Documentation of the package pgfuml

May 4, 2009

Fabien Dagnat [1]

This package contains various macros to make it possible to create pgf (`texdoc pgf` for a documentation of pgf) pictures representing UML diagrams.

1 Conventions on this documentation

All portions of code are presented with a light blue background. If the result of the code portion is included, it has a light orange background.

Each macro and environment defined in this package is configurable by options. These options adopt the now classic notation `keyval`. That is positioning macro option is done by providing pairs of a key (the option's name) and a value. Here are two example:

```
\commande[opt1,opt2=unevaleur]{...}

\begin{environnement}[opt1,opt2=unevaleur]
  ...
\end{environnement}
```

In this document, each option is presented with its name, its type, its default value and its description. As latex is not a typed language, the type is only informative about the form of value you can pass to the option. For boolean, the value is optional and if it is absent, it is considered as being true.

2 Classes

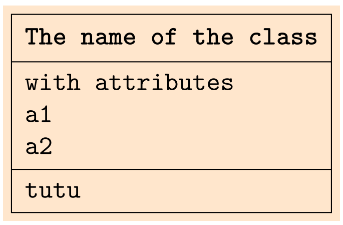
A UML class is created by the macro `\umlclass`. This macro must be called in a `tikzpicture` environment. This macro has two arguments: a list of options (optional) and a body. The list

1. `fabien.dagnat@telecom-bretagne.eu`

of options (described below) define properties of the class such as its name, stereotypes… The body is the text corresponding to the content of the class (its attributes and methods).

When displayed, the class is separated in parts. There may be from one to four parts. The first part contains the name of the class and its stereotypes, the second part consists in the attributes, the third of the methods and the fourth is an extra part not commonly used in diagrams. They are respectivelly named `text`, `attribute`, `method` and `extra`.

Here is a first example:



```
\begin{tikzpicture}[font=\footnotesize]
  \umlclass[name=The name of the class,label=label](0,0){
      with attributes\\
      a1\\
      a2
      \separator
      tutu
    }
\end{tikzpicture}
```

In the body of a class, the various parts must be separated by a macro `\separator`. This macro has an optional parameter describing the part concerned. The default value of this parameter is `method`. See figure 1 on page 4, for an example of a class containing several separators.

The following options are supported by the macro `\umlclass`:

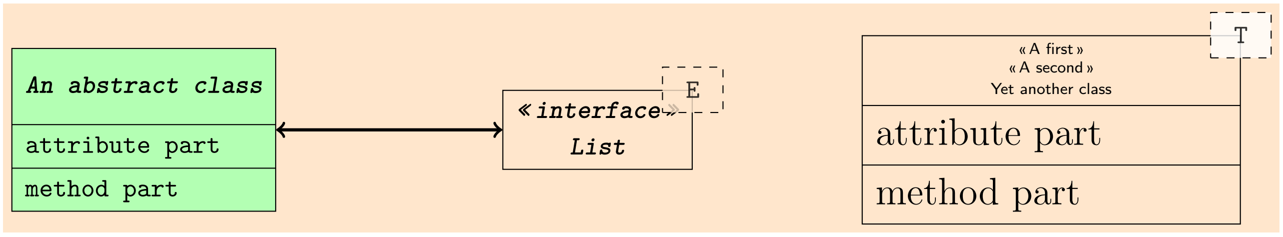
- `name` (**string**, intial value: *empty*)
  The name of the class.
- `abstract` (**boolean**, intial value: *false*)
  A boolean indicating whether the class is abstract or not. When true, `\itshape` is added to the font for the name part.
- `interface` (**boolean**, intial value: *false*)
  A boolean indicating whether the class is an interface or not. When true, `\itshape` is added to the font for the name part. Furthermore, an interface stereotype is added in front of the other stereotypes.
- `stereotype` (**text**, intial value: *empty*)
  The stereotype text for the class. It can support line break using the `\tabularnewline` command. A macro `\stereotype` is provided to add the « guillemets ». It is used on anything:

  « TUTU »

  ```
  \stereotype{TUTU}
  ```
- `param` (**string**, intial value: *empty*)
  The name of the generic parameter.
- `label` (**string**, intial value: *pgfuml@class*)
  The label used for the class. It is a usual `tikz` label and may be used to reference the class in any command accepting such a label.
- `parts` (**number (1-4)**, intial value: *1*)
  The number of displayed parts for the class. If the body is non empty and the parts is not given, parts is automatically set to 3. For other values, the option is required. Figure 1 on page 4 contains a code that shows how is displayed a class when one use the parts option.
- `name font` (**macro**, intial value: *`\ttfamily\bfseries`*)
  The macro is used as a font specification for the name part of the class. By default, a typewriter bold font is used to typeset the name part of the class. Beware, that some font does not support bold typewriter face and use only typewriter face.

– body font (**macro**, intial value: *\ttfamily*)
  The macro is used as a font specification for the body part of the class. By default, a typewriter font is used to typeset the body part of the class.
– fill (**color**, intial value: *none*)
  If fill is not none and is a colour, this colour is used to fill the class.
– param fill (**color**, intial value: *white*)
  If fill is not none and is a colour, this colour is used to fill the generic parameter.
– width (**length**, intial value: *2cm*)
  This length specifies the minimum width of the class. Beware that you **must** provide a length (a number and a unit).
– name height (**length**, intial value: *1.5em*)
  This length specifies the minimum height of the name part of the class. Beware that you **must** provide a length (a number and a unit).

A more complex example can be proposed to show the use of the various options:

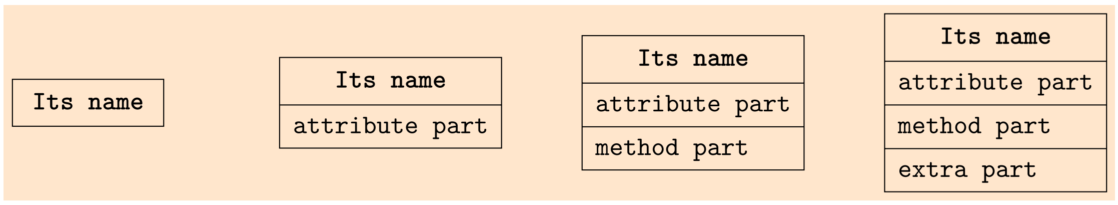


```
\begin{tikzpicture}[font=\footnotesize]
  \umlclass[name=An abstract class,abstract,label=l,fill=green!30,name height=1cm](0,0){
    attribute part
    \separator
    method part
  }
  \umlclass[name=List,interface,label=its label,param=E,param fill=graphicbackground](6,0){}
  \draw[very thick,<->] (l) -- (its label);
  \umlclass[name=Yet another class,param=T,width=5cm,
           stereotype={\stereotype{A first} \tabularnewline \stereotype{A second}},
           body font=\rm\large,name font=\sf\tiny](12,0){
    attribute part
    \separator
    method part
  }
\end{tikzpicture}
```

3 Relations

The second basic macro for UML class diagram is the notion of relation. It specifies a relation between two classes that is represented by a line. In UML, several relations may be used. In the pgfuml package, a common macro is used to typeset all relations but this macro is not aimed to be used directly. The user is invited to use the more specific relations that are defined: associations, inheritance, . . .

The first subsection introduces the relation macro and its options and then all specific relations are presented each in its subsection.

