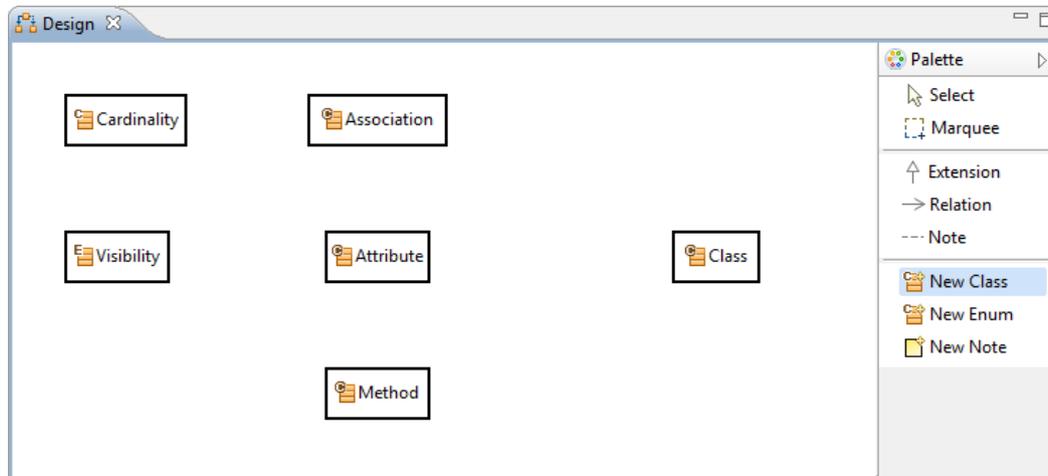
- ① There is an optional Built-in for Graphviz providing a build task that compiles the generated graphviz files into SVG graphics.
- ↵ Open the Project Properties.
- ↵ Select the “actifsource” Settings.
- ↵ On the tab “Built-in Dependencies” add GRAPHVIZ.



Create a UML Model

- ① We need a metamodel that provides those features we want to get displayed on our UML Class Diagram.
- ① The generated diagram will depend only on this model. So, in order to create a Class Diagram that displays the same classes as the generated code, we will have to write a template that generates for every class in the code a class in the Class Diagram.

- ↵ Create a **ClassDiagram** named Metamodel.
- ↵ Add a new **Classes** for Class, Method, Attribute, Association and Cardinality.
- ↵ Add a new **Enum** Visibility.



- ① You don't have to model the UML features explicitly in your real project – but you must be aware how to read those features out of your model. To avoid code duplication, *Functions* as in the *Actifsource Tutorial - Complex Service* might help.

↵ Ctrl+ Left-Click on Cardinality to edit Cardinality in the Resource Editor.

com.actifsource.diagramwithgraphviz.generic

| | | | | | | | | | | | | | | |
|--|---|--------|--------------------------------------|------|------------|----------------|--|--------------------|----------------|-------|----------------|---------------------|--|------------------------|
| typeOf name <i>comment</i> <i>aspect [InitializationAspect]</i> <i>aspect [ResourceValidationAspect]</i> <i>aspect [NameAspect]</i> extends <i>modifier</i> property[1] | ch.actifsource.core.Class Cardinality ch.actifsource.core.Resource | | | | | | | | | | | | | |
| property[2] <i>definesAspect</i> <i>allowRoot</i> <i>classIcon</i> <i>lineColor</i> <i>fillColor</i> <i>shape</i> | <table border="1"> <tr> <td>typeOf</td> <td>ch.actifsource.core.Attribute</td> </tr> <tr> <td>name</td> <td>min</td> </tr> <tr> <td><i>comment</i></td> <td></td> </tr> <tr> <td>subjectCardinality</td> <td>Cardinality0_1</td> </tr> <tr> <td>range</td> <td>IntegerLiteral</td> </tr> <tr> <td><i>defaultValue</i></td> <td></td> </tr> </table> <table border="1"> <tr> <td>max : Attribute</td> </tr> </table> | typeOf | ch.actifsource.core.Attribute | name | min | <i>comment</i> | | subjectCardinality | Cardinality0_1 | range | IntegerLiteral | <i>defaultValue</i> | | max : Attribute |
| typeOf | ch.actifsource.core.Attribute | | | | | | | | | | | | | |
| name | min | | | | | | | | | | | | | |
| <i>comment</i> | | | | | | | | | | | | | | |
| subjectCardinality | Cardinality0_1 | | | | | | | | | | | | | |
| range | IntegerLiteral | | | | | | | | | | | | | |
| <i>defaultValue</i> | | | | | | | | | | | | | | |
| max : Attribute | | | | | | | | | | | | | | |

