Drawing UML with PlantUML

PlantUML Language Reference Guide

(Version 1.2019.9)

PlantUML is a component that allows to quickly write :

- Chapter 1 Sequence diagram
- Chapter 2 Usecase diagram
- Chapter 3 Class diagram
- Chapter 4 Activity diagram
- Chapter 5 Component diagram
- Chapter 6 State diagram
- Chapter 7 Object diagram
- Chapter 8 Deployment diagram
- Chapter 9 Timing diagram

The following non-UML diagrams are also supported:

- Wireframe graphical interface
- Archimate diagram
- Specification and Description Language (SDL)
- Ditaa diagram
- Gantt diagram
- MindMap diagram
- Work Breakdown Structure diagram
- Mathematic with AsciiMath or JLaTeXMath notation

Diagrams are defined using a simple and intuitive language.

3 Class Diagram

3.1 Relations between classes

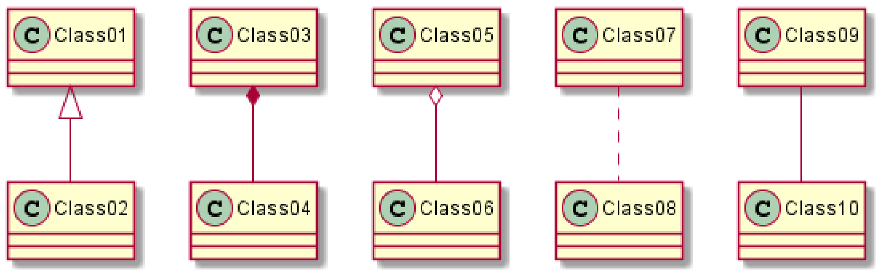
Relations between classes are defined using the following symbols :

Type	Symbol	Drawing
Extension	`<|--`	◁—
Composition	`*--`	◆—
Aggregation	`o--`	◇—

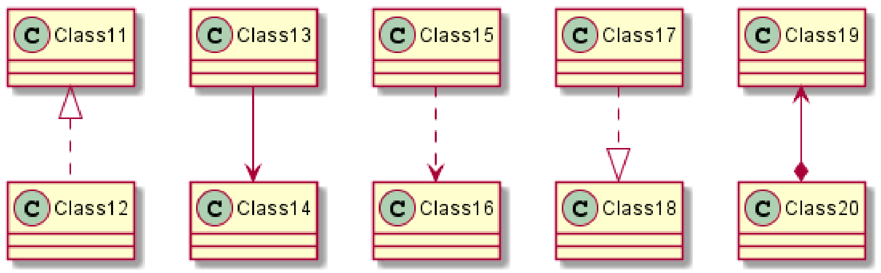
It is possible to replace `--` by `..` to have a dotted line.

Knowing those rules, it is possible to draw the following drawings:

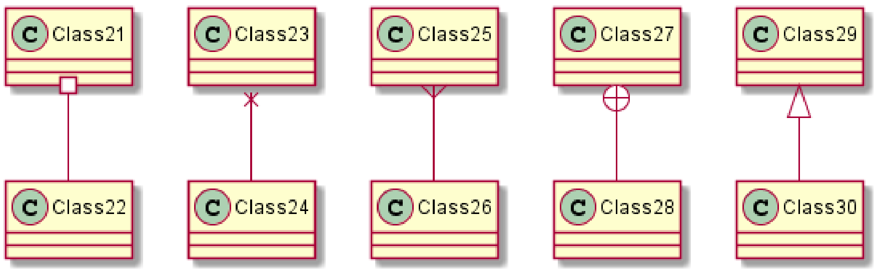
```
@startuml
Class01 <|-- Class02
Class03 *-- Class04
Class05 o-- Class06
Class07 .. Class08
Class09 -- Class10
@enduml
```



```
@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
```

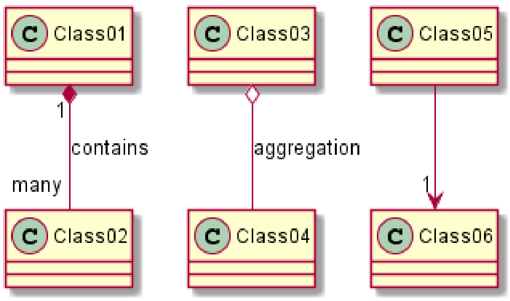


3.2 Label on relations

It is possible a add a label on the relation, using `:`, followed by the text of the label.

