3 Diagramme de classes

3.1 Relations entre classes

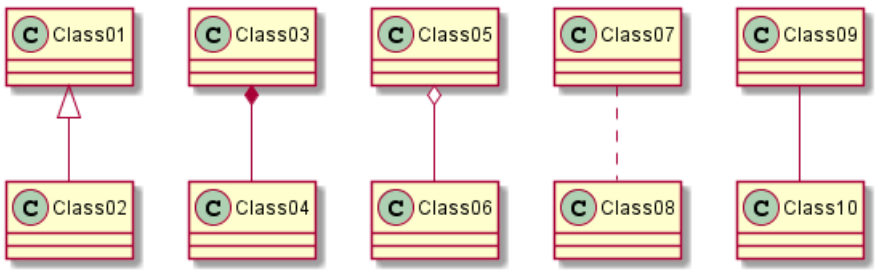
Les relations entre les classes sont définies en utilisant les symboles suivants :

Type	Symbol	Drawing
Extension	`<|--`	
Composition	`*--`	
Aggregation	`o--`	

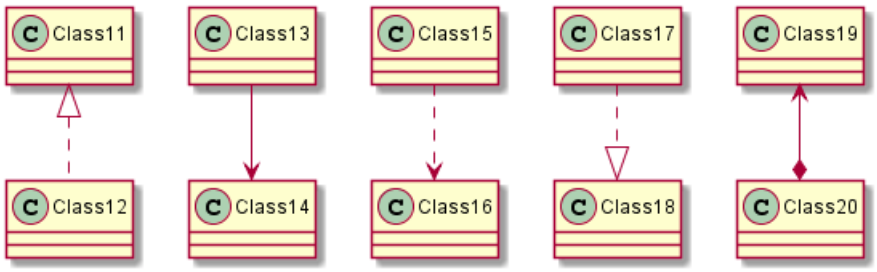
Il est possible de substituer `--` par `..` pour obtenir une ligne en pointillée.

Grâce à ces règles, il est possible de faire les diagrammes suivants :

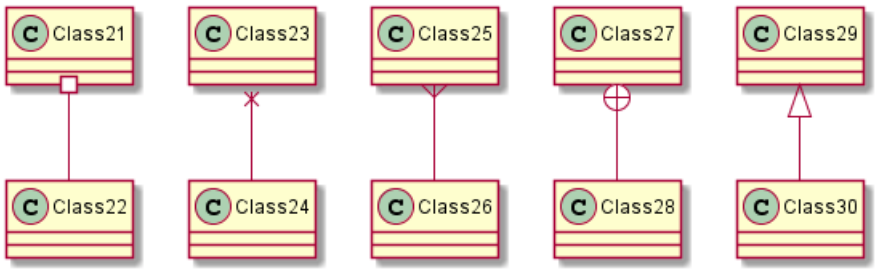
```
@startuml
Class01 <|-- Class02
Class03 *-- Class04
Class05 o-- Class06
Class07 .. Class08
Class09 -- Class10
@enduml
```




```
@startuml
Class11 <|.. Class12
Class13 --> Class14
Class15 ..> Class16
Class17 ..|> Class18
Class19 <--* Class20
@enduml
```




```
@startuml
Class21 #-- Class22
Class23 x-- Class24
Class25 }-- Class26
Class27 +-- Class28
Class29 ^-- Class30
@enduml
```

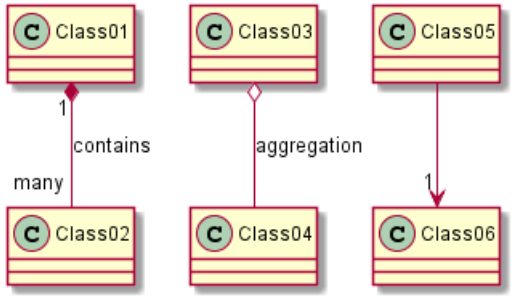


3.2 Libellés sur les relations

Il est possible de rajouter un libellé sur une relation, en utilisant les deux points `:`, suivi du texte du libellé.