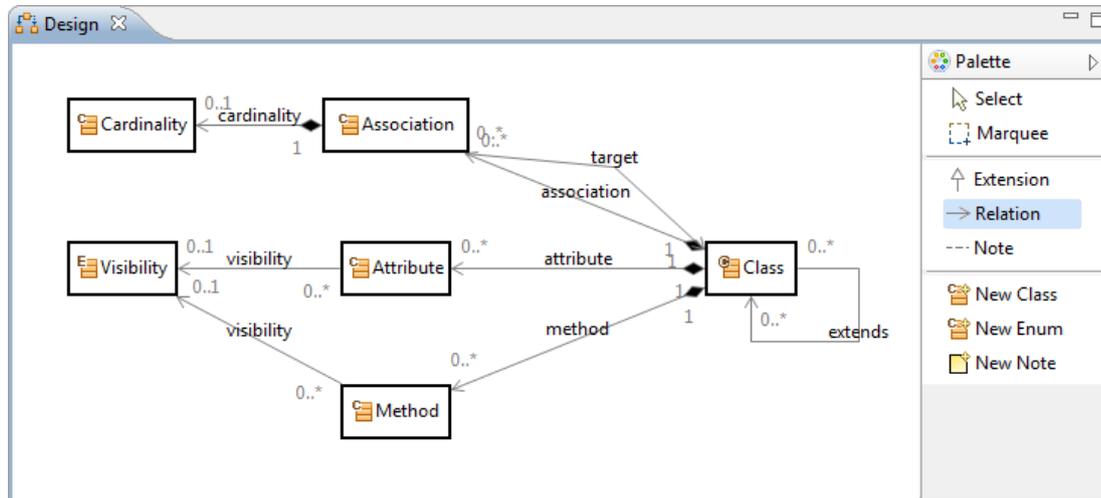
- ↵ Add to IntegerLiteral-Attributes named min and max.
- ↵ Set the (subject-)Cardinality of both attributes to 0..1.
- ↵ Change the supertype extends from NamedResource to Resource, since cardinalities are identified by their minimum and maximum and not by a name.

↩ Ctrl+ Left-Click on the **Enum** Visibility to edit Visibility in the Resource Editor.

The screenshot shows the Resource Editor for an Enum named **Visibility**. The left pane lists various properties and methods, including `typeOf`, `name`, `comment`, `aspect[InitializationAspect]`, `aspect[ResourceValidationAspect]`, `aspect[NameAspect]`, `extends`, `modifier`, `property`, `definesAspect`, `allowRoot`, `classIcon`, `lineColor`, `fillColor`, and `shape`. The right pane shows the Enum's configuration, including the package `ch.actifsource.core.EnumValue` and the `Final` modifier. Below the configuration, four value entries are listed: `public : Visibility`, `protected : Visibility`, `private : Visibility`, and `package : Visibility`. The first entry, `public : Visibility`, is highlighted in orange.

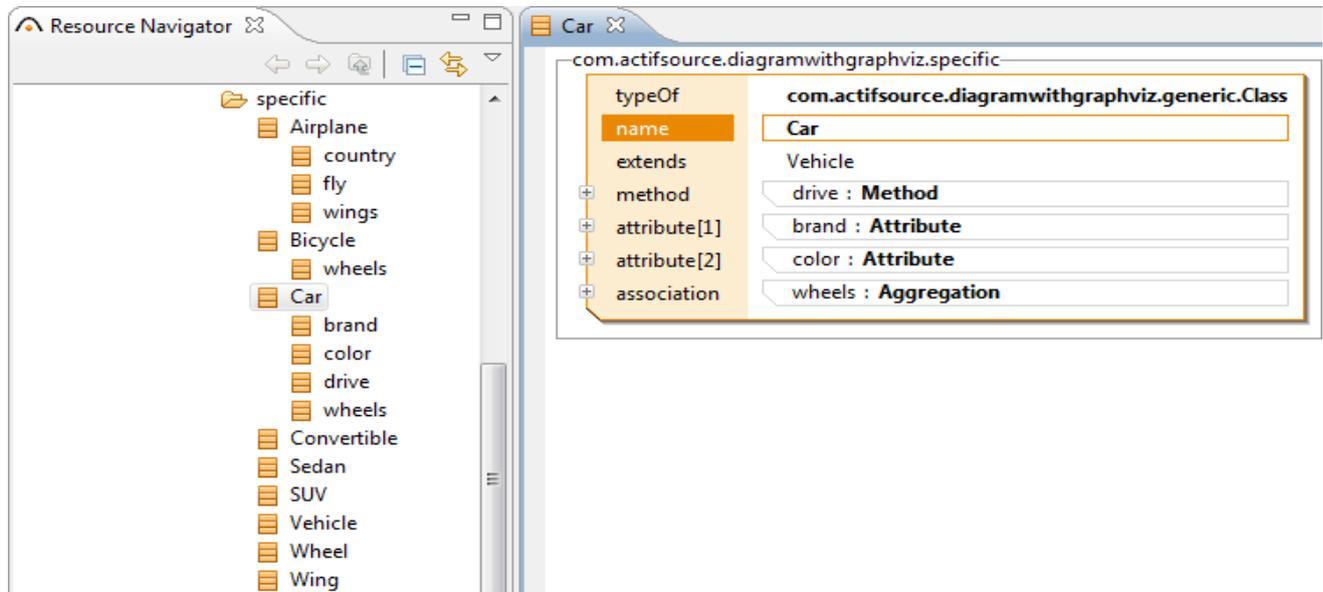
↩ Add the values public, protected, private and package to this **Enum**.