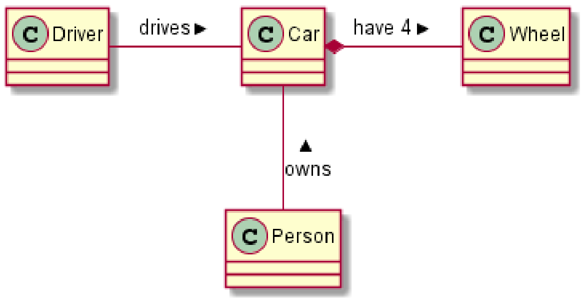
For cardinality, you can use double-quotes `""` on each side of the relation.

```
@startuml

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

Class03 o-- Class04 : aggregation

Class05 --> "1" Class06

@enduml
```



You can add an extra arrow pointing at one object showing which object acts on the other object, using `<` or `>` at the begin or at the end of the label.

```
@startuml
class Car

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

@enduml
```

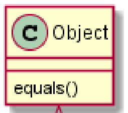


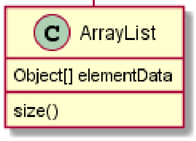
3.3 Adding methods

To declare fields and methods, you can use the symbol `:` followed by the field's or method's name.

The system checks for parenthesis to choose between methods and fields.

```
@startuml
Object <|-- ArrayList

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

@enduml
```

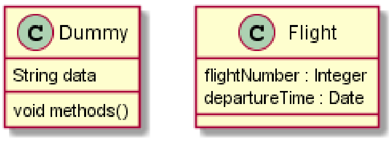




It is also possible to group between brackets `{}` all fields and methods.

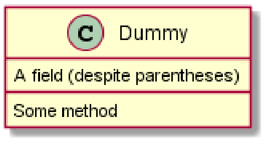
Note that the syntax is highly flexible about type/name order.

```
@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
```

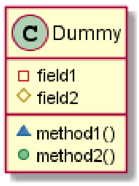


3.4 Defining visibility

When you define methods or fields, you can use characters to define the visibility of the corresponding item:

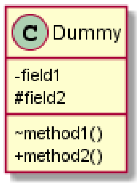
Character	Icon for field	Icon for method	Visibility
-	□	■	private
#	◇	◆	protected
~	△	▲	package private
+	○	●	public

```
@startuml

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

@enduml
```



You can turn off this feature using the `skinparam classAttributeIconSize 0` command :

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

@enduml
```

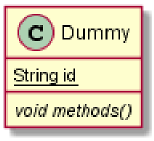


3.5 Abstract and Static

You can define static or abstract methods or fields using the `{static}` or `{abstract}` modifier.

These modifiers can be used at the start or at the end of the line. You can also use `{classifier}` instead of `{static}`.

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

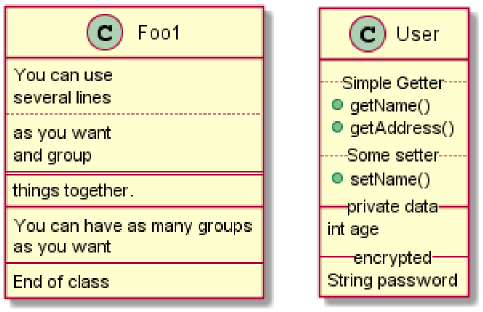


3.6 Advanced class body

By default, methods and fields are automatically regrouped by PlantUML. You can use separators to define your own way of ordering fields and methods. The following separators are possible : `--` `..` `==` `__`.

You can also use titles within the separators:

```
@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 and stereotypes

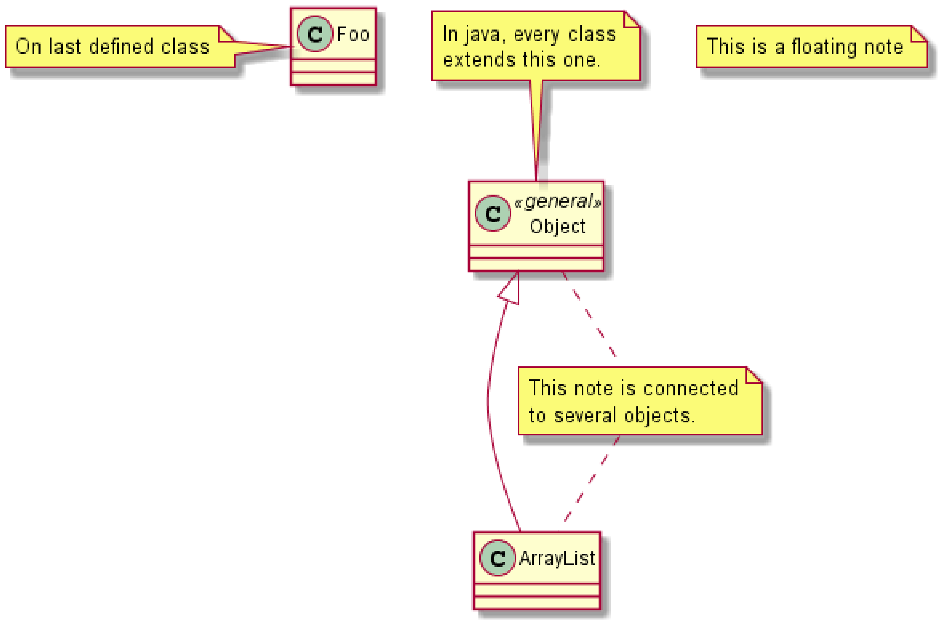
Stereotypes are defined with the `class` keyword, << and >>.

You can also define notes using `note left of` , `note right of` , `note top of` , `note bottom of` keywords.

You can also define a note on the last defined class using `note left`, `note right`, `note top`, `note bottom`.

A note can be also define alone with the `note` keywords, then linked to other objects using the `..` symbol.

```
@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
```


On last defined class
Foo
In java, every class extends this one.
This is a floating note
«general» Object
This note is connected to several objects.
ArrayList

3.8 More on notes

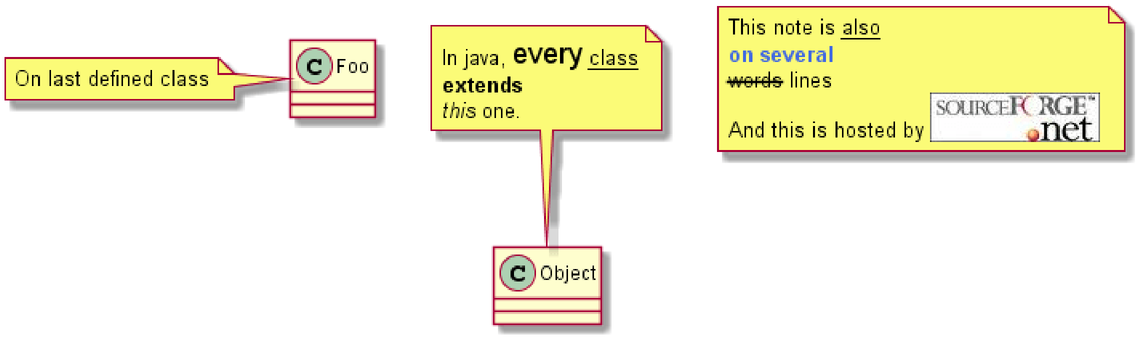
It is also possible to use few html tags like :

- `<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

You can also have a note on several lines.

You can also define a note on the last defined class using `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
```

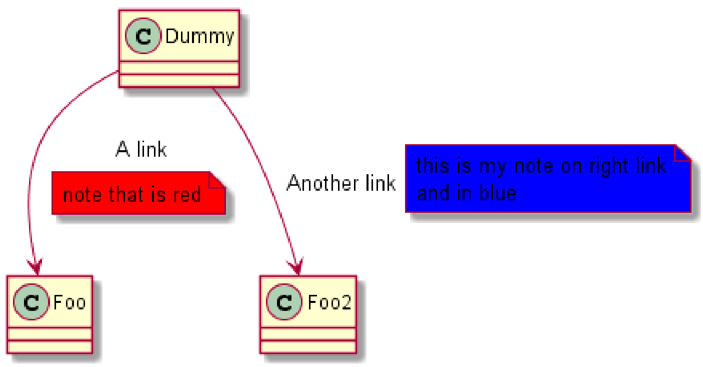


3.9 Note on links

It is possible to add a note on a link, just after the link definition, using `note on link`.

You can also use `note left on link`, `note right on link`, `note top on link`, `note bottom on link` if you want to change the relative position of the note with the label.

```
@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
```



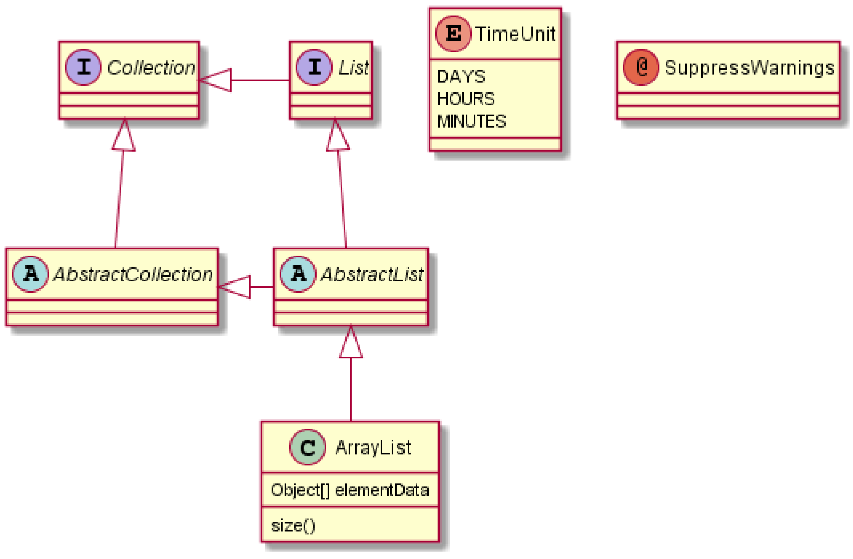
3.10 Abstract class and interface

You can declare a class as abstract using `abstract"` or `abstract class` keywords.

The class will be printed in *italic*.

You can use the `interface`, `annotation` and `enum` keywords too.

```
@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
```

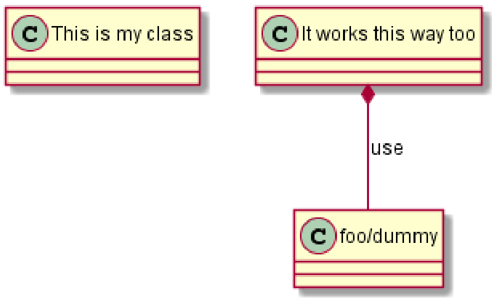


3.11 Using non-letters

If you want to use non-letters in the class (or enum...) display, you can either :

- Use the `as` keyword in the class definition
- Put quotes `""` around the class name

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

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



3.12 Hide attributes, methods...

You can parameterize the display of classes using the `hide/show` command.

The basic command is: `hide empty members`. This command will hide attributes or methods if they are empty.

Instead of `empty members`, you can use:

- `empty fields` or `empty attributes` for empty fields,
- `empty methods` for empty methods,
- `fields` or `attributes` which will hide fields, even if they are described,
- `methods` which will hide methods, even if they are described,
- `members` which will hide fields and methods, even if they are described,
- `circle` for the circled character in front of class name,

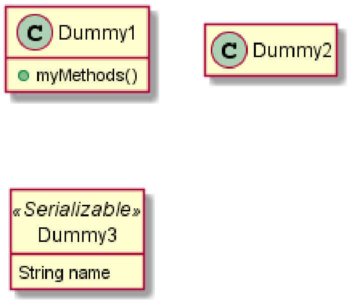
- `stereotype` for the stereotype.

You can also provide, just after the `hide` or `show` keyword:

- `class` for all classes,
- `interface` for all interfaces,
- `enum` for all enums,
- `<<foo1>>` for classes which are stereotyped with *foo1*,
- an existing class name.

You can use several `show/hide` commands to define rules and 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
```

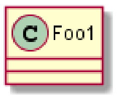


3.13 Hide classes

You can also use the `show/hide` commands to hide classes.

This may be useful if you define a large !included file, and if you want to hide come classes after file inclusion.

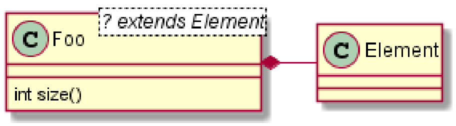
```
@startuml

class Foo1
class Foo2

Foo2 *-- Foo1

hide Foo2

@enduml
```



3.14 Use generics

You can also use bracket < and > to define generics usage in a class.

```
@startuml

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

@enduml
```



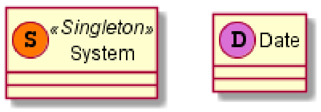
It is possible to disable this drawing using `skinparam genericDisplay old` command.

3.15 Specific Spot

Usually, a spotted character (C, I, E or A) is used for classes, interface, enum and abstract classes.

But you can define your own spot for a class when you define the stereotype, adding a single character and a color, like in this example:

```
@startuml

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



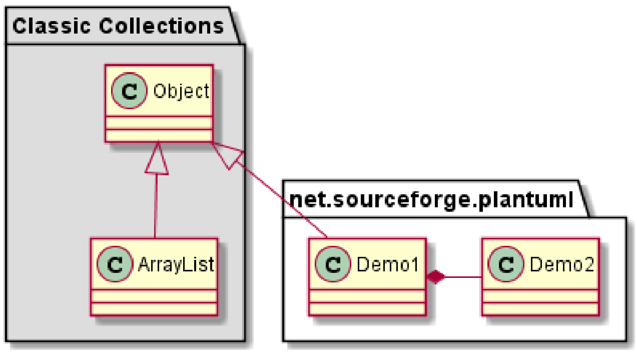
3.16 Packages

You can define a package using the `package` keyword, and optionally declare a background color for your package (Using a html color code or name).

Note that package definitions can be nested.

```
@startuml

package "Classic Collections" #DDDDDD {
  Object <|-- ArrayList
}
```

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

@enduml
```

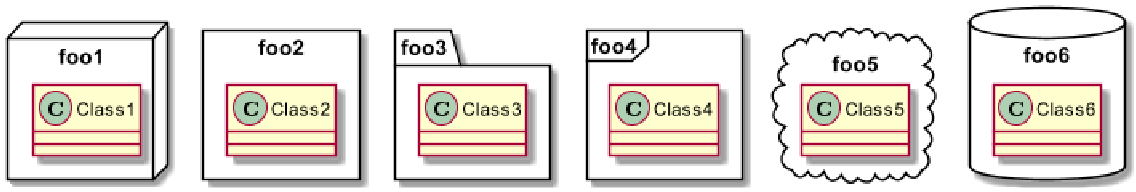


3.17 Packages style

There are different styles available for packages.

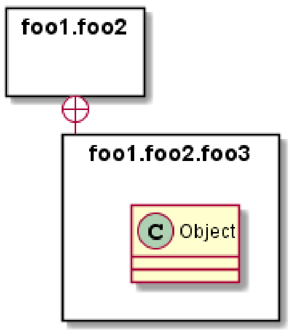
You can specify them either by setting a default style with the command : `skinparam packageStyle`, or by using a stereotype on the package:

```
@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
```



You can also define links between packages, like in the following example:

```
@startuml

skinparam packageStyle rectangle

package foo1.foo2 {
}

package foo1.foo2.foo3 {
  class Object
}

foo1.foo2 +-- foo1.foo2.foo3

@enduml
```



3.18 Namespaces

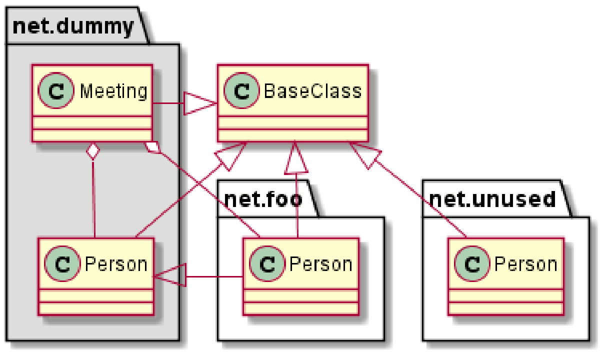
In packages, the name of a class is the unique identifier of this class. It means that you cannot have two classes with the very same name in different packages.

In that case, you should use namespaces instead of packages.

You can refer to classes from other namespaces by fully qualify them. Classes from the default namespace are qualified with a starting dot.

Note that you don't have to explicitly create namespace : a fully qualified class is automatically put in the right namespace.

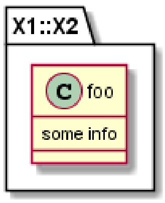
```
@startuml

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 Automatic namespace creation

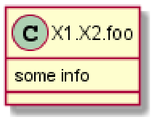
You can define another separator (other than the dot) using the command : `set namespaceSeparator ???`.

```
@startuml

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

@enduml
```



You can disable automatic package creation using the command `set namespaceSeparator none`.

```
@startuml

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

@enduml
```

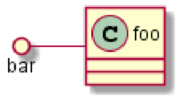


3.20 Lollipop interface

You can also define lollipops interface on classes, using the following syntax:

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

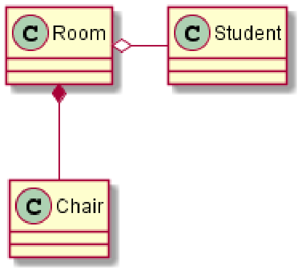
```
@startuml
class foo
bar ()- foo
@enduml
```



3.21 Changing arrows direction

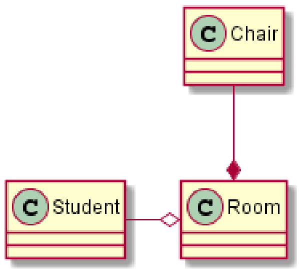
By default, links between classes have two dashes `--` and are vertically oriented. It is possible to use horizontal link by putting a single dash (or dot) like this:

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



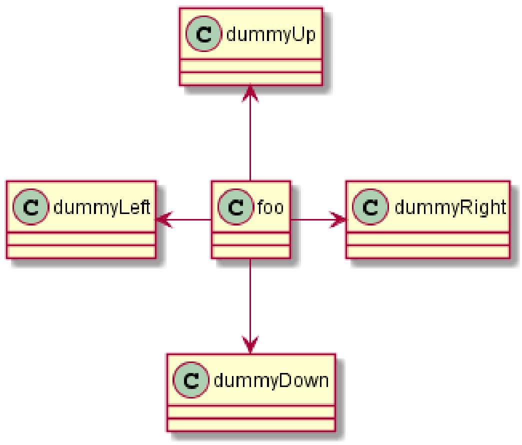
You can also change directions by reversing the link:

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



It is also possible to change arrow direction by adding `left`, `right`, `up` or `down` keywords inside the arrow:

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



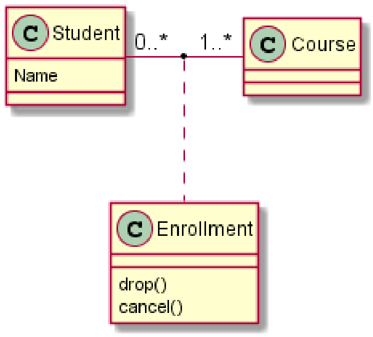
You can shorten the arrow by using only the first character of the direction (for example, `-d-` instead of `-down-`) or the two first characters (`-do-`).

Please note that you should not abuse this functionality : *Graphviz* gives usually good results without tweaking.

3.22 Association classes

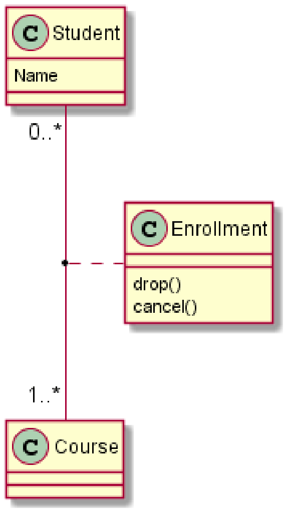
You can define *association class* after that a relation has been defined between two classes, like in this example:

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

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



You can define it in another direction:

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

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



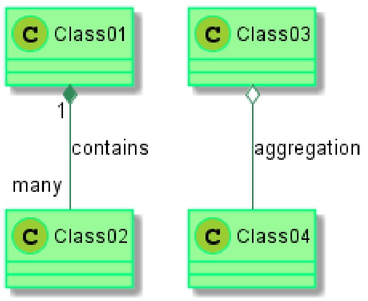
3.23 Skinparam

You can use the `skinparam` command to change colors and fonts for the drawing.

You can use this command :

- In the diagram definition, like any other commands,
- In an included file,
- In a configuration file, provided in the command line or the ANT task.

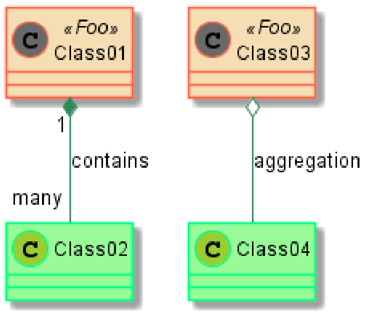
```
@startuml

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

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

Class03 o-- Class04 : aggregation

@enduml
```



3.24 Skinned Stereotypes

You can define specific color and fonts for stereotyped classes.

```
@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 Color gradient

It's possible to declare individual color for classes or note using the # notation.

You can use either standard color name or RGB code.

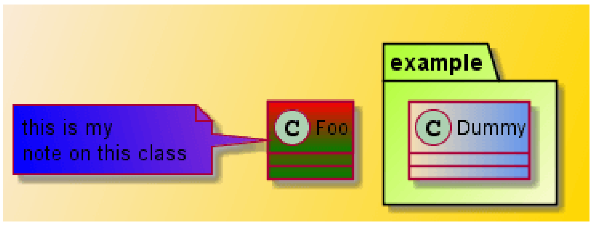
You can also use color gradient in background, with the following syntax: two colors names separated either by:

- |,
- /,
- \,
- or -

depending the direction of the gradient.

For example, you could have:

```
@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
```



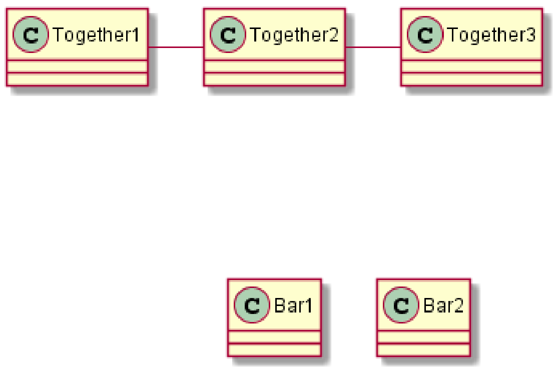
3.26 Help on layout

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 Splitting large files

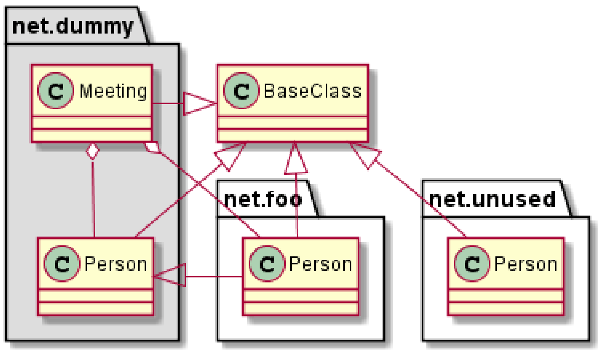
Sometimes, you will get some very large image files.

You can use the `page (hpages)x(vpages)` command to split the generated image into several files :

`hpages` is a number that indicated the number of horizontal pages, and `vpages` is a number that indicated the number of vertical pages.

You can also use some specific skinparam settings to put borders on splitted pages (see example).

```
@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