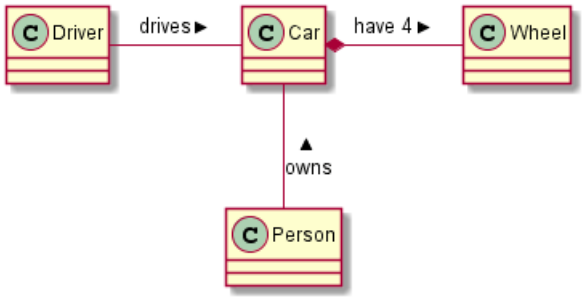
Pour les cardinalité, vous pouvez utiliser des guillemets `""` des deux cotés de la relation.

```
@startuml

Class01 "1" *-- "many" Class02 : contains

Class03 o-- Class04 : aggregation

Class05 --> "1" Class06

@enduml
```



Vous pouvez ajouter une flèche désignant quel objet agit sur l'autre en utilisant < ou > au début ou à la fin du libellé.

```
@startuml
class Car

Driver - Car : drives >
Car *- Wheel : have 4 >
Car -- Person : < owns

@enduml
```

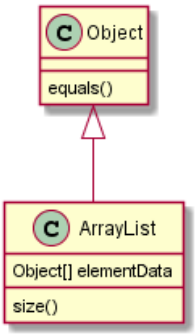


3.3 Définir les méthodes

Pour déclarer des méthodes ou des champs, vous pouvez utiliser le caractère : suivi de la méthode ou du champ.

Le système utilise la présence de parenthèses pour choisir entre méthodes et champs.

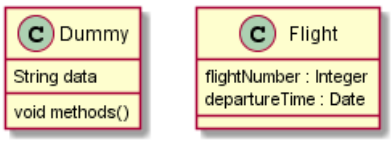
```
@startuml
Object <|-- ArrayList

Object : equals()
ArrayList : Object[] elementData
ArrayList : size()

@enduml
```



Il est possible de regrouper tous les champs et méthodes en utilisant des crochets {}.

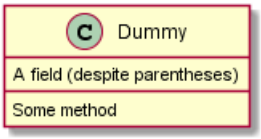
Notez que la syntaxe est très souple sur l'ordre des champs et des méthodes.

```
@startuml
class Dummy {
  String data
  void methods()
}

class Flight {
   flightNumber : Integer
   departureTime : Date
}
@enduml
```



You can use {field} and {method} modifiers to override default behaviour of the parser about fields and methods.

```
@startuml
class Dummy {
  {field} A field (despite parentheses)
  {method} Some method
}

@enduml
```

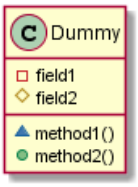


3.4 Définir les visibilités

Quand vous déclarez des champs ou des méthodes, vous pouvez utiliser certains caractères pour définir la visibilité des éléments :

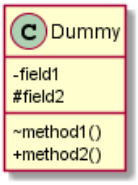
Character	Icon for field	Icon for method	Visibility
-			private
#			protected
~			package private
+			public

```
@startuml

class Dummy {
 -field1
 #field2
 ~method1()
 +method2()
}

@enduml
```



Vous pouvez invalider cette fonctionnalité par la commande `skinparam classAttributeIconSize 0` :

```
@startuml
skinparam classAttributeIconSize 0
class Dummy {
 -field1
 #field2
 ~method1()
 +method2()
}

@enduml
```



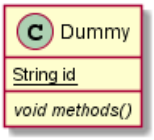
3.5 Abstrait et statique

Vous pouvez définir une méthode statique ou abstraite ou un champ utilisant `{static}` ou `{abstract}` modificateur.

Ce modificateur peut être utilisé au début ou à la fin de la ligne. Vous pouvez alors utiliser `{classifier}` plutôt que `{static}`.

```
@startuml
class Dummy {
  {static} String id
  {abstract} void methods()
}
```

```
@enduml
```

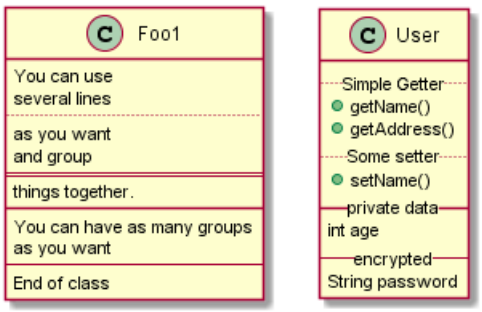


3.6 Corps de classe avancé

Par défaut, méthodes et champs sont automatiquement regroupés par PlantUML. Vous pouvez utiliser un séparateur pour définir votre propre manière d'ordonner les champs et les méthodes. Les séparateurs suivants sont possibles : `--` `..` `==` `__`.