- ↪ Add a **UseRelation** extends from Class to itself.
- ↪ Add a **OwnRelation** from Class to Association.
- ↪ Add a **UseRelation** target back from Association to Class.
- ↪ Set the (subject-)cardinality of target to exactly 1.
- ↪ Add **OwnRelations** from Class to Method and Class to Attribute.



- ↪ Add a **Relation** from Association to Cardinality, let it be an **OwnRelation** for simplicity sake.
- ↪ Add a **UseRelation** from Attribute to Visibility and from Method to Visibility.
- ↪ Set the (subject-)cardinality of the 3 **Relations** to 0..1.

- ① Now let us create a Class Model package `com.actifsource.diagramwithgraphviz.specific` that shows the features we want to look at.
- ↪ For instance, create a Class Vehicle.
- ↪ Create Subclasses Car, Bicycle and Airplane, i.e. Classes having the extends-Relation set to Vehicle.
- ↪ Create Classes Sedan, Convertible, SUV as Subclasses of Car.

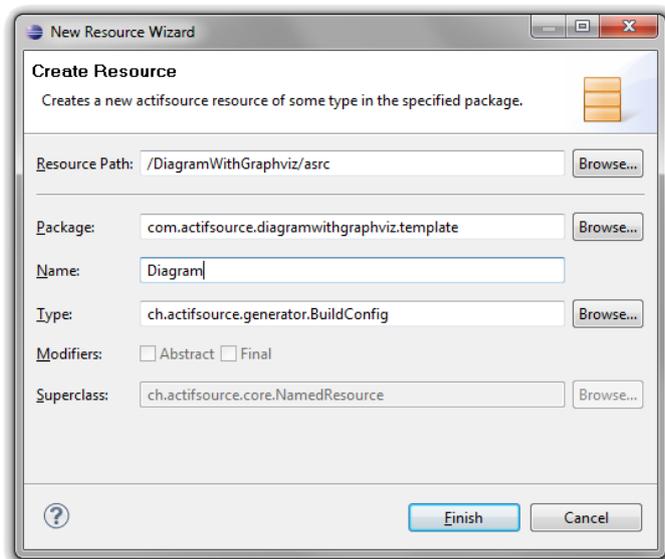
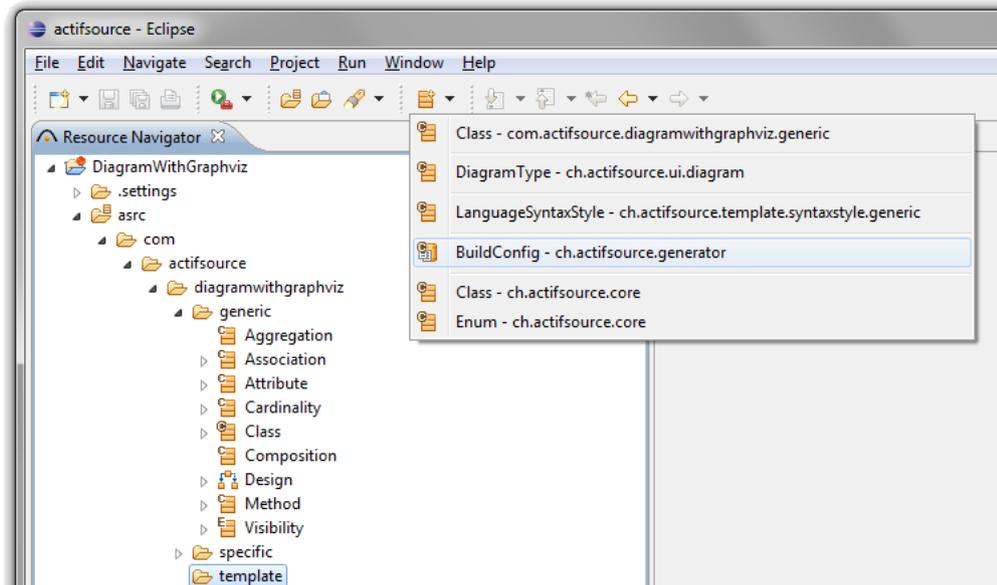


- ↪ Create a Class Wheel and a Class Wing.
- ↪ Add an Association wheels to both Car and also to Bicycle and set the corresponding target to Wheel.
- ↪ Add an Association wings to Airplane which points to Wing.