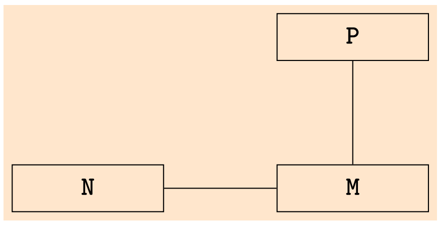


```
\begin{tikzpicture}[font=\footnotesize]
  \newcommand{\myclass}[2]{ \umlclass[name=Its name,label=its label,parts=#1](#2){
      attribute part \separator method part \separator[extra] extra part }
  }
  \foreach \parts/\coord in {1/{0,0},2/{4,0},3/{8,0},4/{12,0}} { \myclass{\parts}{\coord} }
\end{tikzpicture}
```

Figure 1: Use of parts option

3.1 Relations

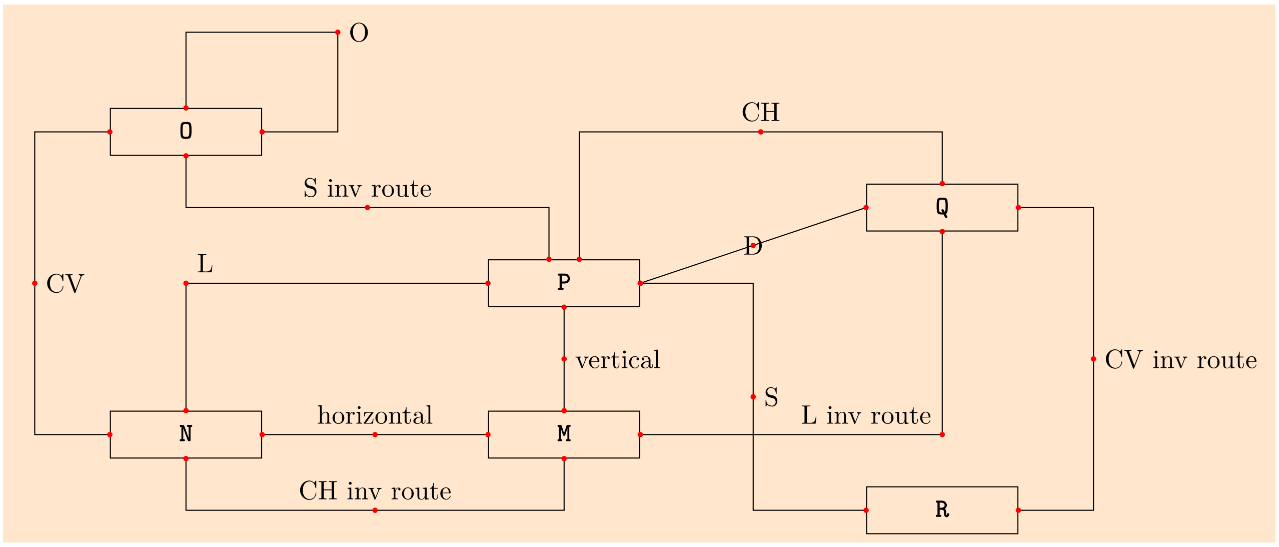
The macro `\umlrelation` takes an optional parameter describing options to use and two mandatory parameters that must be given between parenthesis. The two arguments may be any `tikz` coordinate specification, but are aimed to be the labels of the classes to relate. The following example demonstrates the basic use of the notion of relation:



```
\begin{tikzpicture}[font=\footnotesize]
  \umlclass[name=N,label=n](0,0){}
  \umlclass[name=M,label=m](3.5,0){}
  \umlclass[name=P,label=p](3.5,2){}
  \umlrelation(m)(n)
  \umlrelation(m)(p)
\end{tikzpicture}
```

Relations can be of the 8 following kinds (see figure 2 on the following page), the 3 firsts are determined automatically while the 5 lasts need an option specification (see below):

1. *vertical*, | if the two linked classes have less than 1mm of difference between their x coordinate (`P` and `M` in the previous example).
2. *horizontal*, — if the two linked classes have less than 1mm of difference between their y coordinate (`N` and `M` in the previous example).
3. *L*, ┌─ in other cases classes are linked by an L.
4. *O*, is a relation between a class and itself. It may appear on any of the four corners of the class.
5. *D*, also known as direct relations corresponds to a direct line whatever the coordinates of its extremities.
6. *S*, _┌ classes may also be linked by an S going through the middle of the segment joining them.
7. *C vertical*, ⊏ classes may be linked by a vertical C going through a point that is at distance `C sep` (see `C sep` option explanation) of the middle of the segment joining them. Notice that the classes need not to be aligned.
8. *C horizontal*, ┌─┐ lastlty classes may be linked by an horizontal C going through a point that is at distance `C sep` (see `C sep` option explanation) of the middle of the segment joining them. Notice that the classes need not to be aligned.



```
\pgfkeys{/tikz/point/.style={fill=red,circle,inner sep=0pt,minimum size=2pt}}
\newcommand{\drawPoints}[2]{\draw (start) node[point] {}; \draw (middle) node[#1] {#2};
  \draw (middle) node[point] {};\draw (end) node[point] {};}
\begin{tikzpicture}[font=\footnotesize]
  \umlclass[name=N,label=n](0,0){}    \umlclass[name=O,label=o](0,4){}
  \umlclass[name=M,label=m](5,0){}    \umlclass[name=P,label=p](5,2){}
  \umlclass[name=Q,label=q](10,3){}   \umlclass[name=R,label=r](10,-1){}
  \umlrelation(m)(n)                               \drawPoints{above}{horizontal}
  \umlrelation(m)(p)                               \drawPoints{right}{vertical}
  \umlrelation(n)(p)                               \drawPoints{above right}{L}
  \umlrelation[inv route](m)(q)                    \drawPoints{above left}{L inv route}
  \umlrelation[CV](n)(o)                           \drawPoints{right}{CV}
  \umlrelation[CV,inv route](r)(q)                 \drawPoints{right}{CV inv route}
  \umlrelation[CH,C sep=1.5cm,start shift=2mm](p)(q) \drawPoints{above}{CH}
  \umlrelation[CH,C sep=1cm,inv route](n)(m)       \drawPoints{above}{CH inv route}
  \umlrelation[S](p)(r)                            \drawPoints{right}{S}
  \umlrelation[S,inv route,start shift=-2mm](p)(o)   \drawPoints{above}{S inv route}
  \umlrelation[D](p)(q)                            \drawPoints{}{D}
  \umlrelation[O](o)(o)                            \drawPoints{right}{O}
\end{tikzpicture}
```

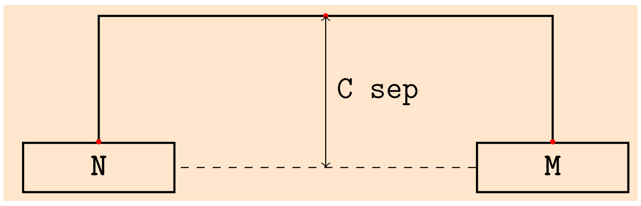
Figure 2: The various kind of relations

O relation and shifting options require that the corresponding parameter reference points must correspond to a rectangular shape[2]. If you do not use shift or O option, you can use any `tikz` coordinate as parameters to the `\umlrelation` macro.