Vous pouvez aussi utiliser les titres dans les séparateurs.

```
@startuml
class Foo1 {
  You can use
  several lines
  ..
  as you want
  and group
  ==
  things together.
  __
  You can have as many groups
  as you want
  --
  End of class
}

class User {
  .. Simple Getter ..
  + getName()
  + getAddress()
  .. Some setter ..
  + setName()
  __ private data __
  int age
  -- encrypted --
  String password
}

@enduml
```



3.7 Notes et stéréotypes

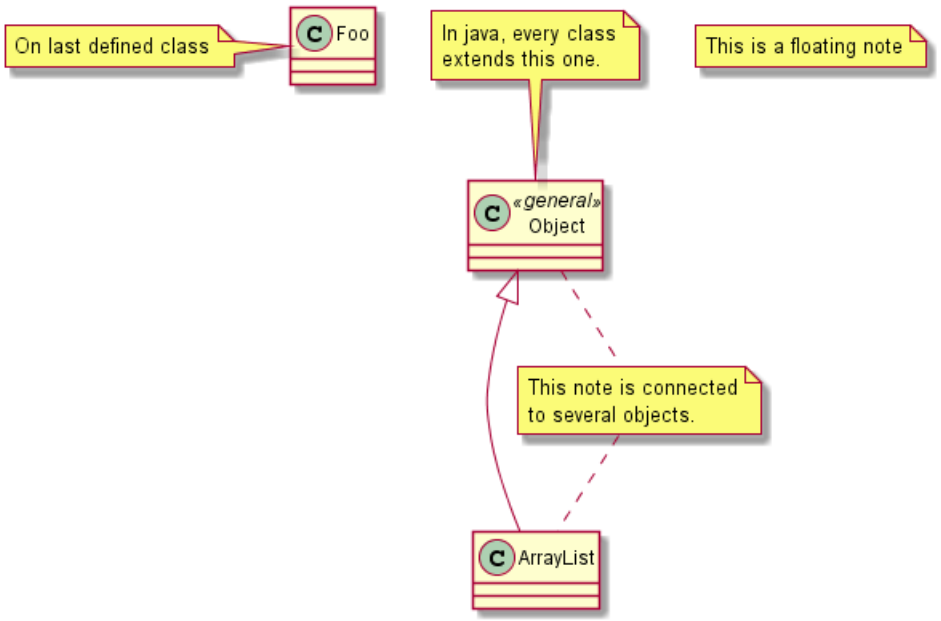
Stéréotypes sont définies avec le mot clé `class`, `<<` et `>>`.

Vous pouvez aussi définir une note en utilisant les mots clés `note left of` , `note right of` , `note top of` , `note bottom of`.

Vous pouvez aussi définir une note sur la dernière classe utilisant `note left`, `note right`, `note top`, `note bottom`.

Une note peut aussi être définie le mot clé `note`, puis être lié à un autre objet en utilisant le symbole `...`

```
@startuml
class Object << general >>
Object <|--- ArrayList

note top of Object : In java, every class\nextends this one.

note "This is a floating note" as N1
note "This note is connected\nto several objects." as N2
Object .. N2
N2 .. ArrayList

class Foo
note left: On last defined class

@enduml
```



3.8 Encore des notes

Il est possible d'utiliser quelques tag HTML comme :

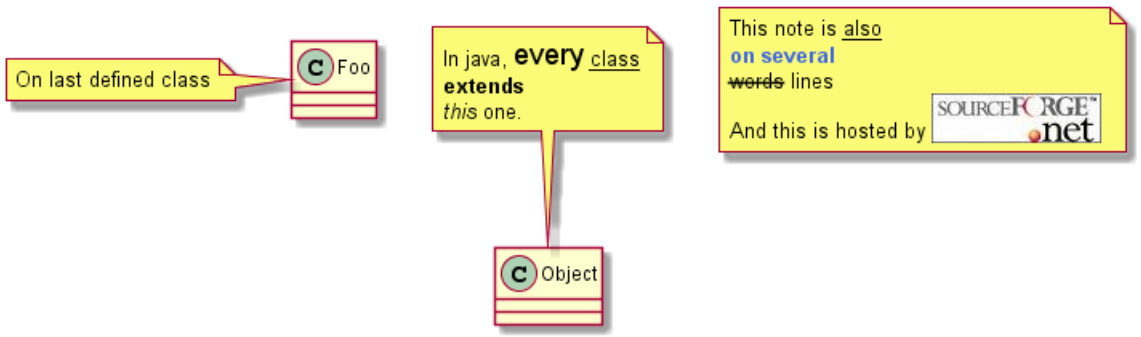
- `<b>`
- `<u>`
- `<i>`
- `<s>`, `<del>`, `<strike>`
- `<font color="#AAAAAA">` or `<font color="colorName">`
- `<color:#AAAAAA>` or `<color:colorName>`

- `<size:nn>` to change font size
- `<img src="file">` or `<img:file>`: the file must be accessible by the filesystem

Vous pouvez aussi définir des notes sur plusieurs lignes.