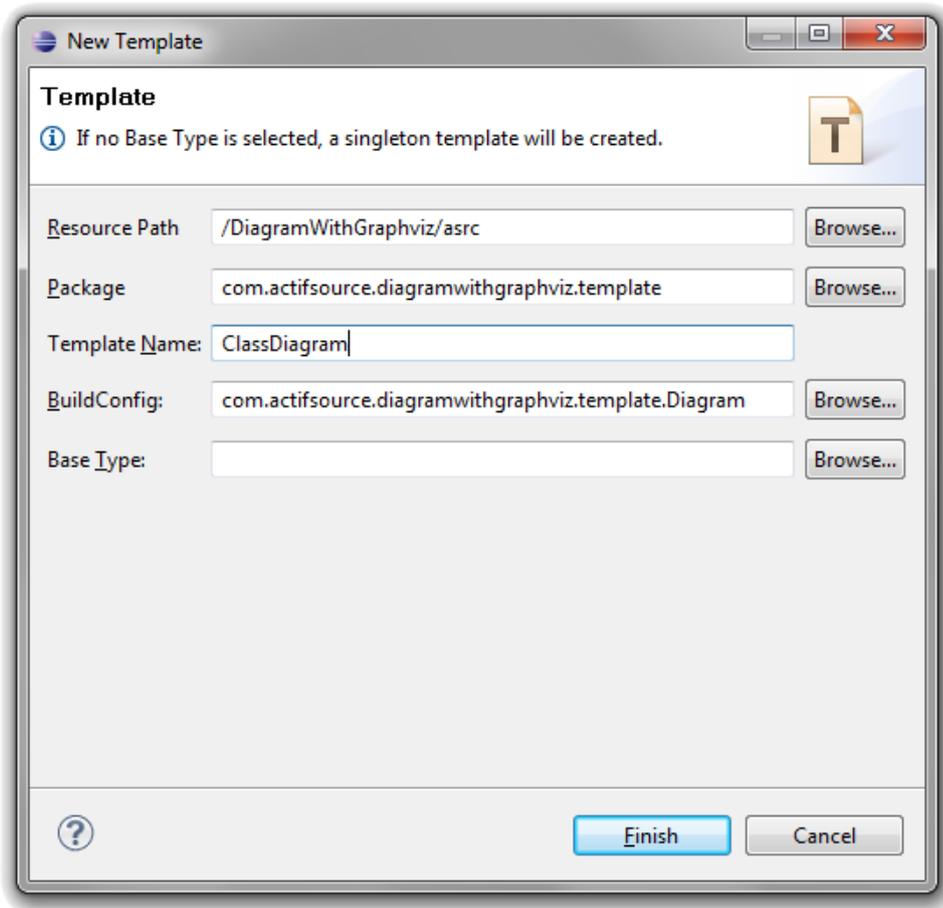
Create a graph template for Graphviz

- ↪ We will generate graph definitions in the Graphviz DOT language and then in layout the graph to define our UML Class Diagram using the Graphviz DOT command. You can find the documentation of the DOT language on the Graphviz homepage.
- ↪ Graphviz provides several output formats: raster graphics like PNG, or vector graphics like PostScript or SVG. We will prefer SVG as output format since it can easily be post-processed, and the graphics can be embedded in HTML Pages and contain tooltips and hyperlinks.

- ↪ Select the package `com.actifsource.diagramwithgraphviz.template`.
- ↪ Create a new **Resource** of type **BuildConfig** and name it Diagram.