The following options are supported by the `\umlrelation` macro:

- `D` (**boolean**, intial value: *false*)
  A boolean indicating whether the relation is a direct line whatever the coordinates of its extremities.
- `O` (**boolean**, intial value: *false*)
  A boolean indicating whether the relation is a self relation. Beware that, this kind of relation is build to be used between a class and itself, this means that both reference points must be equal and be a class label. In other cases, the result is unspecified.
- `O anchor` (**tikz position**, intial value: *north east*)
  A tikz position indicating the point used to determine the cycle relation position. Beware, that the option is meant to be used with the four corners (north east, north west, south east and south west). In other cases, the result is unspecified.
- `O sep` (**length**, intial value: *1cm*)
  The distance the point determining the cycle relation is from its anchor. In fact, this point will have the coordinate $anchor + (O\ sep, O\ sep)$.
- `S` (**boolean**, intial value: *false*)
  A boolean indicating whether the relation has an S form (_|‾).
- `CV` (**boolean**, intial value: *false*)
  A boolean indicating whether the relation has an C vertical form (⊏).
- `CH` (**boolean**, intial value: *false*)
  A boolean indicating whether the relation has an C horizontal form (⊓).
- `C sep` (**length**, intial value: *2cm*)
  For C relation, the distance between the middle of the two classes and the middle of the relation (see figure 3 on the next page).
- `inv route` (**boolean**, intial value: *false*)
  This boolean controls the path of the relation in case of an L (_| instead of |‾), a S (‾|_ instead of _|‾) or a C (⊔ and ⊐ instead of ⊓ and ⊏) kind of relation.
- `start shift` (**length**, intial value: *0mm*)
- `end shift` (**length**, intial value: *0mm*)
- `shift` (**length**, intial value: *0mm*)
  Extremities shifting specifications respectively for the start, the end and both sides. When used the relation start or end point is shifted by the given value staying on the shape border. This means that on west and east side, the shift is a y shift and on north and south border, it is a x shift. The given length may be negative.
- `arrow` (**tikz arrow**, intial value: *empty*)
  This option enable to set the arrow type of the relation. Any `tikz` arrow type specification may be used and we refer the interested reader to the `tikz` documentation.
- `dashed` (**boolean**, intial value: *false*)
  If true the relation is typeset dashed.
- `name pos` (**number between 0 and 1**, intial value: *0.5*)
  The position of the node named `name` along the segment from start to end of the relation. This option is meant mainly to be used indirectly in associations and implementations.

2. In fact, they must have a north east and a south west anchor.



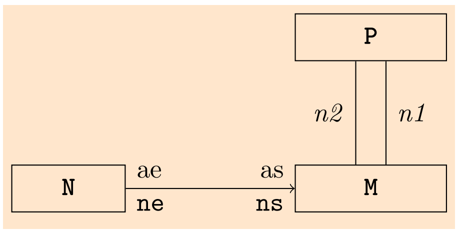
```
\pgfkeys{/tikz/point/.style={fill=red,circle,inner sep=0pt,minimum size=2pt}}
\begin{tikzpicture}
  \begin{scope}[thick]
    \umlclass[name=N,label=n](0,0){}
    \umlclass[name=M,label=m](6,0){}
    \umlrelation[CH](m)(n)
  \end{scope}
  \draw (start) node[point] {}; \draw (end) node[point] {};
  \draw (middle) node[point] {};
  \draw[dashed] (m) -- node[coordinate] (middlemn) {} (n);
  \draw[<->] (middle) -- node[right] {\texttt{C sep}} (middlemn);
\end{tikzpicture}
```

Figure 3: A CH relation with `C sep` options explained

After the relation macro execution four temporary[3] points are defined: `start`, `end`, `middle` and `name`. They correspond intuitively to the two extremities and the (logical) middle and are marked by a red point in the figure 2 on page 5.

3.2 Associations

The first relation macro aimed to be used by user for UML class diagram is the notion of association. It specifies a relation between two classes that support to be annotated. The possible annotations are a name and cardinality and naming of the two extremities. Furthermore, an association may be directed (includes a navigation symbol), a composition or an aggregation. The macro `\umlasso` has the same form of the relation macro as demonstrated by the following example:



```
\begin{tikzpicture}[font=\footnotesize]
  \umlclass[name=N,label=n,width=1.5cm](0,0){}
  \umlclass[name=M,label=m](4,0){}
  \umlclass[name=P,label=p](4,2){}
  \umlasso[arity start=as,arity end=ae,name start=ns,
           name end=ne,nav start](m)(n)
  \umlasso[name=n1,shift=2mm](m)(p)
  \umlasso[name=n2,name anchor=left,shift=-2mm](m)(p)
\end{tikzpicture}
```

Associations are relations, in the sense that they will be represented as relations and therefore have all the options of the relation macro (except for arrows that are specified using specific options). And, in fact, one have the 7 kinds of associations.

Similarly, to shifting and O relation, extremity annotations placement is done automatically using the received arguments as class labels. If no annotation, no shifting and no O is given one may

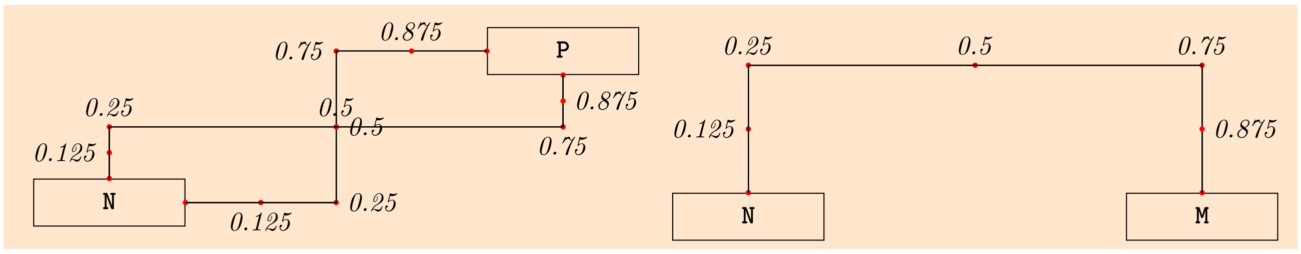
3. As every relation macro uses these points, you should save them if you need them. Saving the point `p` can be done by putting a named coordinate node at its position: `\draw (p) node[coordinate] (name) {};` for example.

use any `tikz` coordinate as parameters to the `\umlasso` macro.

In the specific case of direct associations (option `D`), it is not possible to find easily good positioning of the annotations. In this case, they are not typeset and this has to be done by hands. . .

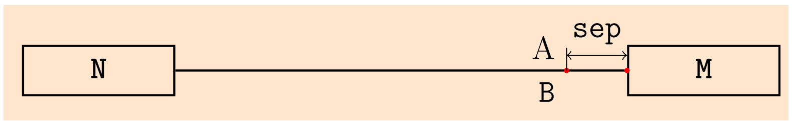
The following options are supported by the `\umlasso` macro:

- `name` (**string**, intial value: *empty*)
  The name of the association.
- `name pos` (**number between 0 and 1**, intial value: *0.5*)
  The position of the name along the segment from start to end of the association. Figure 4 on the next page contains a description of some position for two S association and a C horizontal association.
- `name anchor` (**tikz position**, intial value: *depends on the association*)
  The position of the name of the association with respect to the middle (in the logical sense) of the association. The package tries to find automatically a good position (`above` for "horizontal" associations, `right` for vertical one and none for direct one). But if this placement does not match your will, you can set it by hand. In figure 2 on page 5 the names are placed at the position association macro would have placed them (it can serve as reference).
- `arity start` (**string**, intial value: *empty*)
- `arity end` (**string**, intial value: *empty*)
  The cardinality of the extremities, either the start side or the end side. The package impose no form on the cardinality specification. I advise to use `\textasteriskcentered` ($*$) rather than * when specifying cardinalities.
- `name start` (**string**, intial value: *empty*)
- `name end` (**string**, intial value: *empty*)
  The name of the extremities, either the start side or the end side. The package impose no form on the cardinality specification.
- `sep` (**length**, intial value: *0mm*)
  The length used to separate the extremities from their annotations (see figure 5 on page 10).
- `D` (**boolean**, intial value: *false*)
  A boolean indicating whether the association is a direct line whatever the coordinates of its extremities. Notice that in this case annotations are note displayed.
- `O` (**boolean**, intial value: *false*)
  A boolean indicating whether the relation is a self relation. Beware that, this kind of relation is build to be used between a class and itself, this means that both reference points must be equal and be a class label. In other cases, the result is unspecified.
- `O anchor` (**tikz position**, intial value: *north east*)
  A tikz position indicating the point used to determine the cycle relation position. Beware, that the option is meant to be used with the four corners (north east, north west, south east and south west). In other cases, the result is unspecified.
- `O sep` (**length**, intial value: *1cm*)
  The distance the point determining the cycle relation is from its anchor. In fact, this point will have the coordinate $anchor + (O\ sep, O\ sep)$.
- `S` (**boolean**, intial value: *false*)
  A boolean indicating whether the association has an S form (_|‾).
- `CV` (**boolean**, intial value: *false*)
  A boolean indicating whether the association has an C vertical form (⊏).
- `CH` (**boolean**, intial value: *false*)
  A boolean indicating whether the association has an C horizontal form (⊓).



```
\pgfkeys{/tikz/point/.style={fill=red,circle,inner sep=0pt,minimum size=2pt}}
\begin{tikzpicture}[font=\footnotesize]
  \umlclass[name=N,label=n](0,0){}
  \umlclass[name=P,label=p](6,2){}
  \umlasso[name=0.5,S](n)(p) \draw (start) node[point] {};
  \draw (name) node[point] {}; \draw (end) node[point] {};
  \umlasso[name=0.25,S,name pos=0.25](n)(p) \draw (name) node[point] {};
  \umlasso[name=0.75,S,name anchor=left,name pos=0.75](n)(p) \draw (name) node[point] {};
  \umlasso[name=0.125,S,name anchor=below,name pos=0.125](n)(p) \draw (name) node[point] {};
  \umlasso[name=0.875,S,name anchor=above,name pos=0.875](n)(p) \draw (name) node[point] {};

  \umlasso[name=0.5,S,inv route](n)(p) \draw (start) node[point] {};
  \draw (name) node[point] {}; \draw (end) node[point] {};
  \umlasso[name=0.25,S,inv route,name pos=0.25](n)(p) \draw (name) node[point] {};
  \umlasso[name=0.75,S,inv route,name anchor=below,name pos=0.75](n)(p)
  \draw (name) node[point] {};
  \umlasso[name=0.125,S,inv route,name anchor=left,name pos=0.125](n)(p)
  \draw (name) node[point] {};
  \umlasso[name=0.875,S,inv route,name anchor=right,name pos=0.875](n)(p)
  \draw (name) node[point] {};
\end{tikzpicture}
\begin{tikzpicture}[font=\footnotesize]
  \umlclass[name=N,label=n](0,0){}
  \umlclass[name=M,label=m](6,0){}
  \umlasso[name=0.5,CH](n)(m) \draw (start) node[point] {};
  \draw (name) node[point] {}; \draw (end) node[point] {};
  \umlasso[name=0.25,CH,name pos=0.25](n)(m) \draw (name) node[point] {};
  \umlasso[name=0.75,CH,name pos=0.75](n)(m) \draw (name) node[point] {};
  \umlasso[name=0.125,CH,name anchor=left,name pos=0.125](n)(m) \draw (name) node[point] {};
  \umlasso[name=0.875,CH,name anchor=right,name pos=0.875](n)(m) \draw (name) node[point] {};
\end{tikzpicture}
```

Figure 4: `name pos` option explained



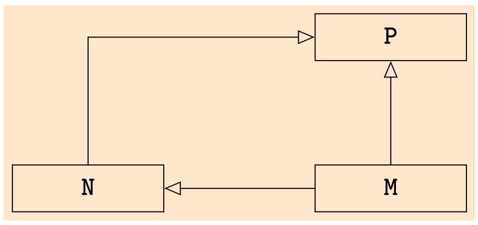
```
\pgfkeys{/tikz/point/.style={fill=red,circle,inner sep=0pt,minimum size=2pt}}
\begin{tikzpicture}
  \begin{scope}[thick]
    \umlclass[name=N,label=n](0,0){}    \umlclass[name=M,label=m](8,0){}
    \umlasso[sep=8mm,arity start=A,name start=B](m)(n)
  \end{scope}
  \draw (start) node[point] {};         \draw ([xshift=-8mm]start) node[point] {};
  \draw ([xshift=-8mm]start) -- +(0,0.3);
  \draw[<->] ([yshift=2mm,xshift=-8mm]start) -- ([yshift=2mm]start)
             node[above,pos=0.5] {\texttt{sep}} ;
\end{tikzpicture}
```

Figure 5: An association with `sep` option explained

- `C sep` (**length**, intial value: *2cm*)
  For C association, the distance between the middle of the two classes and the middle of the association (see figure 3 on page 7).
- `inv route` (**boolean**, intial value: *false*)
  This boolean controls the path of the association in case of an L (__┘ instead of ┌‾‾), a S (┌─┘ instead of _┌‾) or a C (└─┘ and ⊐ instead of ┌‾┐ and ⊏) kind of association.
- `start shift` (**length**, intial value: *0mm*)
- `end shift` (**length**, intial value: *0mm*)
- `shift` (**length**, intial value: *0mm*)
  Extremities shifting specifications respectively for the start, the end and both sides. When used the association start or end point is shifted by the given value staying on the shape border. This means that on west and east side, the shift is a y shift and on north and south border, it is a x shift. The given length may be negative.
- `dashed` (**boolean**, intial value: *false*)
  If true the relation is typeset dashed.
- `compo` (**boolean**, intial value: *false*)
- `agre` (**boolean**, intial value: *false*)
- `nav start` (**boolean**, intial value: *false*)
- `nav end` (**boolean**, intial value: *false*)
  Booleans to specifies the arrow form of the relation. They respectively means composition (◆—), aggregation (◇—), navigation at start (⟵) or navigation at end (⟶). Composition and aggregation "arrows" may only be put at the beginning extremities. Among the three beginning arrows, aggregation wins on composition that wins on navigation.

3.3 Inheritance

The second form of relations is the inheritance relation. The macro `\umlinherit` has the now usual form of the relation macro as demonstrated by the following example:



```
\begin{tikzpicture}[font=\footnotesize]
  \umlclass[name=N,label=n](0,0){}
  \umlclass[name=M,label=m](4,0){}
  \umlclass[name=P,label=p](4,2){}
  \umlinherit(m)(n)   \umlinherit(m)(p)   \umlinherit(n)(p)
\end{tikzpicture}
```

Inheritance are relations, in the sense that they will be represented as relations and therefore have all the options of the relation macro (except for arrow that is specified automatically). And, in fact, one have the 7 kinds of inheritance.

The following options are supported by the `\umlinherit` macro:

– `D` (**boolean**, intial value: *false*)
  A boolean indicating whether the inheritance is a direct line whatever the coordinates of its extremities. Notice that in this case annotations are note displayed.

– `S` (**boolean**, intial value: *false*)
  A boolean indicating whether the inheritance has an S form (_┘‾).

– `CV` (**boolean**, intial value: *false*)
  A boolean indicating whether the inheritance has an C vertical form (⊏).

– `CH` (**boolean**, intial value: *false*)
  A boolean indicating whether the inheritance has an C horizontal form (┌─┐).

– `C sep` (**length**, intial value: *2cm*)
  For C inheritance, the distance between the middle of the two classes and the middle of the inheritance (see figure 3 on page 7).

– `inv route` (**boolean**, intial value: *false*)
  This boolean controls the path of the inheritance in case of an L (__┘ instead of ┌──), a S (┌─┘ instead of _┘‾) or a C (└─┘ and ⊐ instead of ┌─┐ and ⊏) kind of inheritance.

– `start shift` (**length**, intial value: *0mm*)

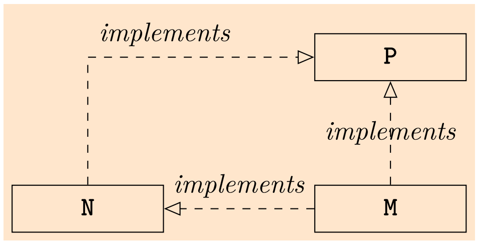
– `end shift` (**length**, intial value: *0mm*)

– `shift` (**length**, intial value: *0mm*)
  Extremities shifting specifications respectively for the start, the end and both sides. When used the inheritance start or end point is shifted by the given value staying on the shape border. This means that on west and east side, the shift is a y shift and on north and south border, it is a x shift. The given length may be negative.

– `dashed` (**boolean**, intial value: *false*)
  If true the inheritance is typeset dashed.

3.4 Implements

The last form of relation is the implements relation. The macro `\umlimplem` has the now usual form of the relation macro as demonstrated by the following example:



```
\begin{tikzpicture}[font=\footnotesize]
  \umlclass[name=N,label=n](0,0){}
  \umlclass[name=M,label=m](4,0){}
  \umlclass[name=P,label=p](4,2){}
  \umlimplem(m)(n)  \umlimplem(m)(p)   \umlimplem(n)(p)
\end{tikzpicture}
```

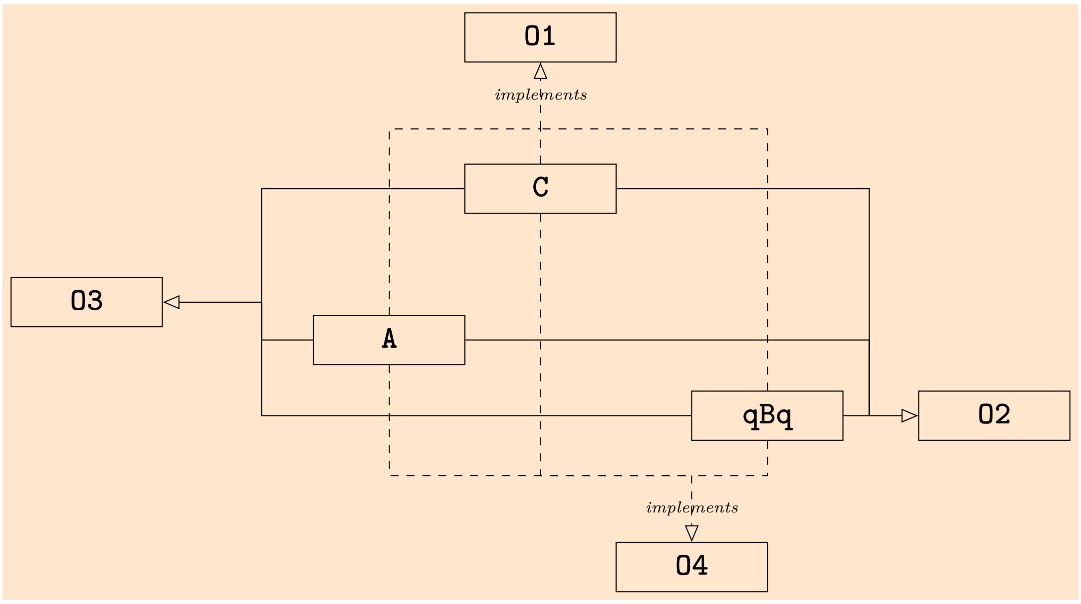
Implements are relations, in the sense that they will be represented as relations and therefore have all the options of the relation macro (except for arrow that is specified automatically). And, in fact, one have the 7 kinds of implements.

The following options are supported by the `\umlimplem` macro:

- `name pos` (**number between 0 and 1**, intial value: *0.5*)
  The position of the name along the segment from start to end of the implementation.
- `D` (**boolean**, intial value: *false*)
  A boolean indicating whether the implements is a direct line whatever the coordinates of its extremities. Notice that in this case annotations are note displayed.
- `S` (**boolean**, intial value: *false*)
  A boolean indicating whether the implements has an S form (─┘┌─).
- `CV` (**boolean**, intial value: *false*)
  A boolean indicating whether the implements has an C vertical form (⊏).
- `CH` (**boolean**, intial value: *false*)
  A boolean indicating whether the implements has an C horizontal form (┌──┐).
- `C sep` (**length**, intial value: *2cm*)
  For C implements, the distance between the middle of the two classes and the middle of the implements (see figure 3 on page 7).
- `inv route` (**boolean**, intial value: *false*)
  This boolean controls the path of the implements in case of an L (──┘ instead of ┌──), a S (┌──┘ instead of ─┘┌─) or a C (└──┘ and ⊐ instead of ┌──┐ and ⊏) kind of implements.
- `start shift` (**length**, intial value: *0mm*)
- `end shift` (**length**, intial value: *0mm*)
- `shift` (**length**, intial value: *0mm*)
  Extremities shifting specifications respectively for the start, the end and both sides. When used the implements start or end point is shifted by the given value staying on the shape border. This means that on west and east side, the shift is a y shift and on north and south border, it is a x shift. The given length may be negative.
- `dashed` (**boolean**, intial value: *true*)
  If true the implements is typeset dashed.

3.5 Multiple inheritance or implements

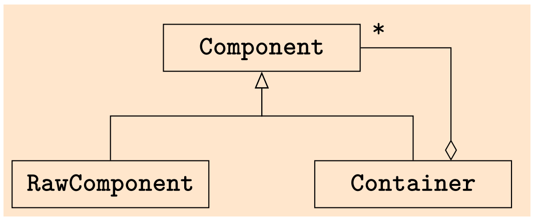
The last relation macros are `\umlinheritmult` and `\umlimplemmult` that connect a set of inheritance (resp. implements) through a common joining point. They are not considered as usual relation and therefore no options are supported. They only take two arguments between parentheses. The first corresponds to a comma separated list of classes (or any other rectangular shapes) and the second argument is the mother class (or any other rectangular shape). The following example illustrate the use of both macros:



```
\begin{tikzpicture}
  \umlclass[name=A,label=a](-1,1){}
  \umlclass[name=qBq,label=b](4,0){}
  \umlclass[name=C,label=c](1,3){}

  \umlclass[name=O1,label=o1](1,5){}
  \umlclass[name=O2,label=o2](7,0){}
  \umlclass[name=O3,label=o3](-5,1.5){}
  \umlclass[name=O4,label=o4](3,-2){}

  \begin{scope}[font=\tiny]
    \umlinheritmult(a,b,c)(o2)
    \umlinheritmult(a,b,c)(o3)
    \umlimplemmult(a,b,c)(o1)
    \umlimplemmult(a,b,c)(o4)
  \end{scope}
\end{tikzpicture}
```

4 Some class diagram examples

First, a classical class diagram for the pattern composite.

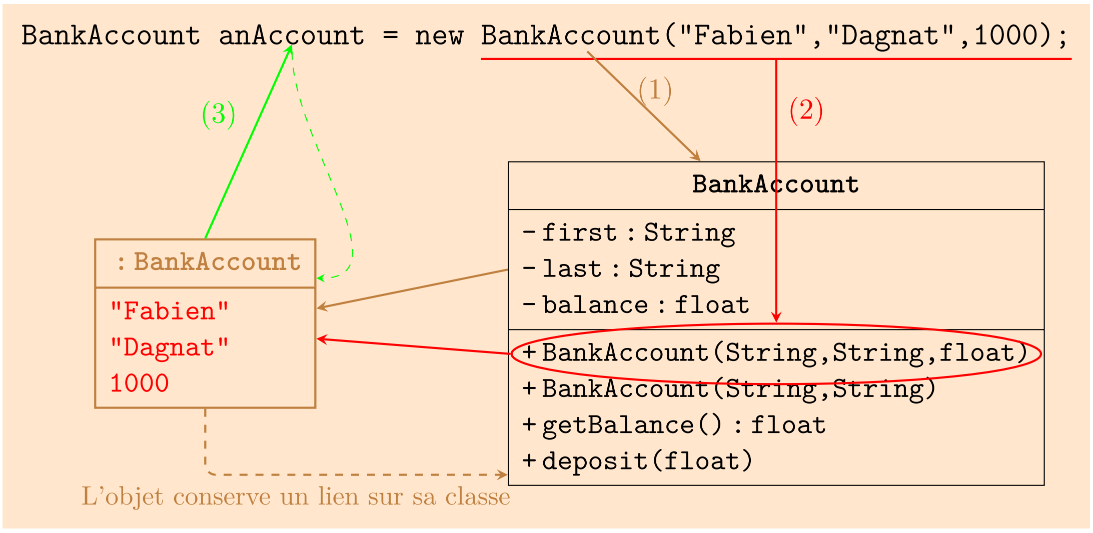


```
\begin{tikzpicture}[font=\footnotesize]
  \umlclass[name=Component,label=Composant,width=2.6cm](4,2.8){}
  \umlclass[name=Container,label=Conteneur,width=2.6cm](6,1){}
  \umlclass[name=RawComponent,label=Interacteur,width=2.6cm](2,1){}
  \umlinherit[S, inv route](Conteneur)(Composant)
  \umlinherit[S, inv route](Interacteur)(Composant)
  \umlasso[agre,start shift=5mm,name end=\textasteriskcentered](Conteneur)(Composant)
\end{tikzpicture}
```

Another one to give some ideas about how to build complex diagrams. Notice, that for aesthetic reason, I advocate the use of thin spaces before and after `:` and after `+` and `-`. For this, the package offer a macro `\feature` that takes three arguments (1) a visibility, (2) a name and (3) a type, the first being optional. It may be used for attributes as well as for methods, as demonstrated above:

```
name : type
name
- name : type
- name
+ name(args) : type
```

```
\ttfamily
\begin{tabular}{l}
  \feature{name}{type} \\
  \feature{name}{} \\
  \feature[-]{name}{type}\\
  \feature[-]{name}{}\\
  \feature[+]{name(args)}{type}
\end{tabular}
```


BankAccount anAccount = new BankAccount("Fabien","Dagnat",1000);
(1)
(2)
(3)
BankAccount
- first : String
- last : String
- balance : float
+ BankAccount(String,String,float)
+ BankAccount(String,String)
+ getBalance() : float
+ deposit(float)
: BankAccount
"Fabien"
"Dagnat"
1000
L'objet conserve un lien sur sa classe

```
\begin{tikzpicture}[remember picture,font=\small]
  \draw (7,4.5) node[font=\ttfamily] (code) { BankAccount \marknode{var}{anAccount} = new
    \marknode{const}{BankAccount("Fabien","Dagnat",1000);}
  };
  \begin{scope}[line width=0.8pt,color=red]
    \draw ([yshift=-0.1cm]const.south west) -- ([yshift=-0.1cm]const.south east)
    node[coordinate,pos=0.5] (middle) {} ;
  \end{scope}
  \umlclass[name=BankAccount,label=class,parts=3]([yshift=-3.5cm]middle) {
    -\,first\,:\,String\\
    \feature[-]{last}{String}\\
    -\,balance\,:\,float
    \separator
    \marknode{constr}{+\,BankAccount(String,String,float)}\\
    +\,BankAccount(String,String)\\
    \feature[+]{getBalance()}{float}\\
    \feature[+]{deposit(float)}{}
  }
  \begin{scope}[line width=0.8pt,color=brown]
    \umlclass[name={\,:\,BankAccount},label=obj,parts=2]([xshift=-4cm]class.west) {
      \color{red}"Fabien"\\
      \color{red}"Dagnat"\\
      \color{red}1000
    }
    \draw[-stealth] ([xshift=-2.5cm]const.south) -- ([xshift=-1cm]class.north)
    node[above,pos=0.6] {(1)} ;
    \draw[-stealth] (class.attribute west) -- ([yshift=2mm]obj.east);
    \draw[dashed,-stealth,rounded corners] (obj.south) |- ([yshift=-2cm]class.west)
    node[below,pos=0.65,font=\footnotesize] {L'objet conserve un lien sur sa classe};
  \end{scope}
  \begin{scope}[line width=0.8pt,color=red]
    \draw (constr) node[draw,ellipse,minimum width=7cm,minimum height=0.8cm] (ellipse) {};
    \draw[stealth-] (ellipse) -- node[right,pos=0.8] {(2)} (middle);
    \draw[line width=0.8pt,-stealth] (ellipse.west) -- ([yshift=-2mm]obj.east);
  \end{scope}
  \begin{scope}[color=green]
    \draw[line width=0.8pt,-stealth] (obj.north) -- node[above left] {(3)} (var.south);
    \draw[dashed,-stealth] ([yshift=-1mm]var.south) to[out=-90,in=0] ([yshift=6mm]obj.east);
  \end{scope}
\end{tikzpicture}
```

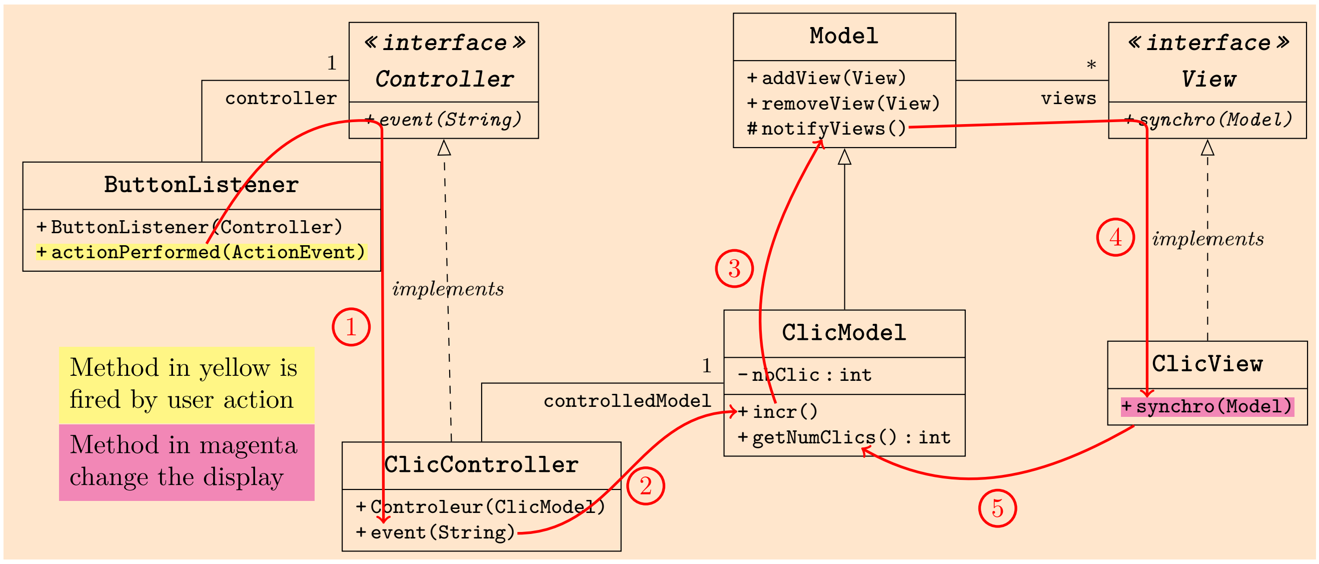
A non convention UML class diagram also illustrating some advanced usage of `tikz`:

– *styles:* a style is a set of options with values that are grouped and named. It is possible to define such style using *tikz* (more precisely, the key management package of `pgf`). Such a style is defined by the syntax `key/.style={`*`set of option setting`*`}`. It can be defined either in the option part of a tikzpicture or more globally using `\pgfkeys` (for more details see the `pgfkeys` part of the `pgf` documentation). Notice that to be a style for a macro `\m` of the package `pgfuml`, the style has to have the path `/pgfuml/m`.

– *scopes:* `tikzpicture`s are composed of drawing elements that may be grouped inside scopes. These scopes may have graphical options that applies to all elements of the scope. In the following example, we furthermore use the `every node` key of a scope to associate a style to each node it contains.

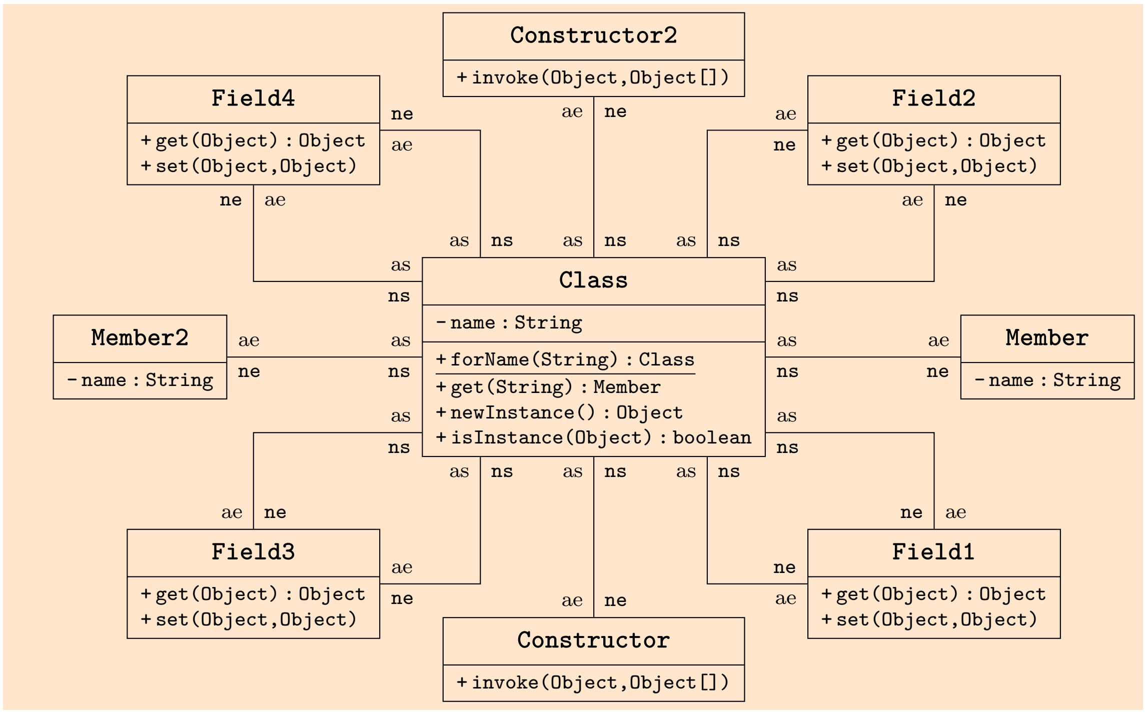
– *pictures inside pictures:* inside a TEX part of a `tikzpicture`, one may use another `tikzpicture`. It is very useful for associating labels to parts of a TEX text. To help, this advanced usage a package `pgfextra` provides a macro `\marknode`. This macro has an optional argument con-

sisting in graphical option passed to the node that will contain the text and two mandatory parameters, first the label and second the text itself. It puts the text inside a node of a subpicture. It is used in the following diagram to mark methods of classes to be able to join them with path.
– *curved path:* The `to` path drawing commands enable to easily build curved path.



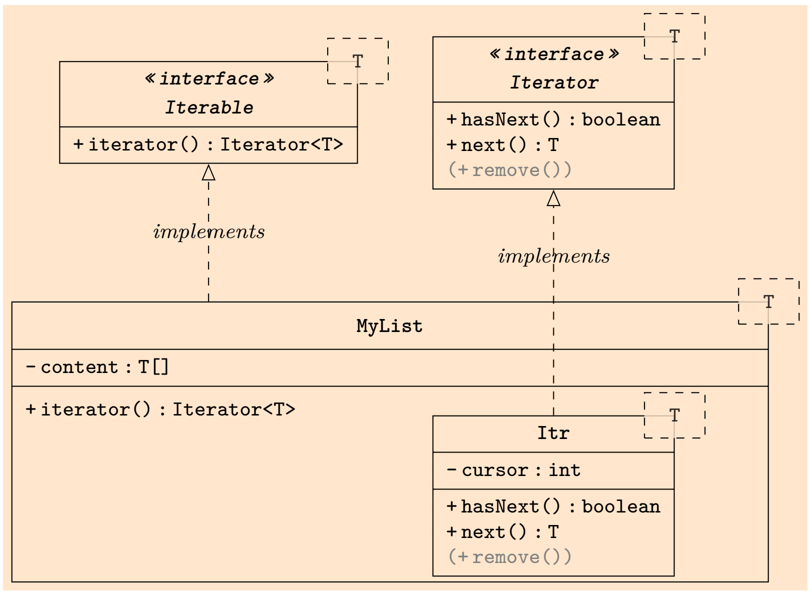
```
\usetikzlibrary{topaths}
\begin{tikzpicture}[font=\footnotesize,out=30,in=150,remember picture,
  /pgfuml/class/cStyle/.style={parts=2,body font=\scriptsize\ttfamily},
  /pgfuml/class/iStyle/.style={interface,parts=2,body font=\scriptsize\itshape\ttfamily}]
  \umlclass[cStyle,name=ButtonListener,label=bl](-8.9,4.2){
    \feature[+]{ButtonListener(Controller)}{}\\
    \marknode[fill=yellow!60]{ap}{\feature[+]{actionPerformed(ActionEvent)}{}}
  }1
  \umlclass[iStyle,name=Controller,label=c](-5.7,6){
      \marknode{e}{\feature[+]{event(String)}{}}
    }
  \umlclass[cStyle,name=Model,label=m]([xshift=5.3cm]c){
    \feature[+]{addView(View)}{}\\
    \feature[+]{removeView(View)}{}\\
    \marknode{nv}{\feature[\#]{notifyViews()}{}}
  }
  \umlclass[iStyle,name=View,label=v]([xshift=4.8cm]m){
    \marknode{synchro}{\feature[+]{synchro(Model)}{}}
  }
  \umlclass[cStyle,name=ClicController,label=ctlr]([yshift=-5.5cm,xshift=5mm]c){
  \feature[+]{Controleur(ClicModel)}{}\\
  \marknode{ec}{\feature[+]{event(String)}{}}
  }
  \umlclass[cStyle,name=ClicModel,label=cm,parts=3]([yshift=-4cm]m){
    \feature[-]{nbClic}{int}
    \separator    \marknode{incr}{\feature[+]{incr()}{}} \\
    \marknode{nbc}{\feature[+]{getNumClics()}{int}}
  }
  \umlclass[cStyle,name=ClicView,label=cv]([xshift=4.8cm]cm){
    \marknode[fill=magenta!60]{synchroC}{\feature[+]{synchro(Model)}{}}
  }
  \begin{scope}[font=\scriptsize]
    \umlasso[arity end=\textasteriskcentered,name end=views](m)(v)
    \umlasso[arity end=1,name end=controller](bl)(c)
    \umlimplem(cv)(v)
    \umlimplem[start shift=-4mm,D](ctlr)(c)
    \umlasso[arity end=1,name end=controlledModel](ctlr)(cm)
  \end{scope}
  \umlinherit(cm)(m)

  \draw ([yshift=-15mm,xshift=-2mm]bl.south) node[text width=3.1cm,fill=yellow!60] (text)
  {Method in yellow is fired by user action};
  \draw (text.south) node[below,text width=3.1cm,fill=magenta!60]
  {Method in magenta change the display};

  \begin{scope}[red,->,line width=1pt,
              every node/.style={circle,draw,inner sep=2pt,outer sep=5pt}]
    \draw (ap) to[out=60,in=180] (e) to[out=0,in=90] ([xshift=-8mm]e.south)
    -- ([xshift=-7mm]ec.north) node[left,pos=0.5] {1} ;
    \draw (ec) to[out=0,in=180] node[below,pos=0.6] {2} (incr);
    \draw (incr) to[out=110,in=-120] node[left] {3} (nv);
    \draw (nv) -- (synchro.west) to[out=0,in=90] ([xshift=-8mm]synchro.south)
    -- ([xshift=-8mm]synchroC.north) node[left,pos=0.4] {4} ;
    \draw (cv) to[out=210,in=-30] node[below] {5} (nbc);
  \end{scope}
\end{tikzpicture}
```

Another example, that illustrates the automatic placement of informations.



```
\newcommand{\aclass}[4]{
  \umlclass[name=#1,label=#2,parts=2,body font=\ttfamily\scriptsize](#3){#4}}
\newcommand{\field}[3]{\aclass{#1}{#2}{#3}{
  \feature[+]{get(Object)}{Object}\\ \feature[+]{set(Object,Object)}{}}
}
\newcommand{\mem}[3]{\aclass{#1}{#2}{#3}{\feature[-]{name}{String}}}
\newcommand{\cst}[3]{\aclass{#1}{#2}{#3}{\feature[+]{invoke(Object,Object[])}{}}}
\begin{tikzpicture}[font=\footnotesize,
  /pgfuml/asso/extremities/.style={font=\scriptsize,arity end=ae,arity start=as,
                                   name end=ne,name start=ns}]
  \umlclass[name=Class,label=class,body font=\ttfamily\scriptsize](2,6.5){
    \feature[-]{name}{String}
    \separator
    \underline{\feature[+]{forName(String)}{Class}}\\
    \feature[+]{get(String)}{Member}\\
    \feature[+]{newInstance()}{Object}\\
    \feature[+]{isInstance(Object)}{boolean}
  }
  \mem{Member}{mem}{8,6.5}      \mem{Member2}{mem2}{-4,6.5}
  \field{Field1}{f}{6.5,3.5}    \field{Field2}{f2}{6.5,9.5}
  \field{Field3}{f3}{-2.5,3.5}  \field{Field4}{f4}{-2.5,9.5}
  \cst{Constructor}{c}{2,2.5}   \cst{Constructor2}{c2}{2,10.5}
  \umlasso[extremities](class)(mem)
  \umlasso[extremities](class)(mem2)
  \umlasso[extremities,inv route](class)(c)
  \umlasso[extremities,inv route](class)(c2)
  \umlasso[extremities,start shift=15mm](class)(f)
  \umlasso[extremities,inv route,start shift=-10mm](class)(f)
  \umlasso[extremities,start shift=15mm](class)(f2)
  \umlasso[extremities,inv route,start shift=10mm](class)(f2)
  \umlasso[extremities,start shift=-15mm](class)(f3)
  \umlasso[extremities,inv route,start shift=-10mm](class)(f3)
  \umlasso[extremities,start shift=-15mm](class)(f4)
  \umlasso[extremities,inv route,start shift=10mm](class)(f4)
\end{tikzpicture}
```

And lastly a more usual UML class diagram that includes an inner class.



```
\begin{tikzpicture}[remember picture,font=\scriptsize,
  /pgfuml/class/iStyle/.style={interface,param=T,parts=2,param fill=graphicbackground},
  /pgfuml/class/cStyle/.style={param=T,parts=3,param fill=graphicbackground}]
  \umlclass[cStyle,name=MyList,label=myList,width=10cm](5.5,2){
    \feature[-]{content}{T[]}
    \separator
    \feature[+]{iterator()}{Iterator<T>}
    \hspace{1.5cm}
    \begin{tikzpicture}[remember picture,baseline=(itr.north)]
      \umlclass[cStyle,name=Itr,label=itr](0,0){
        \feature[-]{cursor}{int}
        \separator        \feature[+]{hasNext()}{boolean}\\
        \feature[+]{next()}{T} \\
        {\color{gray}(\feature[+]{remove()}{})}
      }
    \end{tikzpicture}
  }
  \umlclass[iStyle,name=Iterable,label=iterable]([xshift=-2.4cm,yshift=2.5cm]myList.north){
    \feature[+]{iterator()}{Iterator<T>}
  }
  \umlclass[iStyle,name=Iterator,label=iterator]([yshift=4cm]itr.north){
    \feature[+]{hasNext()}{boolean}\\
    \feature[+]{next()}{T} \\
    {\color{gray}(\feature[+]{remove()}{})}
  }
  \umlimplem[start shift=-2.4cm](myList)(iterable);
  \umlimplem[name pos=0.7](itr)(iterator);
\end{tikzpicture}
```

5 Some other diagrams examples

5.1 Les diagrammes de séquences

Un premier exemple basique pour l'introduction des commandes.

Un exemple un peu complet qui permet d'illustrer la création et le même ensuite avec des labels qui permet d'illustrer la création de nœuds réutilsable par la suite.

Un diagramme un peu plus réaliste.

On peut également faire des boucles et des alternatives.

5.2 Les diagrammes de cas d'utilisation

Il est enfin possible de réaliser des diagrammes de cas d'utilisation. Attention, vu la technique utilisée deux compilations sont necéssaires pour que les liens soient corrects.

Et pour finir un dernier diagramme de cas d'utilisation un peu plus complexe.

6 Modes et variables

6.1 L'environnement `diagseq`

L'environnement `diagseq` peut être configuré par les options suivantes:

Nom	Type	Intention	Valeur(s) par défaut
`step`	longueur	l'écart entre les messages	10mm
`start`	longueur	l'écart au départ	5mm
`base`	longueur	la ligne de base des objets préexistant	0mm
`font`	chaîne	font de la figure	`\normalsize`