Vous pouvez également définir une note sur la dernière classe définie en utilisant `note left`, `note right`, `note top`, `note bottom`.

```
@startuml

class Foo
note left: On last defined class

note top of Object
  In java, <size:18>every</size> <u>class</u>
  <b>extends</b>
  <i>this</i> one.
end note

note as N1
  This note is <u>also</u>
  <b><color:royalBlue>on several</color>
  <s>words</s> lines
  And this is hosted by <img:sourceforge.jpg>
end note

@enduml
```

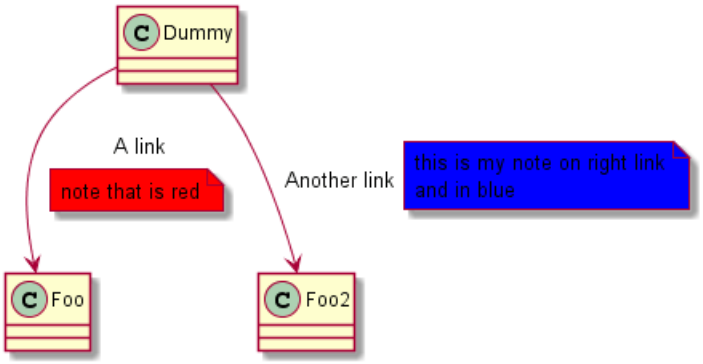


3.9 Note sur les liens

Il est possible d'ajouter une note sur un lien, juste après la définition d'un lien, utiliser `note on link`.

Vous pouvez aussi utiliser `note left on link`, `note right on link`, `note top on link`, `note bottom on link` si vous voulez changer la position relative de la note avec l'étiquette.

```
@startuml

class Dummy
Dummy --> Foo : A link
note on link #red: note that is red

Dummy --> Foo2 : Another link
note right on link #blue
this is my note on right link
and in blue
end note

@enduml
```

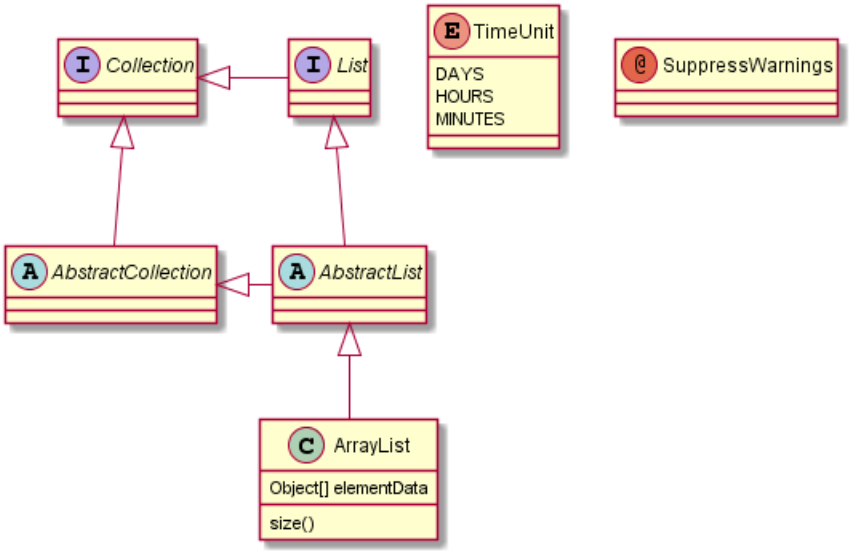


3.10 Classe abstraite et Interface

Vous pouvez déclarer un classe abstraite en utilisant `abstract` ou `abstract class`. La classe sera alors écrite en *italique*.

Vous pouvez aussi utiliser `interface`, `annotation` et `enum`.

```
@startuml

abstract class AbstractList
abstract AbstractCollection
interface List
interface Collection

List <|-- AbstractList
Collection <|-- AbstractCollection

Collection <|- List
AbstractCollection <|- AbstractList
AbstractList <|-- ArrayList

class ArrayList {
  Object[] elementData
  size()
}

enum TimeUnit {
  DAYS
  HOURS
  MINUTES
}

annotation SuppressWarnings

@enduml
```

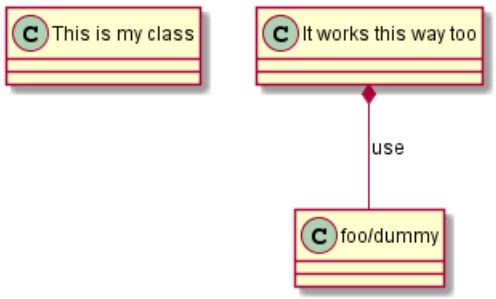


3.11 Caractères non alphabétiques

Si nous voulez utiliser autre chose que des lettres dans les classes (ou les enums...), vous pouvez:

- Utiliser le mot clé `as` dans la définition de la classe
- Metter des guillemets `""` autour du nom de la classe

```
@startuml
class "This is my class" as class1
class class2 as "It works this way too"

class2 *-- "foo/dummy" : use
@enduml
```



3.12 Masquer les attributs et les méthodes

Vous pouvez paramétrer l'affichage des classes à l'aide de la commande `hide/show` .