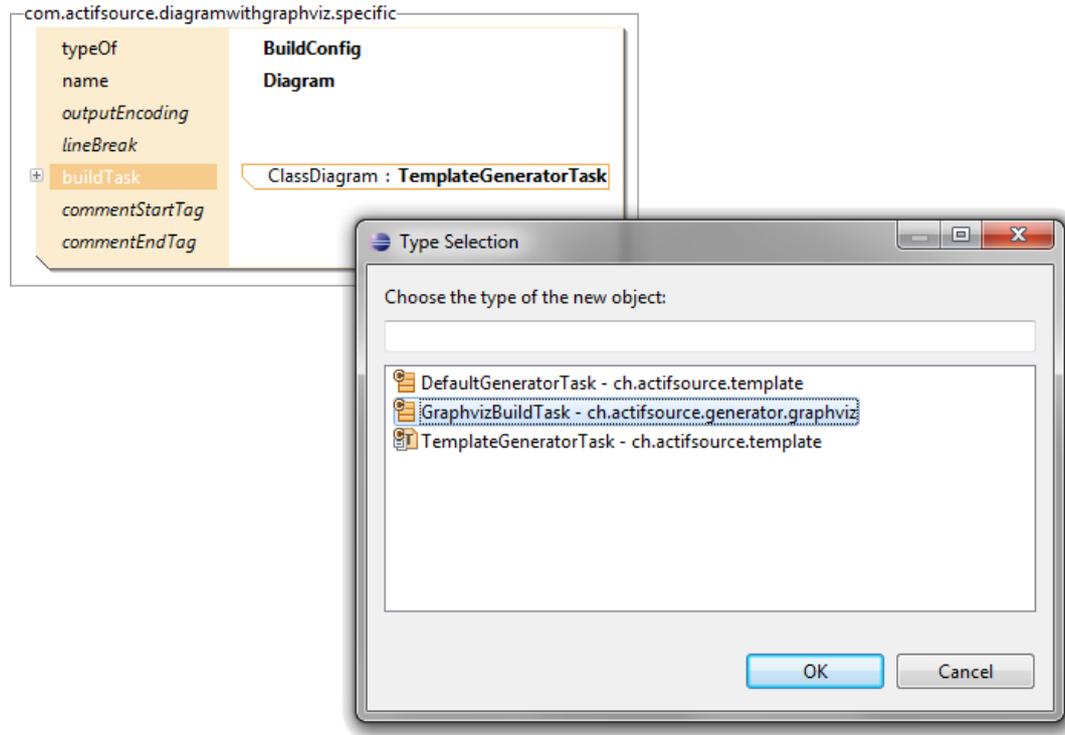


- ↪ Create a new **Template** named ClassDiagram.



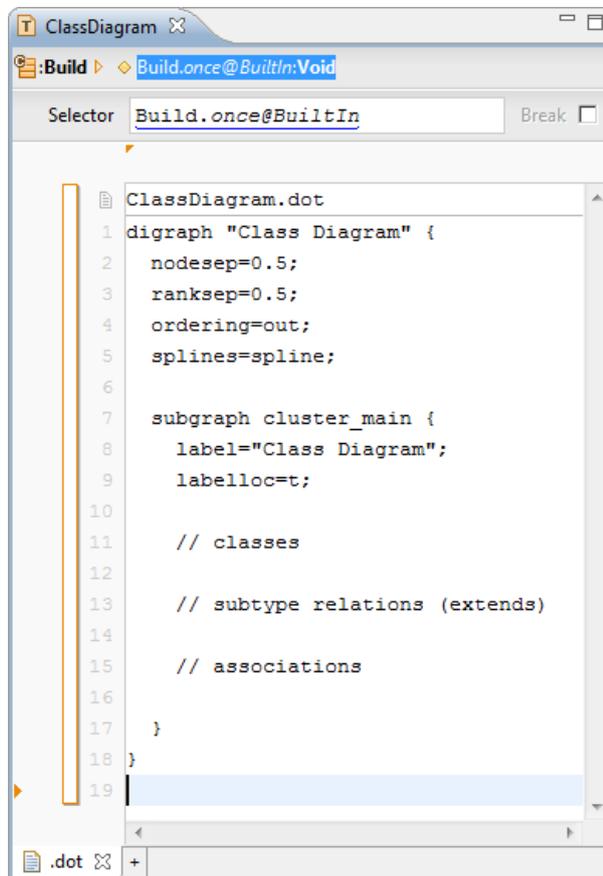
- ① The **Template** ClassDiagram is now placed into the selected **BuildConfig** Diagram.

- ↩ Select the ClassDiagram **TemplateGeneratorTask**.
- ↩ Insert after the selection a **GraphvizBuildTask** with Ctrl+Enter.



- ① The **GraphvizBuildTask** will effect that every File with *File Extension .dot* in the **Target Folder** and its subdirectories is processed with the Graphviz *DOT command* and an SVG File with the same name but *File Extension .svg* is written.
- ① Note that the **GraphvizBuildTask** has to be executed after all DOT Files have been generated. Therefore it is always the last **BuildTask** in the list. If you add new **Templates** to the **BuildConfig**, move the **GraphvizBuildTask** to the end (by pressing Alt+PgDn).

- ① If you are interested in the full capabilities of Graphviz, consult the documentation on www.graphviz.org.
- ↪ Open the **ClassDiagram Template** with the **Template Editor**.
- ↪ Create a directed graph (*digraph*) with the following attributes:
 - Default *rankdir* (i.e. *rankdir=TB*: top-bottom), so unconnected nodes (i.e. Classes) will appear on top.
 - *ordering=out*: Preserve the order of the outgoing edges (i.e. left sibling before right sibling, etc.)
 - *nodesep=0.5; ranksep=0.5*: Space among nodes of same rank, Space between ranks
 - *splines=spline*: Draw the edges (i.e. Associations and extends-Relations) as cubic splines.
- ↪ Place nothing in it but a single cluster, that will be drawn with a frame around
 - Label this cluster `Class Diagram`.



The screenshot shows a window titled "ClassDiagram" with a "Selector" field containing "Build.once@BuiltIn" and a "Break" checkbox. The main area displays the content of "ClassDiagram.dot" with line numbers 1 through 19. The code defines a directed graph with specific styling and a subgraph for the main cluster.

```
1 digraph "Class Diagram" {
2   nodesep=0.5;
3   ranksep=0.5;
4   ordering=out;
5   splines=spline;
6
7   subgraph cluster_main {
8     label="Class Diagram";
9     labelloc=t;
10
11    // classes
12
13    // subtype relations (extends)
14
15    // associations
16
17  }
18 }
19
```

- ① The main elements of a DOT file are *graph*, *node* and *edge*.
 - *graph* Definition: `graph GraphID { ... }`
 - *node* Definition: `NodeID [attr1=value1, ... , attrn=valuen];`
 - *edge* Definition: `NodeIDtail -> NodeIDhead [attr1=value1, ... , attrn=valuen];`
- ① The default values of the attributes can be set with special statements:
 - `node [attr1=value1, ... , attrn=valuen];`
 - `edge [attr1=value1, ... , attrn=valuen];`

```
// classes
node [shape=record, style=filled, fillcolor="lightyellow"];

// subtype relations (extends)
edge [dir=back, arrowtail=empty, headport=n];

// associations
edge [dir=normal, arrowhead=vee, headport="", labeldistance=2.5];
```

- ↗ Set the default attributes for Class: Set its node *shape* to *record* and set fill color to *lightyellow*.
- ↗ Set the default attributes for the extends Relation: The arrow should point from bottom to top (*dir=back*) and have an *empty* arrow head.
- ↗ Set the default attributes for the Associations: The arrow head should have a V-shape (*vee*).

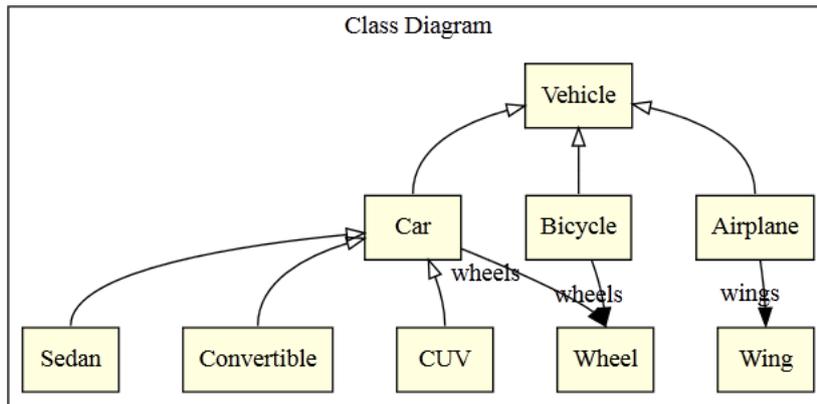
- ① Let us now insert line contexts for the UML artifacts we discussed before:

```

11 // classes
12 node [shape=record, style=filled, fillcolor="lightyellow"];
13 Class.name [label="Class.name"];
14
15 // subtype relations (extends)
16 edge [dir=back, arrowtail=empty, headport=n];
17 Class.name -> Class[2].name;
18
19 // associations
20 edge [dir=normal, arrowtail=vee, labeldistance=2];
21 Class.name -> Association.target.name [taillabel="Association.name"];
22 }
23 }
    
```

- ↪ Add a line context for all Classes.
- ① Be sure you choose com.actifsource.diagramwithgraphviz.generic.Class and not ch.actifsource.core.Class!
- ↪ Add a line context for all Classes with a nested line context which iterates over Class.extends.
- ① The Class of the outer (hidden) context is named in the template editor with Class[2], the Class of the inner context (so the base class) is named with Class.
- ↪ Add an edge statement back from Class to Class[2].
- ↪ Add a line context for all Classes with a nested line context for all Associations.
- ↪ Add an edge statement from Class to the Association.target labeled with the Association.name.

① The resulting graph will look like this:



- ① We want to subdivide the class nodes to display the methods and attributes. For this feature, it will help we did not choose a simple *rect* shape for class nodes, but a *record* shape: With pipe characters “|” we can subdivide node shapes and with curly brackets “{” and “}” swap the direction of division from left-right to top-bottom and vice versa.