La commande de base est: `hide empty members`. Cette commande va masquer la zone des champs ou des méthodes si celle-ci est vide.

A la place de `empty members`, vous pouvez utiliser:

- `empty fields` ou `empty attributes` pour des champs vides,
- `empty methods` pour des méthodes vides,
- `fields` or `attributes` qui masque les champs, même s'il y en a de définis,
- `methods` qui masque les méthodes, même s'il y en a de définies,
- `members` qui masque les méthodes ou les champs, même s'il y en a de définies,

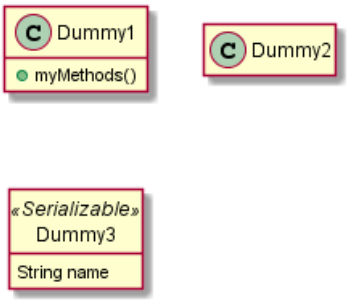
- `circle` pour le caractère entouré en face du nom de la classe,
- `stereotype` pour le stéréotype.

Vous pouvez aussi fournir, juste après le mot-clé `hide` ou `show` :

- `class` pour toutes les classes,
- `interface` pour toutes les interfaces,
- `enum` pour tous les enums,
- `<<foo1>>` pour les classes qui sont stéréotypée avec *foo1*,
- Un nom de classe existant

Vous pouvez utiliser plusieurs commandes `show/hide` pour définir des règles et des exceptions.

```
@startuml

class Dummy1 {
  +myMethods()
}

class Dummy2 {
  +hiddenMethod()
}

class Dummy3 <<Serializable>> {
String name
}

hide members
hide <<Serializable>> circle
show Dummy1 methods
show <<Serializable>> fields

@enduml
```



3.13 Cacher des classes

Vous pouvez également utiliser la commande `show/hide` pour cacher une classe.

Cela peut être utile si vous définissez un fichier inclus de grande taille, et si vous voulez en cacher quelques classes après l'inclusion de ce fichier.

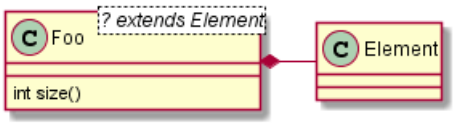
```
@startuml

class Foo1
class Foo2

Foo2 *-- Foo1

hide Foo2
```

```
@enduml
```

3.14 Utilisation de la généricité

Vous pouvez aussi utiliser les signes inférieur < et supérieur > pour définir l'utilisation de la généricité dans une classe.

```
@startuml

class Foo<? extends Element> {
  int size()
}
Foo *- Element

@enduml
```



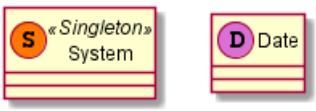
On peut désactiver ce comportement avec la commande `skinparam genericDisplay old`.

3.15 Caractère spécial

Normalement, un caractère (C, I, E ou A) est utilisé pour les classes, les interfaces ou les énum.

Vous pouvez aussi utiliser le caractère de votre choix, en définissant le stéréotype et en ajoutant une couleur, comme par exemple :

```
@startuml

class System << (S,#FF7700) Singleton >>
class Date << (D,orchid) >>
@enduml
```



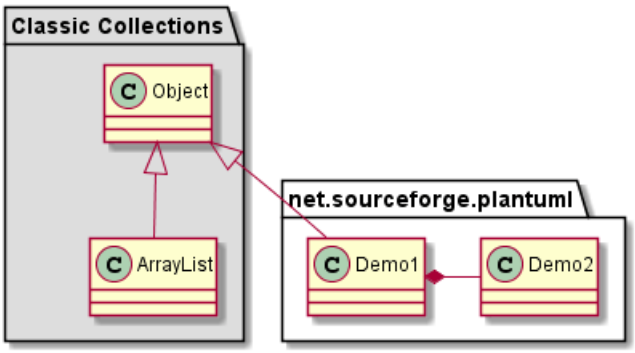
3.16 Packages

Vous pouvez définir un package en utilisant le mot-clé `package`, et optionnellement déclarer une couleur de fond pour votre package (en utilisant un code couleur HTML ou son nom).

Notez que les définitions de packages peuvent être imbriquées.

```
@startuml

package "Classic Collections" #DDDDDD {
```

```
  Object <|-- ArrayList
}

package net.sourceforge.plantuml {
  Object <|-- Demo1
  Demo1 *- Demo2
}

@enduml
```



3.17 Modèle de paquet

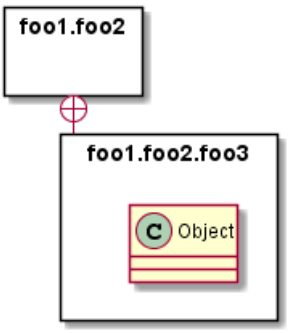
Il y a différents styles de paquets disponibles.

Vous pouvez les spécifier chacun par un réglage par défaut avec la commande : `skinparam packageStyle`, ou par l'utilisation d'un stéréotype sur le paquet:

```
@startuml
scale 750 width
package foo1 <<Node>> {
  class Class1
}

package foo2 <<Rectangle>> {
  class Class2
}

package foo3 <<Folder>> {
  class Class3
}

package foo4 <<Frame>> {
  class Class4
}

package foo5 <<Cloud>> {
  class Class5
}

package foo6 <<Database>> {
  class Class6
}

@enduml
```

Vous pouvez aussi définir les liens entre les paquets, comme dans l'exemple suivant :

```
@startuml

skinparam packageStyle rectangle

package foo1.foo2 {
}

package foo1.foo2.foo3 {
  class Object
}

foo1.foo2 +-- foo1.foo2.foo3

@enduml
```



3.18 Les espaces de nommage

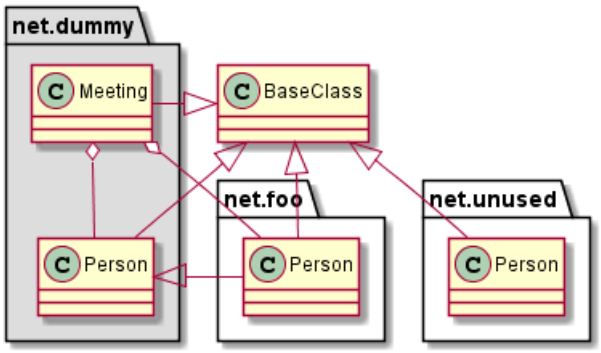
Avec les packages, le nom de la classe est l'identifiant unique de la classe. Cela signifie qu'on ne peux pas avoir deux classes avec le même nom dans deux packages différents. Pour ce faire, vous devez utiliser des espace de nommage (*namespace*) à la place des packages.

Vous pouvez faire référence à des classes d'autres espace de nommage en les nommant complétement. Les classes de l'espace de nommage par défaut (racine) sont nommées en commençant par un point.

Il n'est pas obligatoire de créer les espaces de nom. Un classe avec son nom complet sera automatiquement ajoutée au bon espace de nommage.

```
@startuml

class BaseClass

namespace net.dummy #DDDDDD {
.BaseClass <|-- Person
Meeting o-- Person

.BaseClass <|- Meeting
}

namespace net.foo {
```

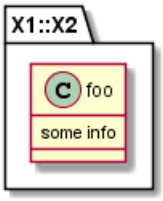
```
  net.dummy.Person  <|- Person
  .BaseClass <|-- Person

  net.dummy.Meeting o-- Person
}

BaseClass <|-- net.unused.Person

@enduml
```



3.19 Creation automatique d'espace de nommage

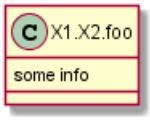
Vous pouvez définir une autre séparateur (autre que le point) en utilisant la commande : `set namespaceSeparator ???`.

```
@startuml

set namespaceSeparator ::
class X1::X2::foo {
  some info
}

@enduml
```



Vous pouvez désactiver la création automatique de package en utilisant la commande `set namespaceSeparator none`.

```
@startuml

set namespaceSeparator none
class X1.X2.foo {
  some info
}

@enduml
```

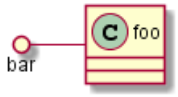


3.20 Interface boucle

Vous pouvez aussi rajouter des interfaces sur les classes avec la syntaxe suivante:

- `bar ()- foo`
- `bar ()-- foo`
- `foo -() bar`

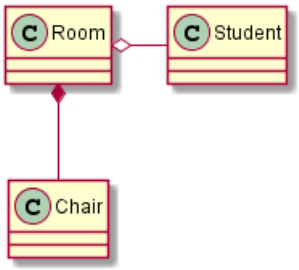
```
@startuml
class foo
bar ()- foo
@enduml
```



3.21 Changer la direction

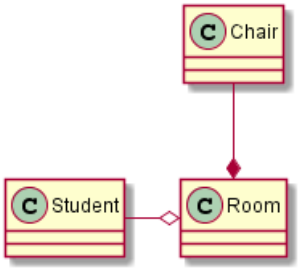
Par défaut, les liens entre les classe ont deux tirets `--` et sont orientés verticalement. Il est possible d'utiliser une ligne horizontal en mettant un simple tiret (Ou un point) comme ceci:

```
@startuml
Room o- Student
Room *-- Chair
@enduml
```



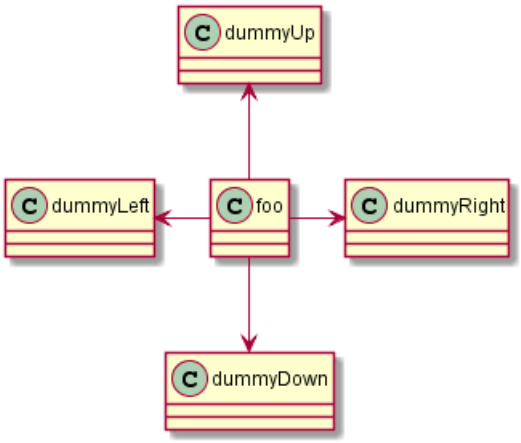
Vous pouvez aussi changer le sens en renversant le lien :

```
@startuml
Student -o Room
Chair --* Room
@enduml
```



Il est aussi possible de changer la direction d'une flèche en ajoutant les mots clés `left`, `right`, `up` ou `down` à l'intérieur de la flèche:

```
@startuml
foo -left-> dummyLeft
foo -right-> dummyRight
foo -up-> dummyUp
foo -down-> dummyDown
@enduml
```



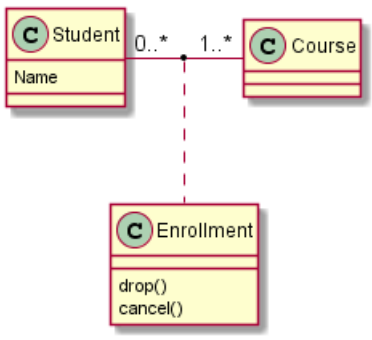
Il est possible de raccourcir la flèche en n'utilisant que la première lettre de la direction (par exemple, `-d-` au lieu de `-down-`) ou les deux premières lettres (`-do-`)

Attention à ne pas abuser de cette fonctionnalité : *GraphViz* donne généralement de bons résultats sans trop de raffistolages.

3.22 Classes d'association

Vous pouvez définir une *classe d'association* après qu'une relation ait été définie entre deux classes, comme dans l'exemple suivant:

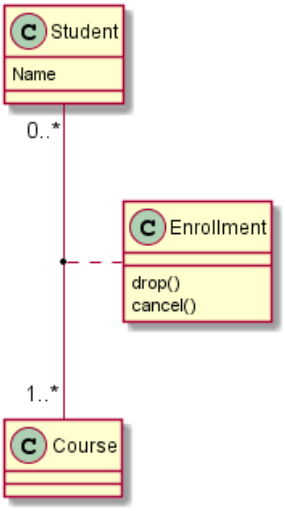
```
@startuml
class Student {
  Name
}
Student "0..*" - "1..*" Course
(Student, Course) .. Enrollment

class Enrollment {
  drop()
  cancel()
}
@enduml
```



Vous pouvez la définir dans une autre direction :

```
@startuml
class Student {
```

```
  Name
}
Student "0..*" -- "1..*" Course
(Student, Course) . Enrollment

class Enrollment {
  drop()
  cancel()
}
@enduml
```



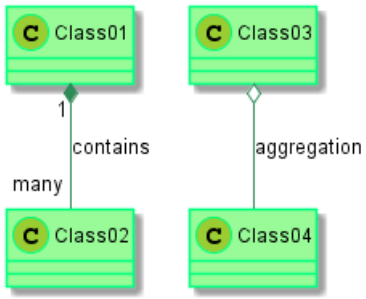
3.23 Personnalisation

La commande `skinparam` permet de changer la couleur et les polices de caractères.

Vous pouvez utiliser cette commande :

- Dans le diagramme, comme toutes les autre commandes,
- Dans un fichier inclus,
- Dans un fichier de configuration précisé par la ligne de commande ou la tâche ANT.

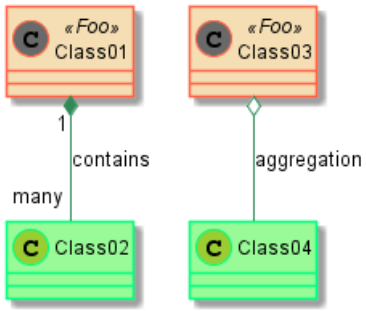
```
@startuml

skinparam class {
BackgroundColor PaleGreen
ArrowColor SeaGreen
BorderColor SpringGreen
}
skinparam stereotypeCBackgroundColor YellowGreen

Class01 "1" *-- "many" Class02 : contains

Class03 o-- Class04 : aggregation

@enduml
```



3.24 Stéréotypes Personnalisés

Vous pouvez définir des couleurs et des fontes de caractères spécifiques pour les classes stéréotypées.

```
@startuml

skinparam class {
BackgroundColor PaleGreen
ArrowColor SeaGreen
BorderColor SpringGreen
BackgroundColor<<Foo>> Wheat
BorderColor<<Foo>> Tomato
}
skinparam stereotypeCBackgroundColor YellowGreen
skinparam stereotypeCBackgroundColor<< Foo >> DimGray

Class01 <<Foo>>
Class03 <<Foo>>
Class01 "1" *-- "many" Class02 : contains

Class03 o-- Class04 : aggregation

@enduml
```



3.25 Dégradé de couleur

Il est possible de déclarer individuellement une couleur pour des classes ou une note en utilisant la notation #.

Vous pouvez utiliser un nom de couleur standard ou un code RGB.

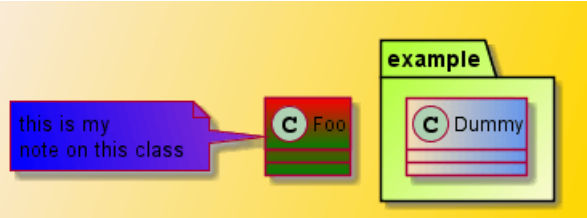
Vous pouvez aussi utiliser un dégradé de couleur en fond, avec la syntaxe suivante : deux noms de couleurs séparés par :

- |,
- /,
- \,
- ou -

en fonction de la direction du dégradé

Par exemple, vous pouvez avoir :

```
@startuml

skinparam backgroundcolor AntiqueWhite/Gold
skinparam classBackgroundColor Wheat|CornflowerBlue

class Foo #red-green
note left of Foo #blue\9932CC
  this is my
  note on this class
end note

package example #GreenYellow/LightGoldenRodYellow {
  class Dummy
}

@enduml
```



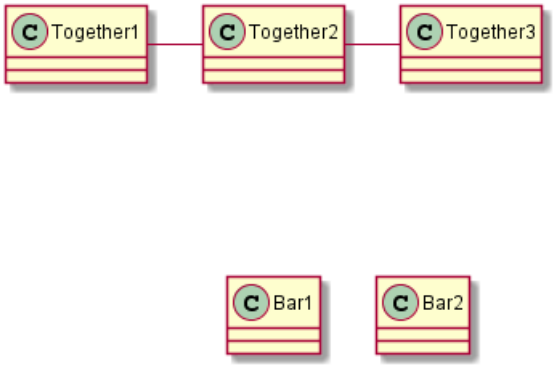
3.26 Aide pour la mise en page

Sometimes, the default layout is not perfect...

You can use `together` keyword to group some classes together : the layout engine will try to group them (as if they were in the same package).

You can also use `hidden` links to force the layout.

```
@startuml

class Bar1
class Bar2
together {
  class Together1
  class Together2
  class Together3
}
Together1 - Together2
Together2 - Together3
Together2 -[hidden]--> Bar1
Bar1 -[hidden]> Bar2


@enduml
```



3.27 Découper les grands diagrammes

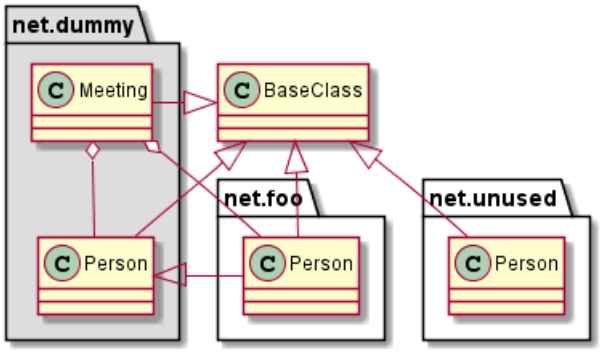
Parfois, vous obtiendrez des images de taille importante.

Vous pouvez utiliser la commande `page (hpages)x(vpages)` pour découper l'image en plusieurs fichiers:

`hpages` est le nombre de pages horizontales et `vpages` indique le nombre de pages verticales.

Vous pouvez aussi utiliser des paramètres spécifiques pour rajouter des bords sur les pages découpées (voir l'exemple).

```
@startuml
' Split into 4 pages
page 2x2
skinparam pageMargin 10
skinparam pageExternalColor gray
skinparam pageBorderColor black

class BaseClass

namespace net.dummy #DDDDDD {
.BaseClass <|-- Person
Meeting o-- Person

.BaseClass <|- Meeting

}

namespace net.foo {
  net.dummy.Person  <|- Person
  .BaseClass <|-- Person

  net.dummy.Meeting o-- Person
}

BaseClass <|-- net.unused.Person
@enduml
```


net.dummy
Meeting
BaseClass
net.foo
net.unused
Person
Person
Person