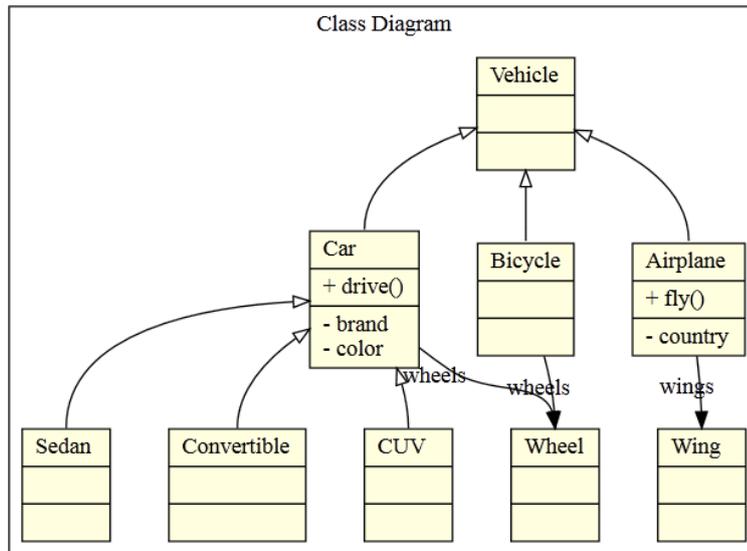
```
// classes
node [shape=record, style=filled, fillcolor="lightyellow"];
Class.name [label="{Class.name}\1|\1"}];
```

- ↪ Change the label of the class node, so the node is partitioned vertically into 3 fields with the Class.name in the first field, all fields left-justified.
- ① It is not possible to have column contexts nested.
- ↪ Therefore split the label into three lines using the backslash “\” at the line end.
- ↪ Add line contexts to iterate over the Methods of a Class and over the Attributes of a Class.

```
11 // classes
12 node [shape=record, style=filled, fillcolor="lightyellow"];
13 Class.name [label="{Class.name}\1|\1"}];
14 Method.name ()\1\
15 |\
16 Attribute.name\1\
17 }];
```

Attribute.visibility:Visibility

① The resulting diagram will now look like this:



- ① We need the visibility symbols in our model in order to display it in the graph. Or, we need a Java function (see *Actifsource Tutorial – Complex Service*) that maps the Visibility elements in the model onto the UML symbols.
- ↪ Open the **Enum** Visibility in the **Resource Editor**.
- ↪ Add a **StringLiteral Property** umlSymbol, with (subject-)cardinality exactly 1.
- ↪ Set the umlSymbol of the enumeration values to the respective UML symbols: +, #, -, ~.

com.actifsource.diagramwithgraphviz.generic

| | | | | | | | |
|----------------------------------|--|--------|-------------------|------|---------------|-----------|---|
| typeOf | Enum | | | | | | |
| name | Visibility | | | | | | |
| comment | | | | | | | |
| aspect[InitializationAspect] | | | | | | | |
| aspect[ResourceValidationAspect] | | | | | | | |
| aspect[NameAspect] | | | | | | | |
| extends | ch.actifsource.core.EnumValue | | | | | | |
| modifier | Final | | | | | | |
| property | umlSymbol : Attribute | | | | | | |
| definesAspect | | | | | | | |
| allowRoot | | | | | | | |
| classIcon | | | | | | | |
| lineColor | | | | | | | |
| fillColor | | | | | | | |
| shape | | | | | | | |
| value[1] | <table border="1"> <tr> <td>typeOf</td> <td>Visibility</td> </tr> <tr> <td>name</td> <td>public</td> </tr> <tr> <td>umlSymbol</td> <td>+</td> </tr> </table> | typeOf | Visibility | name | public | umlSymbol | + |
| typeOf | Visibility | | | | | | |
| name | public | | | | | | |
| umlSymbol | + | | | | | | |
| value[2] | protected : Visibility | | | | | | |
| value[3] | private : Visibility | | | | | | |
| value[4] | package : Visibility | | | | | | |

- ↵ In the template, insert a new *Column Context* in the Context which iterates over all Methods.
- ↵ Select in this *Context* the Method.visibility.
- ↵ In this context Insert Visibility.umlSymbol.
- ↵ Repeat the 3 steps for Attribute.visibility.

```
11 // classes
12 node [shape=record, style=filled, fillcolor="lightyellow"];
13 Class.name [label="{Class.name}\1\
14 Visibility.umlSymbol Method.name ()\1\
15 |\
16 Visibility.umlSymbol Attribute.name\1\
17 }"];
Attribute.visibility:Visibility
```

- ① You can eliminate the empty record fields in the class nodes of the graph by checking if there are any methods or attributes:

```

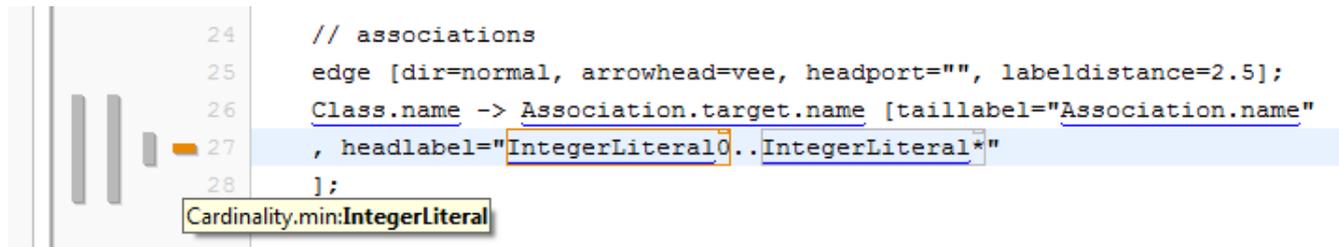
11 // classes
12 node [shape=record, style=filled, fillcolor="lightyellow"];
13 Class.name [label="{Class.name}\|\
14 Visibility.umlSymbol Method.name ()\|\
15 |\
16 Visibility.umlSymbol Attribute.name\|\
17 }"];
18 Class.name [label="Class.name"];

```

Class.method union Class.attribute:NamedResource

- ↵ Add a **Line Context** inside the context which iterates over all Classes.
- ↵ Set the **Selector** to Class.method union Class.attribute.
- ① This will iterate over the set union of the Elements in Class.method and those in Class.attribute.
- ↵ Choose *First* (Alt + 1), so that this Context is entered at most once.
- ↵ Add a line that generates only a simple rect for a Class.
- ↵ Choose *Empty* (Alt + 5), so that this line is evaluated if the *Context* "Class.method union Class.attribute" is empty.

① We did not cover cardinalities so far.



```
24 // associations
25 edge [dir=normal, arrowhead=vee, headport="", labeldistance=2.5];
26 Class.name -> Association.target.name [taillabel="Association.name"
27 , headlabel="IntegerLiteral0..IntegerLiteral*"
28 ];
```

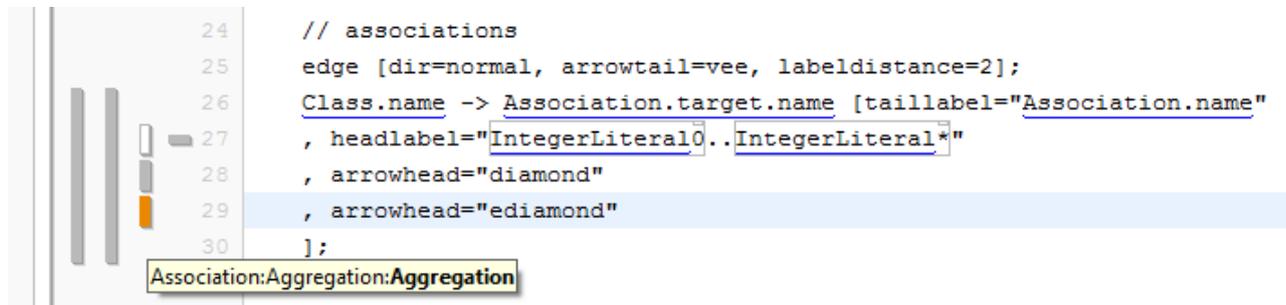
Cardinality.min:IntegerLiteral

↵ Add a *Line Context* that selects the Association's Cardinality.

↵ In this Context, add a *headlabel*, that contains the min and max Cardinality:

- In the label insert a *Column Context* for Cardinality.min and one for Cardinality.max.
- For empty Cardinality.min context (Alt + 5), insert 0.
- For empty Cardinality.max context (Alt + 5), insert *.

- ① In UML, associations are marked with a diamond, if they are aggregations or compositions.
- ↪ Add the **Classes** Aggregation and Composition which are subclasses of Association.
- ↪ Edit the **typeOf Property** of the **Relations** wheels and wings and replace Association by Aggregation and Composition respectively.



```
24 // associations
25 edge [dir=normal, arrowtail=vee, labeldistance=2];
26 Class.name -> Association.target.name [taillabel="Association.name"
27 , headlabel="IntegerLiteral0..IntegerLiteral*"
28 , arrowhead="diamond"
29 , arrowhead="ediamond"
30 ];
```

Association:Aggregation:Aggregation

- ↪ Add a *Line Context*
- ↪ In the **Selector**, cast Association to Aggregation and Composition respectively. The Line Contexts will be entered then only if it is an Aggregation or Composition.
- ↪ Set the *arrowhead* attribute to *diamond* and to *ediamond* (=empty diamond).

① The resulting graph will look now similar to this one:

