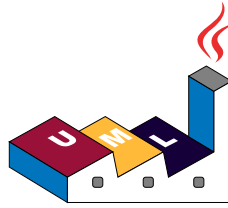


Dessiner de l'UML avec PlantUML



Guide de référence du langage (Version 6085)

PlantUML est un projet Open Source qui permet de dessiner rapidement :

- des diagrammes de séquences,
- des diagrammes de cas d'utilisation,
- des diagrammes de classes,
- des diagrammes d'activités,
- des diagrammes de composants,
- des diagrammes d'états,
- des diagrammes d'objets.

Les diagrammes sont définis à l'aide d'un langage simple et intuitif.

1 Diagramme de séquence

1.1 Exemples de base

Chaque description de diagramme doit commencer par `@startuml` et finir par `@enduml`.

Le symbole `"->"` est utilisé pour dessiner un message entre deux participants.

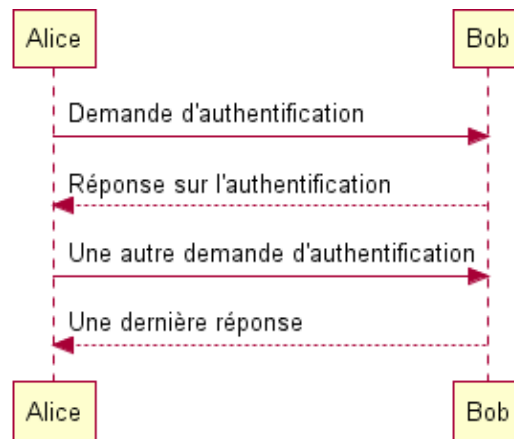
Les participants n'ont pas besoin d'être explicitement déclarés.

Pour avoir une flèche en pointillés, il faut utiliser `"-->"`.

Il est aussi possible d'utiliser `"<-"` et `"<--"`. Cela ne change pas le dessin, mais cela peut améliorer la lisibilité du texte source.

Exemple :

```
@startuml
Alice -> Bob: Demande d'authentification
Bob --> Alice: Réponse sur l'authentification
Alice -> Bob: Une autre demande d'authentification
Alice <-- Bob: Une dernière réponse
@enduml
```



1.2 Déclaration de participants

Il est possible de changer l'ordre des participants à l'aide du mot clé `participant`.

Pour déclarer un acteur à la place d'un participant, il faut utiliser le mot clé `actor`.

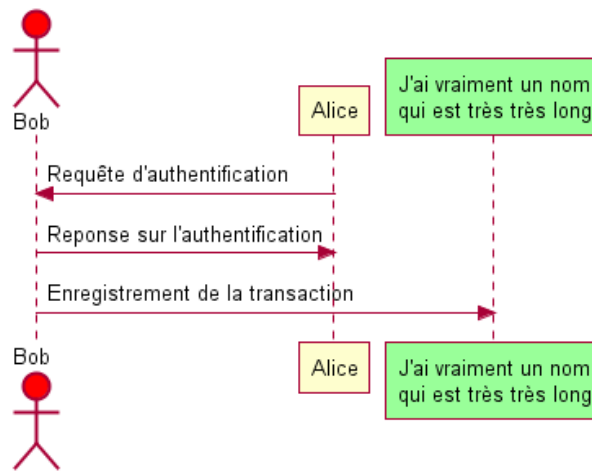
On peut aussi utiliser un nom court à l'aide grâce au mot-clé `as`.

La couleur de fond d'un acteur ou d'un participant peut être définie avec son code ou son nom HTML.

Notez que tout ce qui commence par une apostrophe ' est ignoré.

```
@startuml
actor Bob #red
' La seule différence entre acteur et participant est le rendu graphique
participant Alice
participant "J'ai vraiment un nom\nqui est très très long" as L #99FF99

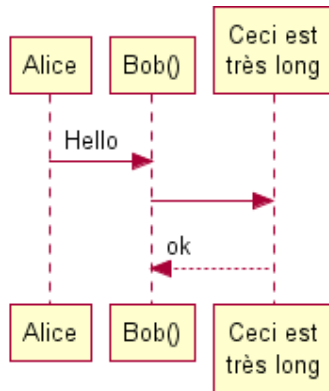
Alice->>Bob: Requête d'authentification
Bob->>Alice: Reponse sur l'authentification
Bob->>L: Enregistrement de la transaction
@enduml
```



1.3 Caractères non alphanumérique dans les participants

Si vous voulez mettre des caractères non alphanumériques, il est possible d'utiliser des guillemets. Et on peut utiliser le mot clé `as` pour définir un alias pour ces participants.

```
@startuml
Alice -> "Bob()" : Hello
"Bob()" -> "Ceci est\ntrès long" as Long
' Il est aussi possible de faire
' "Bob()" -> Long as "Ceci est\ntrès long"
Long --> "Bob()" : ok
@enduml
```

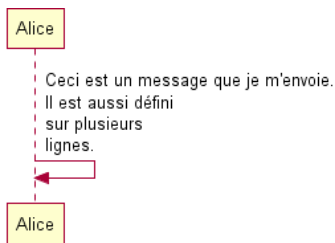


1.4 Message à soi-même

Un participant peut très bien s'envoyer un message.

Il est possible de mettre un message sur plusieurs lignes grâce à `\n`.

```
@startuml
Alice->>Alice: Ceci est un message que je m'envoie.\nIl est aussi défini\nsur plusieurs\nlignes.
@enduml
```

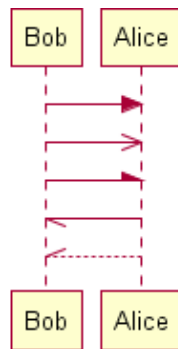


1.5 Autre style de flèches

Vous pouvez changer les flèches de plusieurs façon :

- Utiliser \ ou / à la place de < ou > pour avoir seulement la partie supérieure ou inférieure de la flèche.
- Doubler un des caractères (par exemple, >> ou //) pour avoir une flèche plus fine.
- Utiliser -- à la place de - pour avoir des pointillés.

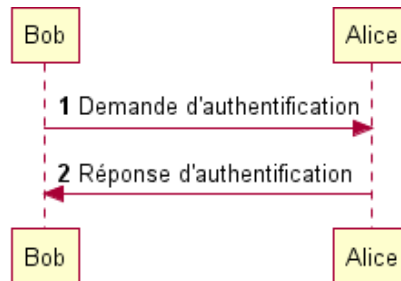
```
@startuml
Bob -> Alice
Bob ->> Alice
Bob -\ Alice
Bob \- Alice
Bob //-- Alice
@enduml
```



1.6 Numérotation automatique

Le mot clé `autonumber` est utilisé pour ajouter automatiquement des numéros aux messages.

```
@startuml
autonumber
Bob -> Alice : Demande d'authentification
Bob <- Alice : Réponse d'authentification
@enduml
```



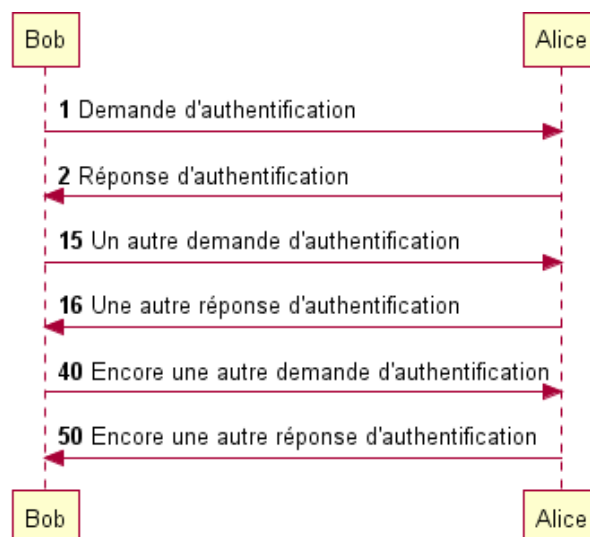
Vous pouvez spécifier

- Un numéro de départ avec `"autonumber 'départ'"`,
- Un incrément avec `"autonumber 'départ' 'incrément'"`

```
@startuml
autonumber
Bob -> Alice : Demande d'authentification
Bob <- Alice : Réponse d'authentification

autonumber 15
Bob -> Alice : Un autre demande d'authentification
Bob <- Alice : Une autre réponse d'authentification

autonumber 40 10
Bob -> Alice : Encore une autre demande d'authentification
Bob <- Alice : Encore une autre réponse d'authentification
@enduml
```



Vous pouvez aussi formater le nombre en définissant le format entre guillemets. Le formatage est fait pas la classe `DecimalFormat` ('0' signifie un chiffre, '#' signifie un chiffre ou zéro si absent).

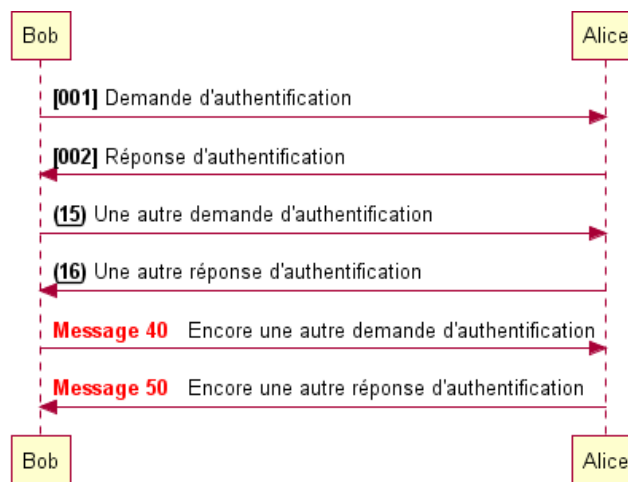
Il est possible de mettre des tags HTML dans le format.

```
@startuml
autonumber "<b>[000]"
Bob -> Alice : Demande d'authentification
Bob <- Alice : Réponse d'authentification

autonumber 15 "<b>(<u>##</u>)"
Bob -> Alice : Une autre demande d'authentification
Bob <- Alice : Une autre réponse d'authentification

autonumber 40 10 "<font color=red><b>Message 0  "
Bob -> Alice : Encore une autre demande d'authentification
Bob <- Alice : Encore une autre réponse d'authentification

@enduml
```



1.7 Titre

Le mot clé `title` est utilisé pour mettre un titre.

```
@startuml
```

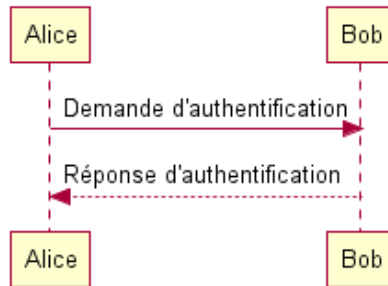
```
title Exemple de communication simple
```

```
Alice -> Bob: Demande d'authentification
```

```
Bob --> Alice: Réponse d'authentification
```

```
@enduml
```

Exemple de communication simple



1.8 Découper un diagramme

Le mot clé `newpage` est utilisé pour découper un diagramme en plusieurs images. Vous pouvez mettre un titre pour la nouvelle page juste après le mot clé `newpage`. Ceci est très pratique pour mettre de très longs diagrammes sur plusieurs pages.

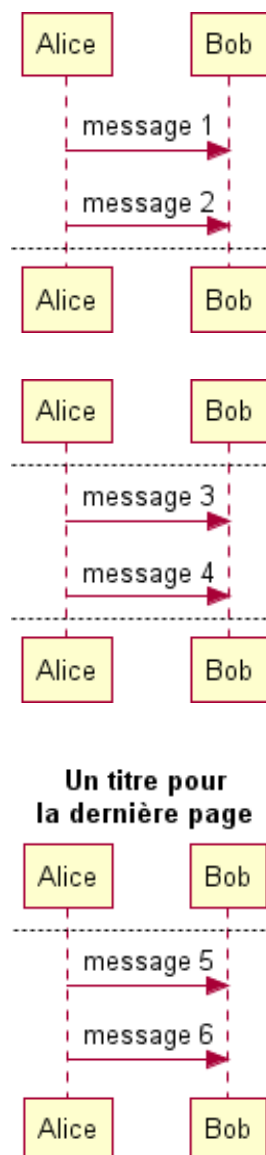
```
@startuml
Alice -> Bob : message 1
Alice -> Bob : message 2

newpage

Alice -> Bob : message 3
Alice -> Bob : message 4

newpage Un titre pour\nla dernière page

Alice -> Bob : message 5
Alice -> Bob : message 6
@enduml
```



1.9 Regrouper les messages

Il est possible de regrouper les messages à l'aide d'un des mot clés suivants :

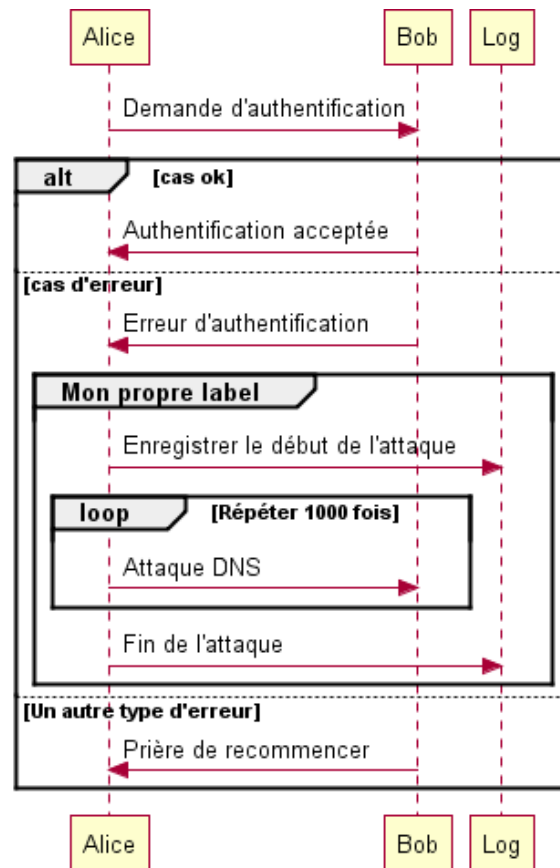
- alt/else
- opt
- loop
- par
- break
- critical
- group, suivi par le texte à afficher

Il est aussi possible de mettre un texte à afficher dans l'entête. Le mot-clé **end** est utilisé pour fermer le groupe. Il est aussi possible d'imbriquer les groupes.

```

@startuml
Alice -> Bob: Demande d'authentification
alt cas ok
  Bob -> Alice: Authentification acceptée
else cas d'erreur
  Bob -> Alice: Erreur d'authentification
  group Mon propre label
    Alice -> Log : Enregistrer le début de l'attaque
    loop Répéter 1000 fois
      Alice -> Bob: Attaque DNS
    end
    Alice -> Log : Fin de l'attaque
  end
else Un autre type d'erreur
  Bob -> Alice: Prière de recommencer
end
@enduml

```



1.10 Note sur les messages

Il est possible de mettre des notes sur les messages en utilisant :

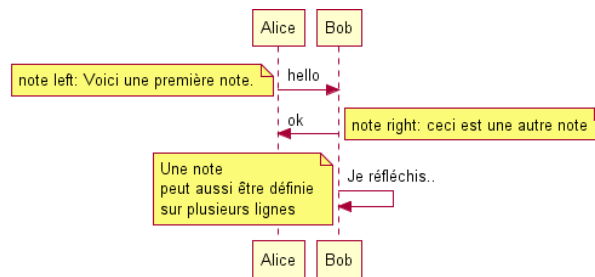
- `note left` ou
- `note right` juste après le message.

Il est possible d'avoir une note sur plusieurs lignes avec le mot clé `end note`.

```
@startuml
Alice->Bob : hello
note left: note left: Voici une première note.

Bob->Alice : ok
note right: note right: ceci est une autre note

Bob->Bob : Je réfléchis..
note left
  Une note
  peut aussi être définie
  sur plusieurs lignes
end note
@enduml
```



1.11 Encore plus de notes

Il est aussi possible de mettre des notes placées par rapport aux participants.

- note left of,
- note right of,
- note over .

Il est aussi possible de faire ressortir une note en changeant sa couleur de fond. On peut aussi avoir des notes sur plusieurs lignes à l'aide du mot clé `end note`.

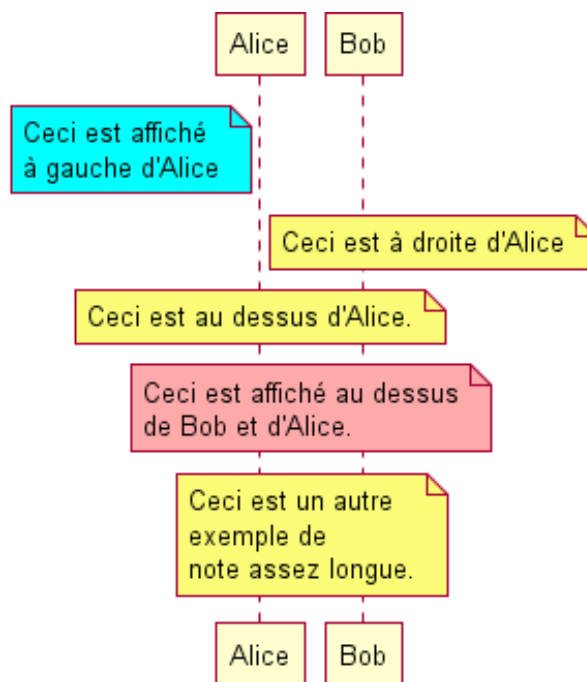
```
@startuml
participant Alice
participant Bob
note left of Alice #aqua
    Ceci est affiché
    à gauche d'Alice
end note

note right of Alice: Ceci est à droite d'Alice

note over Alice: Ceci est au dessus d'Alice.

note over Alice, Bob #FFAAAA: Ceci est affiché au dessus\nde Bob et d'Alice.

note over Bob, Alice
    Ceci est un autre
    exemple de
    note assez longue.
end note
@enduml
```



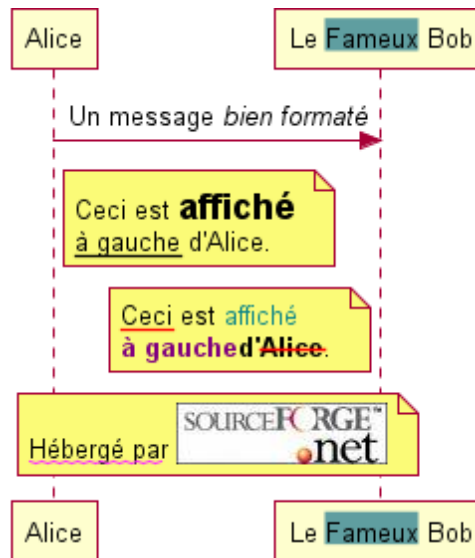
1.12 Formattage grâce au HTML

Il est possible d'utiliser quelques tags HTML :

- pour mettre en gras
- <u> ou <u:nomDeCouleur> pour souligner
- <i> pour l'italique
- <s> ou <s:nomDeCouleur> pour rayer un texte
- <w> ou <w:#AAAAAA> ou <w:nomDeCouleur> pour souligner avec des vagues.
- <color:#AAAAAA> ou <color:nomDeCouleur>
- <back:#AAAAAA> ou <back:nomDeCouleur> pour changer la couleur de fond
- <size:nn> pour changer la taille du texte
- <img:file> avec un fichier contenant une image.

```
@startuml
participant Alice
participant "Le <back:cadetblue>Fameux</back> Bob" as Bob

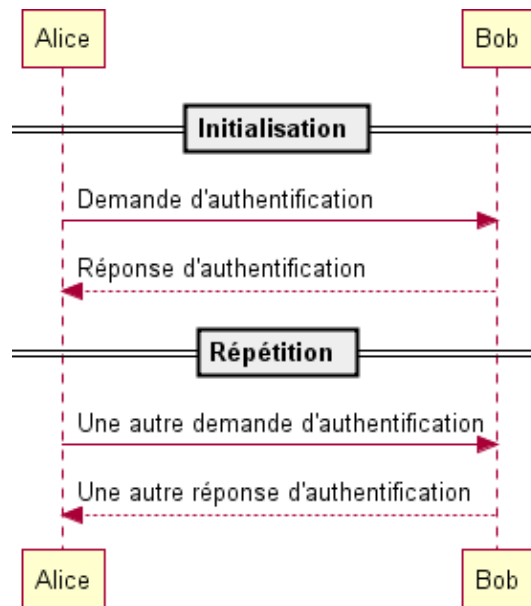
Alice -> Bob : Un message <i>bien formaté</i>
note right of Alice
    Ceci est <b><size:18>affiché</size></back>
    <u>à gauche</u> d'Alice.
end note
note left of Bob
    <u:red>Ceci</u> est <color:#118888>affiché</color>
    <b><color:purple>à gauche</color>d'<s:red>Alice</s></b>.
end note
note over Alice, Bob
    <w:#FF33FF>Hébergé par</w> <img:img/sourceforge.jpg>
end note
@enduml
```



1.13 Séparation

Si vous voulez, vous pouvez séparer le diagramme avec l'aide de "==" en étapes logiques..

```
@startuml
== Initialisation ==
Alice -> Bob: Demande d'authentification
Bob --> Alice: Réponse d'authentification
== Répétition ==
Alice -> Bob: Une autre demande d'authentification
Alice <-- Bob: Une autre réponse d'authentification
@enduml
```



1.14 Lignes de vie

Vous pouvez utiliser `activate` et `deactivate` pour marquer l'activation des participants.

Une fois qu'un participant est activé, sa ligne de vie apparaît.

Les ordres `activate` et `deactivate` s'applique sur le message situé juste avant.

Le mot clé `destroy` sert à montrer la fin de vie d'un participant.

```
@startuml
participant User

User -> A: DoWork
activate A

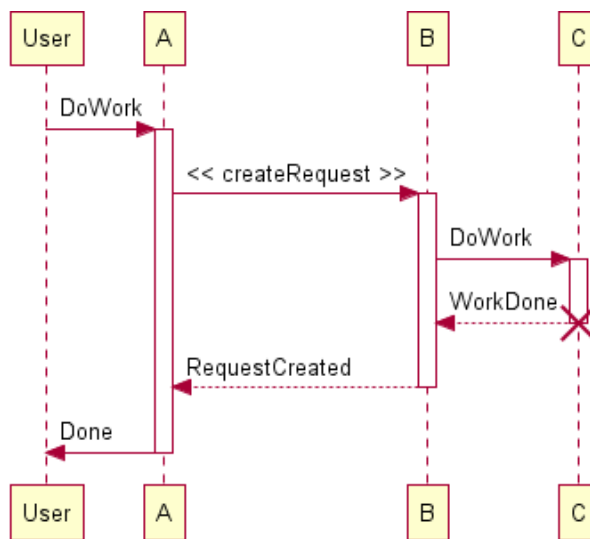
A -> B: << createRequest >>
activate B

B -> C: DoWork
activate C
C --> B: WorkDone
destroy C

B --> A: RequestCreated
deactivate B

A -> User: Done
deactivate A

@enduml
```



Les lignes de vie peuvent être imbriquées, et il est possible de les colorer.

```

@startuml
participant User

User -> A: DoWork
activate A #FFBBBB

A -> A: Internal call
activate A #DarkSalmon

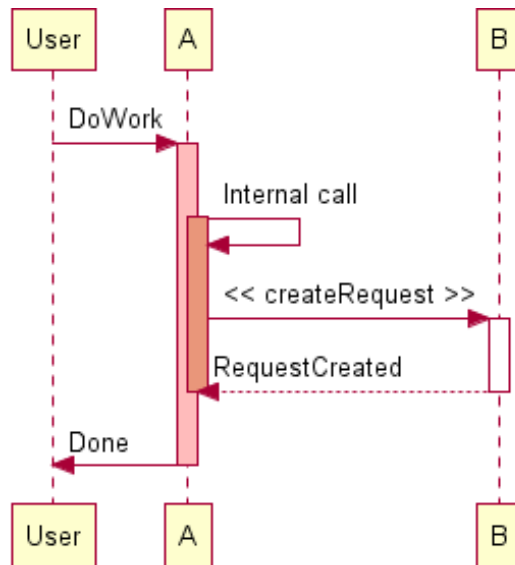
A -> B: << createRequest >>
activate B

B --> A: RequestCreated
deactivate B
deactivate A

A -> User: Done
deactivate A

@enduml

```



1.15 Création de participants.

Vous pouvez utiliser le mot clé **create** juste avant la première réception d'un message pour montrer que le message en question est une *création* d'un nouvelle objet.

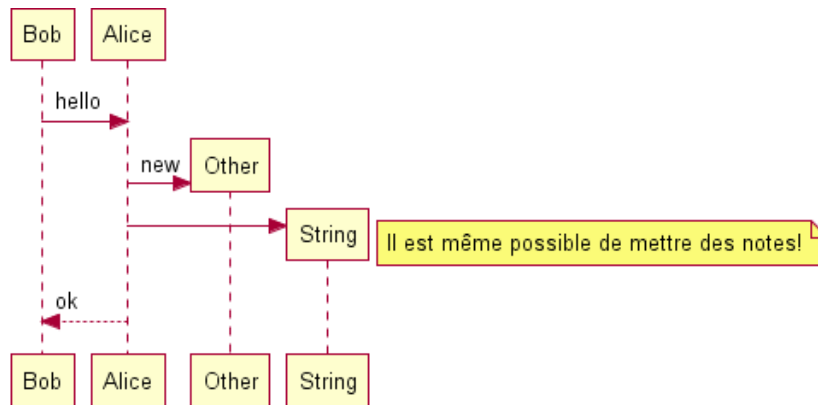
```
@startuml
Bob -> Alice : hello

create Other
Alice -> Other : new

create String
Alice -> String : note right : Il est même possible de mettre des notes!

Alice --> Bob : ok

@enduml
```



1.16 Messages entrant et sortant

Vous pouvez utiliser des flèches qui viennent de la droite ou de la gauche pour dessiner un sous-diagramme.

Il faut utiliser des crochets pour indiquer la gauche "]" ou la droite "[" du diagramme.

```

@startuml
[-> A: DoWork

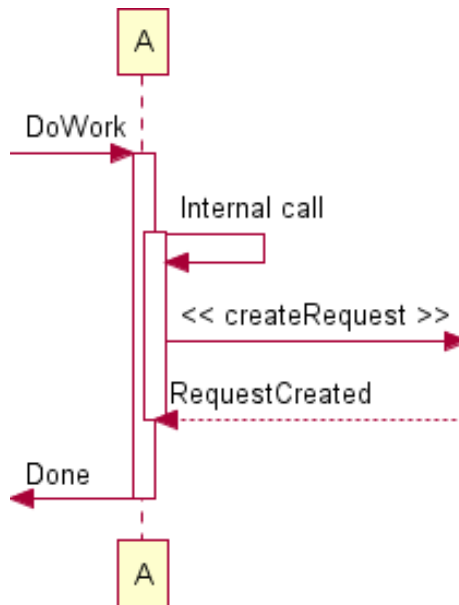
activate A

A -> A: Internal call
activate A

A ->] : << createRequest >>

A<--] : RequestCreated
deactivate A
[<- A: Done
deactivate A
@enduml

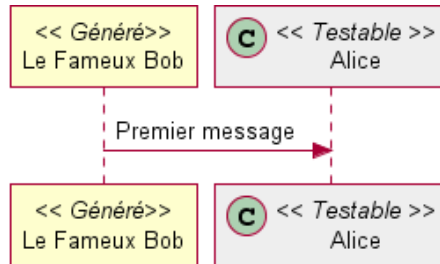
```



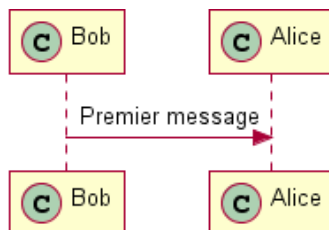
1.17 Stéréotypes et décoration

Il est possible de rajouter un stéréotype aux participants en utilisant "<<" et ">>". Dans un stéréotype, on peut avoir un caractère entouré d'un cercle en utilisant la syntaxe "(X,couleur)".

```
@startuml
participant "Le Fameux Bob" as Bob << Généré>>
participant Alice << (C,#ADD1B2) Testable >> #EEEEEE
Bob->>Alice: Premier message
@enduml
```



```
@startuml
participant Bob << (C,#ADD1B2) >>
participant Alice << (C,#ADD1B2) >>
Bob->>Alice: Premier message
@enduml
```

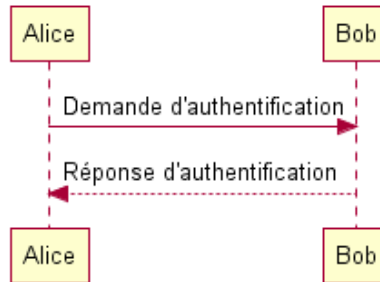


1.18 Plus d'information sur les titres

Il est possible d'utiliser des tags HTML dans les titres.

```
@startuml
title <u>Simple</u> exemple de communication
Alice -> Bob: Demande d'authentification
Bob --> Alice: Réponse d'authentification
@enduml
```

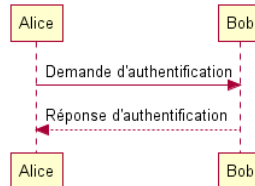
Simple exemple de communication



Vous pouvez mettre des retours à la ligne en utilisant `\n` dans la description.

```
@startuml
title <u>Simple</u> exemple de communication sur plusieurs lignes
Alice -> Bob: Demande d'authentification
Bob --> Alice: Réponse d'authentification
@enduml
```

Simple exemple de communication sur plusieurs lignes



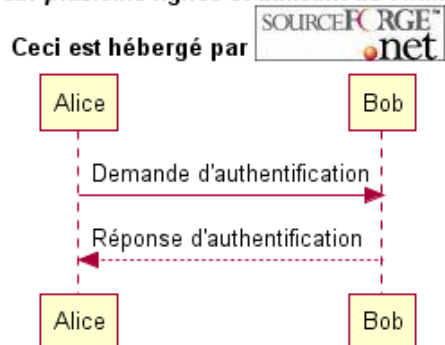
Vous pouvez aussi mettre un titre sur plusieurs lignes à l'aide des mots-clé `title` et `end title`.

```
@startuml
title
  <u>Simple</u> exemple de communication
  sur <i>plusieurs</i> lignes et utilisant de l'<font color=red>html</font>
  Ceci est hébergé par <img src=img/sourceforge.jpg>
end title

Alice -> Bob: Demande d'authentification
Bob --> Alice: Réponse d'authentification

@enduml
```

**Simple exemple de communication
sur *plusieurs* lignes et utilisant de l'**html****



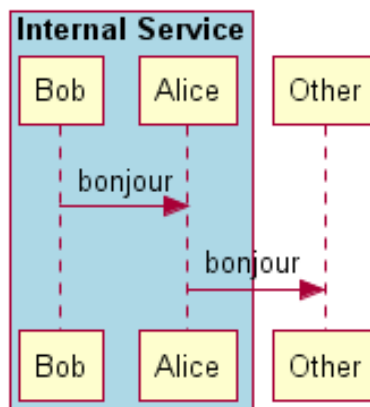
1.19 Cadre pour les participants

Il est possible de dessiner un cadre autour de certains participants, en utilisant les commandes `box` et `end box`.

Vous pouvez ajouter un titre ou bien une couleur de fond après le mot-clé `box`.

```
@startuml
box "Internal Service" #LightBlue
    participant Bob
    participant Alice
end box
participant Other

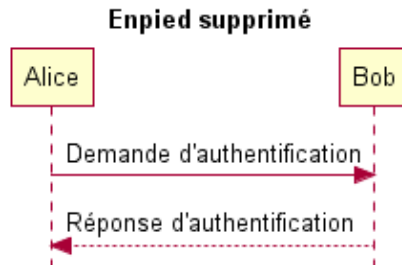
Bob -> Alice : bonjour
Alice -> Other : bonjour
@enduml
```



1.20 Supprimer les en-pieds

Vous pouvez utiliser le mot-clé `hide footbox` pour supprimer la partie basse du diagramme.

```
@startuml
hide footbox
title Empied supprimé
Alice -> Bob: Demande d'authentification
Bob --> Alice: Réponse d'authentification
@enduml
```



1.21 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 autres commandes,
- Dans un fichier inclus,
- Dans un fichier de configuration, donné à la ligne de commande ou à la tâche ANT.

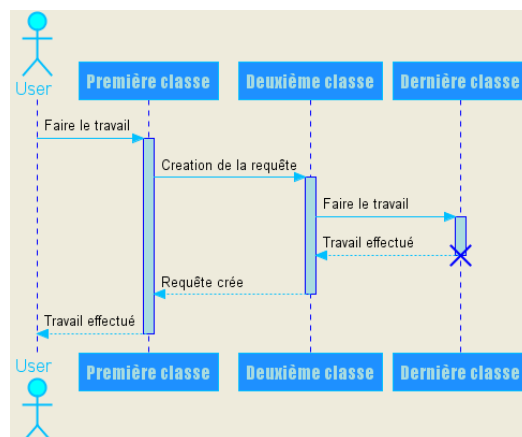
```
@startuml
skinparam backgroundColor #EEEEBC

skinparam sequence {
    ArrowColor DeepSkyBlue
    ActorBorderColor DeepSkyBlue
    LifeLineBorderColor blue
    LifeLineBackgroundColor #A9DCDF

    ParticipantBorderColor DeepSkyBlue
    ParticipantBackgroundColor DodgerBlue
    ParticipantFontName Impact
    ParticipantFontSize 17
    ParticipantFontColor #A9DCDF

    ActorBackgroundColor aqua
    ActorFontColor DeepSkyBlue
    ActorFontSize 17
    ActorFontName Aapex
}

actor User
participant "Première classe" as A
participant "Deuxième classe" as B
participant "Dernière classe" as C
User -> A: Faire le travail
activate A
A -> B: Creation de la requête
activate B
B -> C: Faire le travail
activate C
C --> B: Travail effectué
destroy C
B --> A: Requête crée
deactivate B
A --> User: Travail effectué
deactivate A
@enduml
```



1.22 Thème

Le mot-clé `skin` sert à changer l'apparence du diagramme généré. Pour l'instant, il y a deux thèmes différents (*Rose*, qui est le thème par défaut, et *BlueModern*), mais il est possible d'écrire votre propre thème.

```
@startuml
skin BlueModern

actor User
participant "First Class" as A
participant "Second Class" as B
participant "Last Class" as C

User -> A: DoWork
activate A

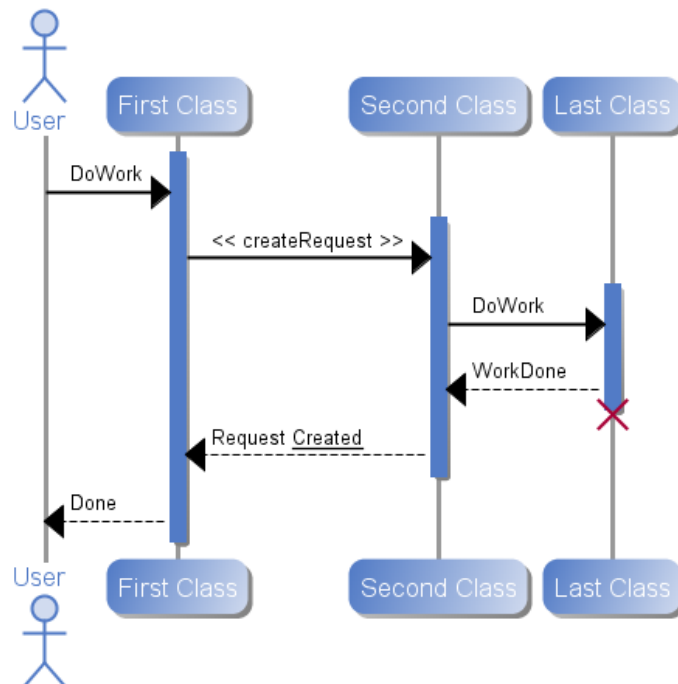
A -> B: << createRequest >>
activate B

B -> C: DoWork
activate C
C --> B: WorkDone
destroy C

B --> A: Request <u>Created</u>
deactivate B

A --> User: Done
deactivate A

@enduml
```



2 Diagramme de cas d'utilisation

2.1 Cas d'utilisation

Les cas d'utilisation sont mis entre parenthèses (car deux parenthèses forment un ovale).

Vous pouvez aussi utiliser le mot-clé `usecase` pour définir un cas d'utilisation.

Et vous pouvez définir un alias avec le mot-clé `as`. Cet alias sera ensuite utilisé lors de la définition des relations.

```
@startuml
```

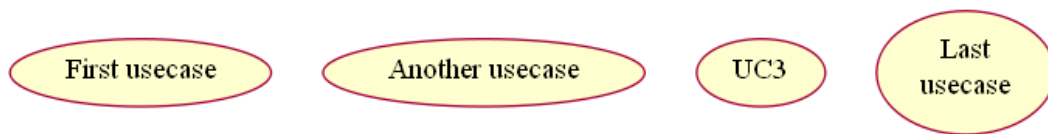
```
(First usecase)
```

```
(Another usecase) as (UC2)
```

```
usecase UC3
```

```
usecase (Last\nusecase) as UC4
```

```
@enduml
```



2.2 Acteurs

Un Acteur est encadré par des deux points.

Vous pouvez aussi utiliser le mot-clé `actor` pour définir un acteur.

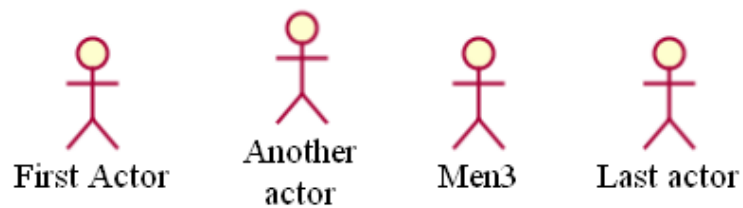
Et vous pouvez définir un alias avec le mot-clé `as`. Cet alias sera ensuite utilisé lors de la définition des relations.

Nous verrons que la définition des acteurs est optionnelle.

```
@startuml
```

```
:First Actor:  
:Another\nactor: as Men2  
actor Men3  
actor :Last actor: as Men4
```

```
@enduml
```



2.3 Exemples très simples

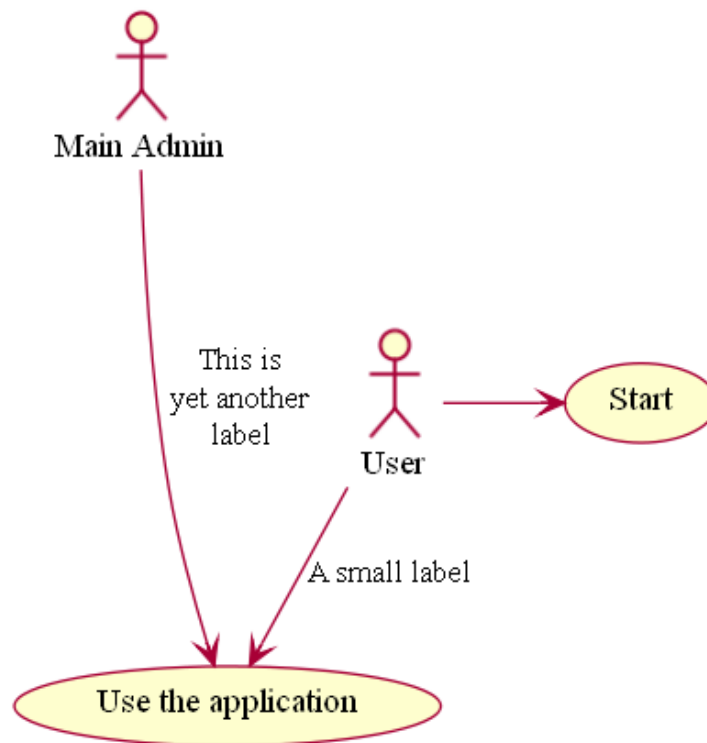
Pour lier les acteurs et les cas d'utilisation, la flèche "-->" est utilisée.

Plus il y a de tirets "-" dans la flèche, plus elle sera longue.


You can add a label on the arrow, by adding a ":" character in the arrow definition.


In this example, you see that *User* has not been defined before, and is implicitly defined as an actor.

```
@startuml
User -> (Start)
User --> (Use the application) : A small label
:Main Admin: ---> (Use the application) : This is\nyet another\nlabel
@enduml
```



2.4 Extension

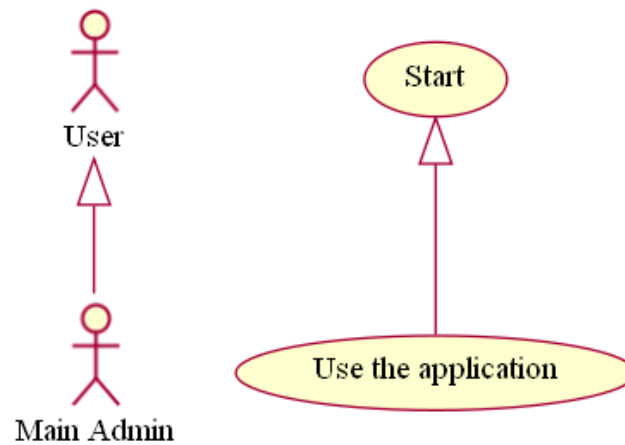
If one actor/use case extends another one, you can use the symbol <|-- (which stands for .

As for smiley, when you turn your head, you will see the symbol .

```
@startuml
:Main Admin: as Admin
(Use the application) as (Use)

User <|-- Admin
(Start) <|-- (Use)

@enduml
```



2.5 Using notes

You can use the :

- note left of,
- note right of,
- note top of,
- note bottom of

keywords to define notes related to a single object. A note can be also define alone with the note keywords, then linked to other objects using the .. symbol.

```
@startuml
:Main Admin: as Admin
(Use the application) as (Use)

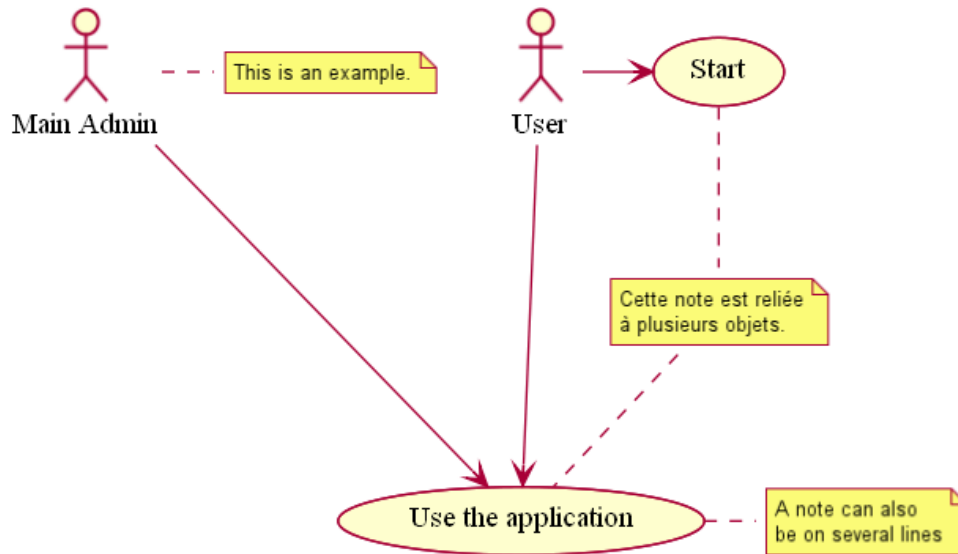
User -> (Start)
User --> (Use)

Admin ---> (Use)

note right of Admin : This is an example.

note right of (Use)
  A note can also
  be on several lines
end note

note "Cette note est reliée\nà plusieurs objets." as N2
(Start) .. N2
N2 .. (Use)
@enduml
```



2.6 Stereotypes

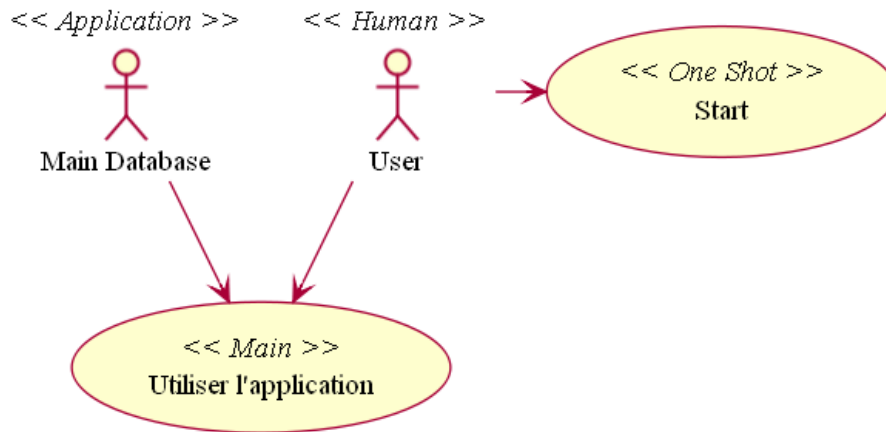
You can add stereotypes while defining actors and use cases using "<<" and ">>".

```
@startuml
User << Human >>
:Main Database: as MySQL << Application >>
(Start) << One Shot >>
(Utiliser l'application) as (Use) << Main >>

User -> (Start)
User --> (Use)

MySQL --> (Use)

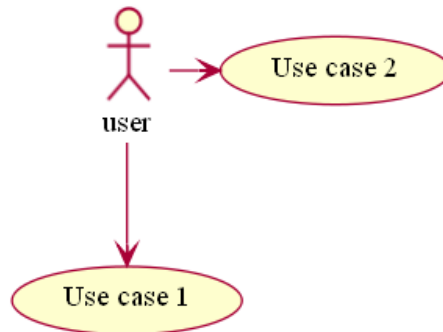
@enduml
```



2.7 Changing arrows direction

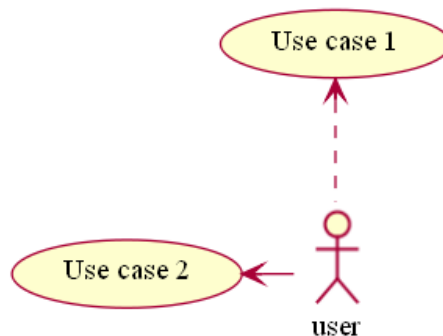
By default, links between classes have two dashes -- and are vertically oriented. It is possible to use horizontal link by putting a single dash (or dot) like this :

```
@startuml
:user: --> (Use case 1)
:user: -> (Use case 2)
@enduml
```



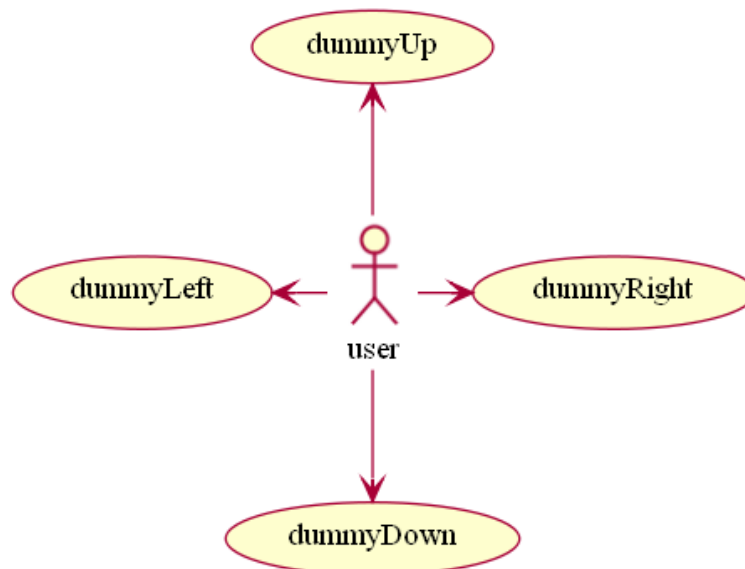
You can also change directions by reversing the link :

```
@startuml
(Use case 1) <.. :user:
(Use case 2) <- :user:
@enduml
```



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

```
@startuml
:user: -left-> (dummyLeft)
:user: -right-> (dummyRight)
:user: -up-> (dummyUp)
:user: -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 fonctionnality : *GraphViz* gives usually good results without tweaking.

2.8 Title the diagram

The `title` keywords is used to put a title.

You can use `title` and `end title` keywords for a longer title, as in sequence diagrams.

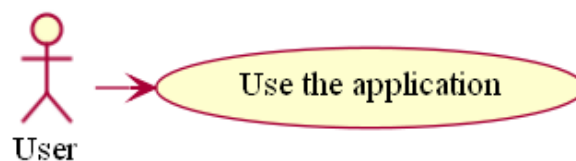
```

@startuml
title Simple <b>Usecase</b>\nwith one actor

usecase (Use the application) as (Use)
User -> (Use)

@enduml
  
```

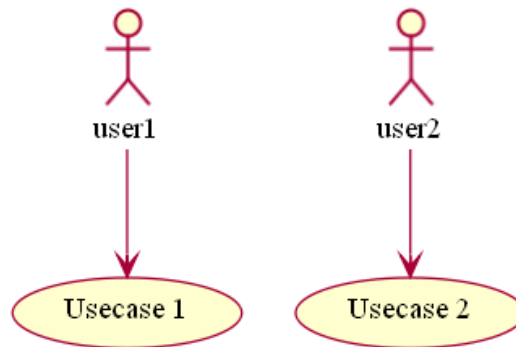
Simple Usecase with one actor



2.9 Left to right direction

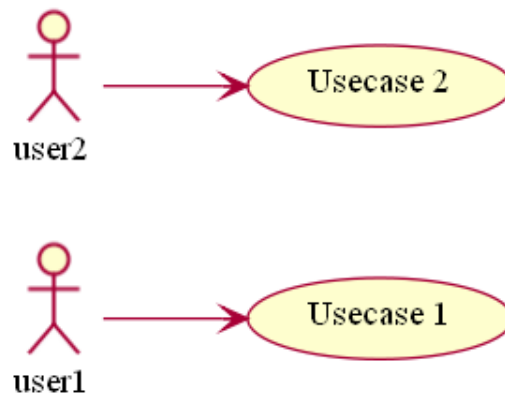
The general default behaviour when building diagram is top to bottom.

```
@startuml
'par défaut
top to bottom direction
user1 --> (Usecase 1)
user2 --> (Usecase 2)
@enduml
```



You may change to left to right using the `left to right direction` command. The result is often better with this direction.

```
@startuml
left to right direction
user1 --> (Usecase 1)
user2 --> (Usecase 2)
@enduml
```



2.10 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 usecase {
  BackgroundColor DarkSeaGreen
  BorderColor DarkSlateGray

  BackgroundColor<< Main >> YellowGreen
  BorderColor<< Main >> YellowGreen

  ArrowColor Olive
  ActorBorderColor black
  ActorFontName Courier

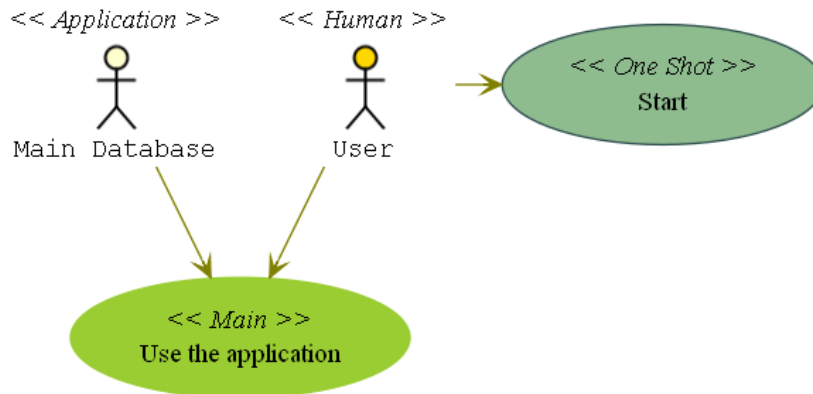
  ActorBackgroundColor<< Human >> Gold
}

User << Human >>
:Main Database: as MySql << Application >>
(Start) << One Shot >>
(Use the application) as (Use) << Main >>

User -> (Start)
User --> (Use)

MySql --> (Use)

@enduml
```



3 Diagramme de classes

3.1 Relations entre classes

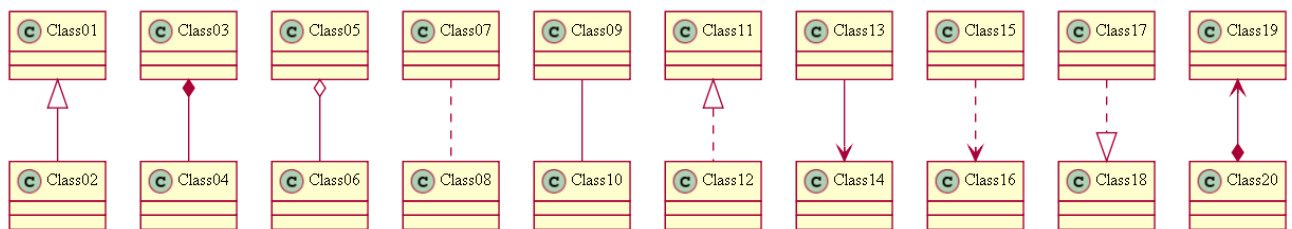
Les relations entre les classes sont définies en utilisant les symboles suivants :

Extension	< --	
Composition	*--	
Agrégation	o--	

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

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

```
@startuml
Class01 <|-- Class02
Class03 *-- Class04
Class05 o-- Class06
Class07 .. Class08
Class09 -- Class10
Class11 <|.. Class12
Class13 --> Class14
Class15 ..> Class16
Class17 ..|> Class18
Class19 <--* Class20
@enduml
```

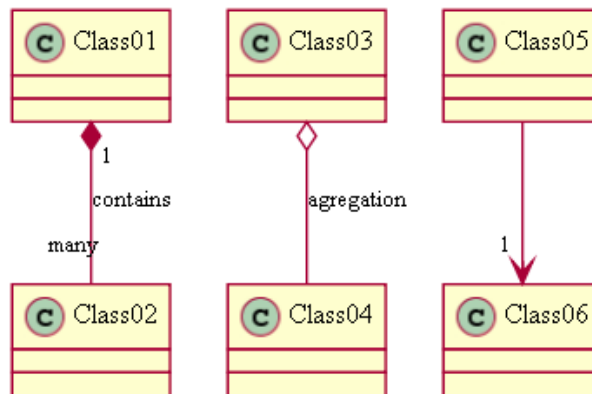


3.2 Label on relations

It is possible to 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 : agregation
Class05 --> "1" Class06
@enduml
```



3.3 Adding methods

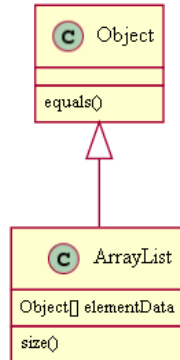
To declare fields and methods, you can use the symbol ":" followed by the field's or method's name.

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

```
@startuml
Object <|-- ArrayList

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

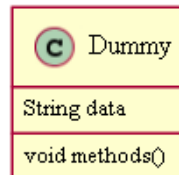
@enduml
```



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

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

@enduml
```

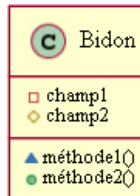


3.4 Defining visibility

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

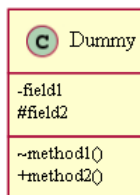
-	□	■	privé
#	◇	◆	protected
~	△	▲	package private
+	○	●	publique

```
@startuml
class Bidon{
  -champ1
  #champ2
  ~methode1()
  +methode2()
}
@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 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.

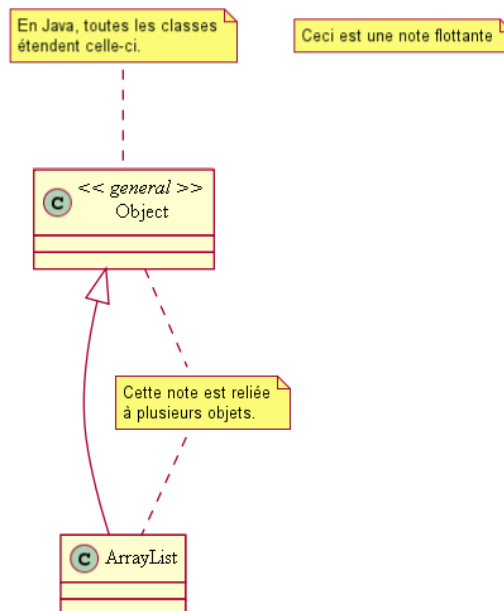
A note can be also define alone with the `note` keywords, then linked to other objects using the `..` symbol.

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

note top of Object : En Java, toutes les classes\nétendent celle-ci.

note "Ceci est une note flottante" as N1
note "Cette note est reliée\nà plusieurs objets." as N2
Object .. N2
N2 .. ArrayList

@enduml
```



3.6 Encore des notes

Il est possible d'utiliser quelques tag HTML comme :

-
 - <u>
 - <i>
 - <s>, , <strike>
 - ou
 - <color:#AAAAAA> ou <color:colorName>
 - <size:nn> pour changer la taille de la police
 - ou <img:file> : le fichier doit être accessible
- On peut aussi avoir une note sur plusieurs lignes.

```
@startuml
```

```
note top of Object
```

```
  In java, every <u>class</u>
```

```
  <b>extends</b>
```

```
  <i>this</i> one.
```

```
end note
```

```
note as N1
```

```
  This <size:10>note</size> is <u>also</u>
```

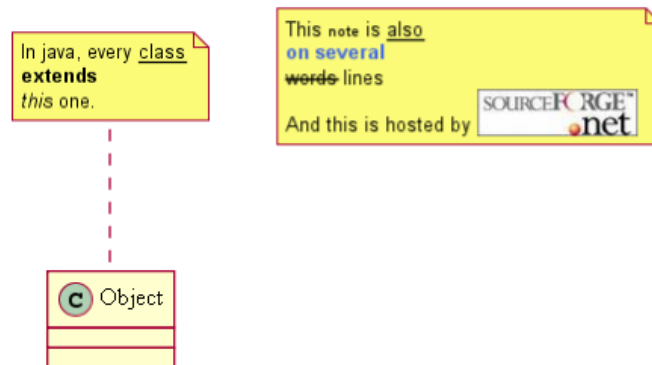
```
  <b><color:royalBlue>on several</color>
```

```
  <s>words</s> lines
```

```
  And this is hosted by <img:img/sourceforge.jpg>
```

```
end note
```

```
@enduml
```



3.7 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` and `enum` keywords too.

```
@startuml
abstract class AbstractList
abstract AbstractCollection
interface List
interface Collection

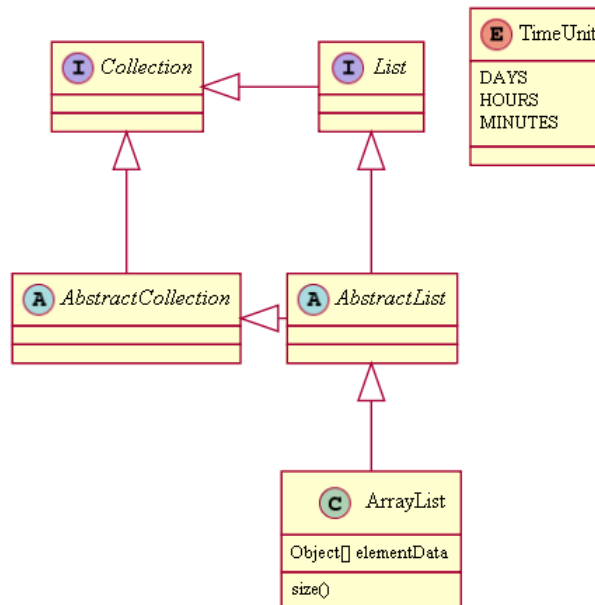
List <|-- AbstractList
Collection <|-- AbstractCollection

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

ArrayList : Object[] elementData
ArrayList : size()

enum TimeUnit
TimeUnit : DAYS
TimeUnit : HOURS
TimeUnit : MINUTES

@enduml
```



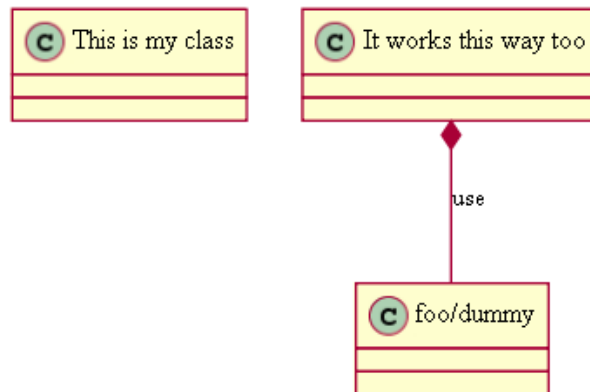
3.8 Using non-letters

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
```



3.9 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.

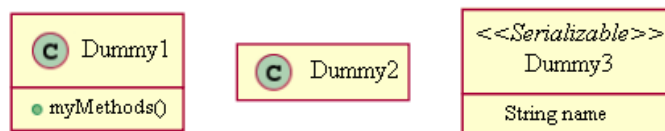
Vous pouvez utiliser plusieurs commandes `show/hide` pour définir des règles et des exceptions.

```
@startuml
class Dummy1 {
+myMethods()
}

class Dummy2 {
+hiddenMethod()
}

class Dummy3 <<Serializable>> {
String name
}

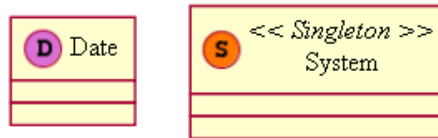
hide members
hide <<Serializable>> circle
show Dummy1 method
show <<Serializable>> fields
@enduml
```



3.10 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
```



3.11 Packages

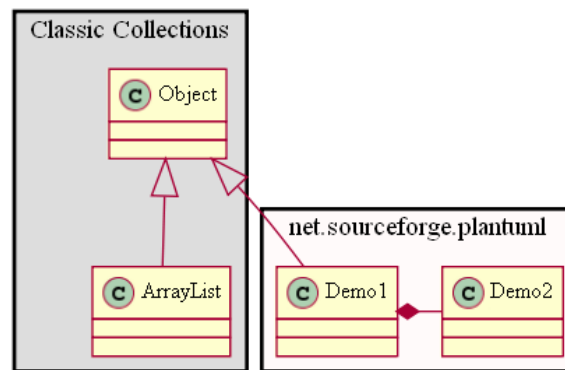
You can define a package using the `package` keyword, and optionally declare a background color for your package (Using a html color code or name). When you declare classes, they are automatically put in the last used package, and you can close the package definition using the `end package` keyword. You can also use brackets `{ }`.

Note that package definitions can be nested.

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

package net.sourceforge.plantuml #Snow
  Object <|-- Demo1
  Demo1 *- Demo2
end package

@enduml
```



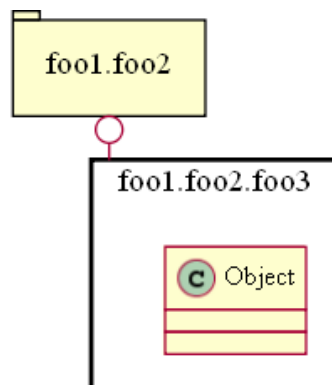
You can also define links between packages, like in the following example :

```
@startuml
package foo1.foo2
end package

package foo1.foo2.foo3 {
  class Object
}

foo1.foo2 +-- foo1.foo2.foo3

@enduml
```



3.12 Namespaces

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

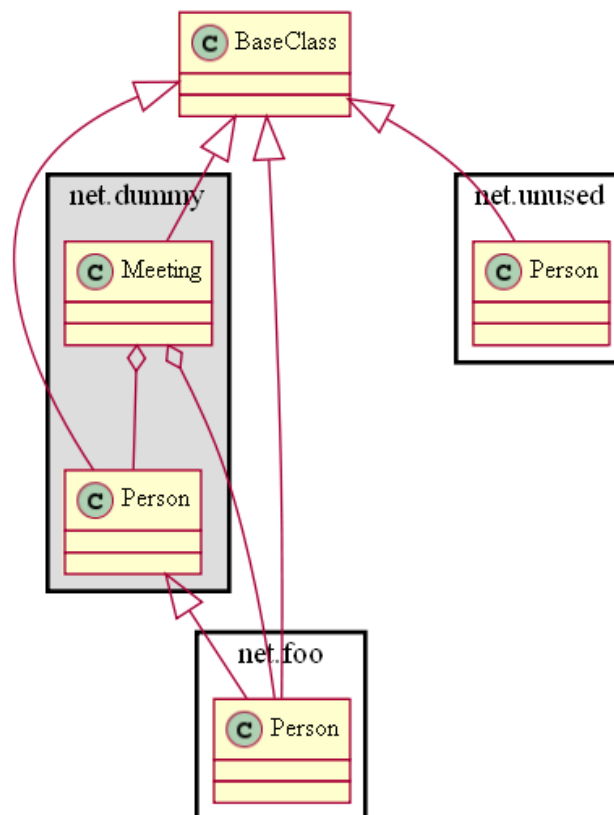
    .BaseClass <|-- Meeting
end namespace

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

    net.dummy.Meeting o-- Person
}

BaseClass <|-- net.unused.Person

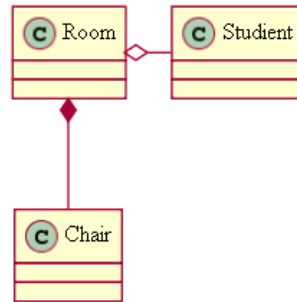
@enduml
```



3.13 Changing arrows direction

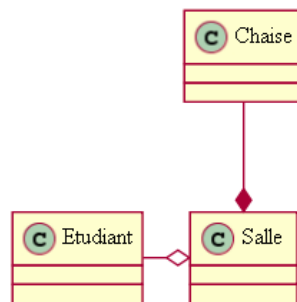
By default, links between classes have two dashes -- and are vertically oriented. It is possible to use horizontal link by putting a single dash (or dot) like this :

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



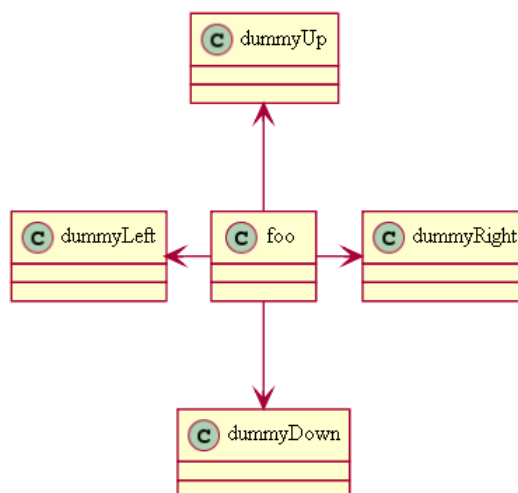
Vous pouvez aussi changer le sens en renversant le lien :

```
@startuml
Etudiant -o Salle
Chaise --* Salle
@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-`)

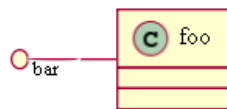
Please note that you should not abuse this fonctionnality : *GraphViz* gives usually good results without tweaking.

3.14 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
```

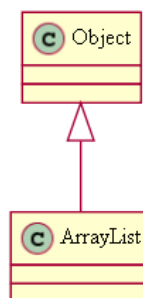


3.15 Title the diagram

The `title` keywords is used to put a title. You can use `title` and `end title` keywords for a longer title, as in sequence diagrams.

```
@startuml
title Simple <b>example</b>\nof title
Object <|-- ArrayList
@enduml
```

Simple example of title

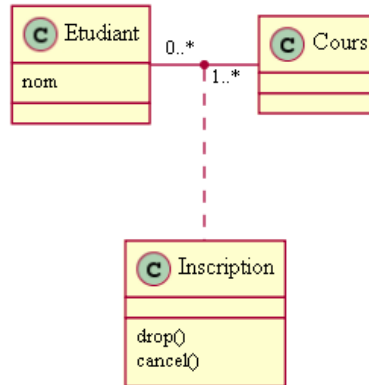


3.16 Association classes

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

```
@startuml
Etudiant : nom
Etudiant "0..*" -- "1..*" Cours
(Etudiant , Cours) .. Inscription

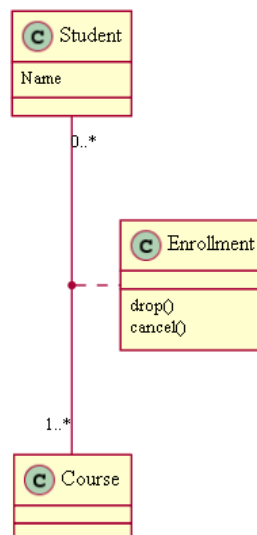
Inscription : drop()
Inscription : cancel()
@enduml
```



You can define it in another direction :

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

Enrollment : drop()
Enrollment : cancel()
@enduml
```



3.17 Skinparam

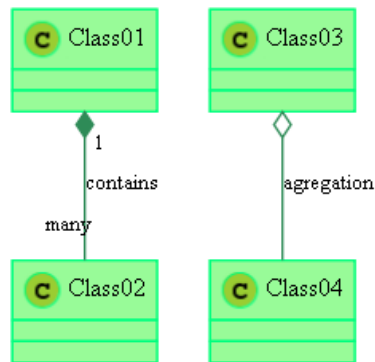
You can use the `skinparam` command to change colors and fonts for the drawing. You can use this command :

- 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 classBackgroundColor PaleGreen
skinparam classArrowColor SeaGreen
skinparam classBorderColor SpringGreen
skinparam stereotypeCBackgroundColor YellowGreen

Class01 "1" *-- "many" Class02 : contains
Class03 o-- Class04 : agregation

@enduml
```



3.18 Skinned Stereotypes

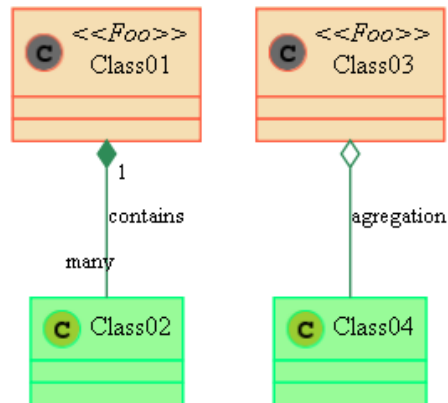
You can define specific color and fonts for stereotyped classes.

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

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

Class03 <<Foo>> o-- Class04 : agregation

@enduml
```



3.19 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,
- *vpages* is a number that indicated the number of vertical pages.

```
@startuml
' Split into 4 pages
page 2x2

class BaseClass

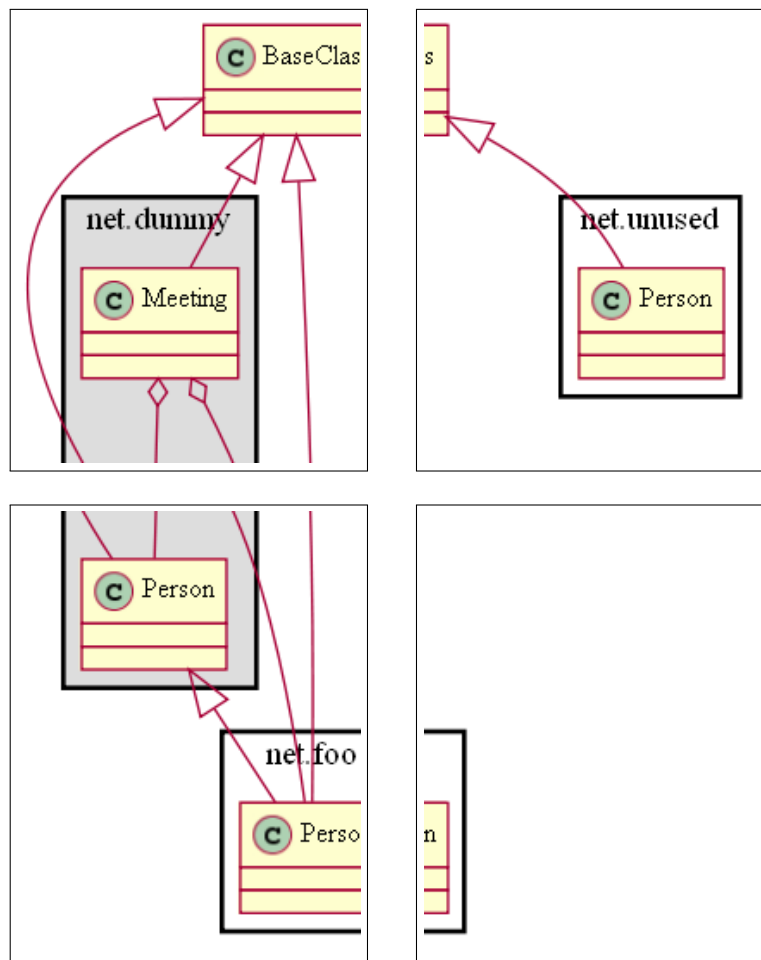
namespace net.dummy #DDDDD
.BaseClass <|-- Person
.Meeting o-- Person

.BaseClass <|-- Meeting
end namespace

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

.net.dummy.Meeting o-- Person
}

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



4 Diagrammes d'activité

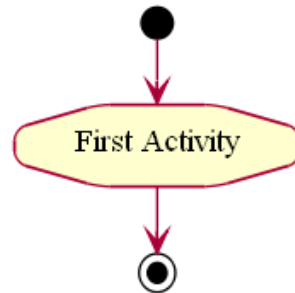
4.1 Exemple de base

You can use (*) for the starting point and ending point of the activity diagram.

Use --> for arrows.

Exemple :

```
@startuml
(*) --> "First Activity"
"First Activity" --> (*)
@enduml
```

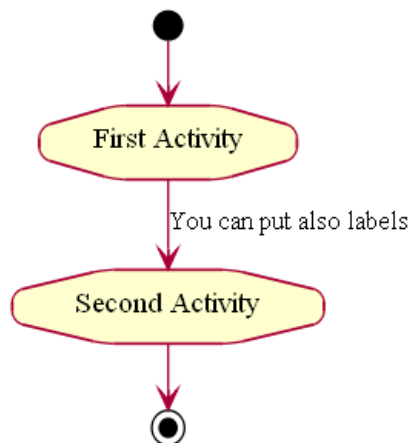


4.2 Label on arrows

By default, an arrow starts at the last used activity.

You can put a label on a arrow using brackets [and] just after the arrow definition.

```
@startuml
(*) --> "First Activity"
-->[You can put also labels] "Second Activity"
--> (*)
@enduml
```

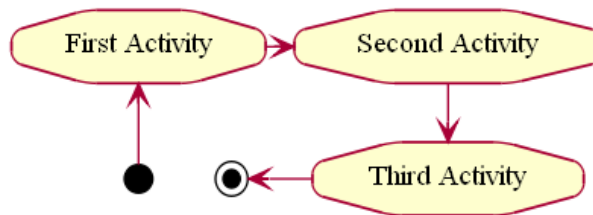


4.3 Changing arrow direction

You can use `->` for horizontal arrows. It is possible to force arrow's direction using the following syntax :

- `-down->` (default arrow)
- `-right->` or `->`
- `-left->`
- `-up->`

```
@startuml
(*) -up-> "First Activity"
-right-> "Second Activity"
--> "Third Activity"
-left-> (*)
@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-`).

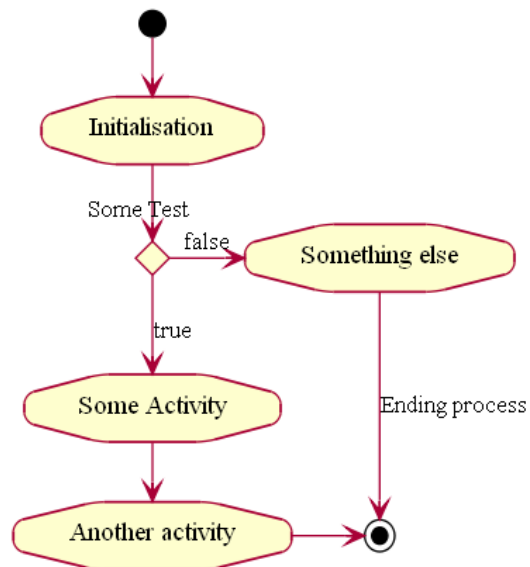
Please note that you should not abuse this fonctionnality : *GraphViz* gives usually good results without tweaking.

4.4 Branches

You can use if/then/else keywords to define branches.

```
@startuml
(*) --> "Initialisation"

if "Some Test" then
-->[true] "Some Activity"
--> "Another activity"
-right-> (*)
else
->[false] "Something else"
-->[Ending process] (*)
endif
end
@enduml
```



4.5 Encore des branches

Par défaut, une branche commence à la dernière activité définie, but it is possible to override this and to define a link with the if keywords.

It is also possible to nest branches.

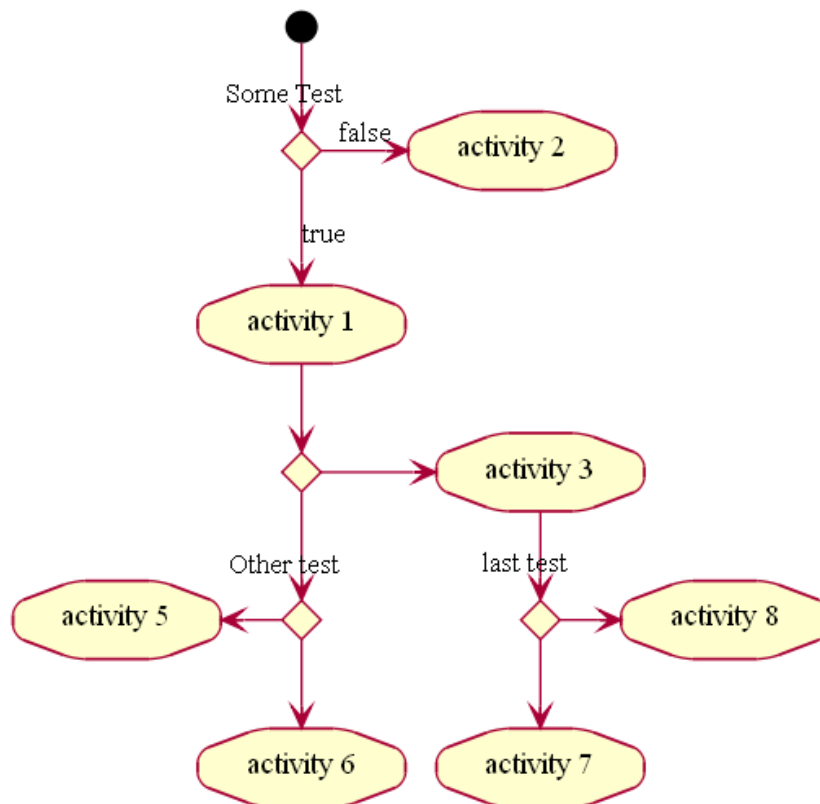
```
@startuml
(*) --> if "Some Test" then
  -->[true] "activity 1"

  if "" then
    -> "activity 3" as a3
  else
    if "Other test" then
      -left-> "activity 5"
    else
      --> "activity 6"
    endif
  endif
endif

else
  ->[false] "activity 2"

endif

a3 --> if "last test" then
  --> "activity 7"
else
  -> "activity 8"
endif
@enduml
```



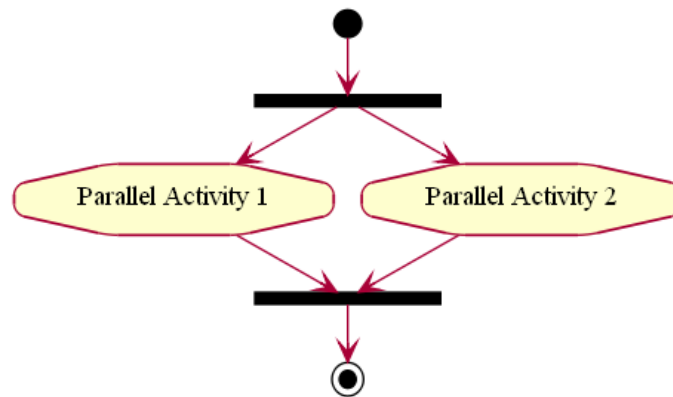
4.6 Synchronisation

Vous pouvez utiliser la syntaxe "=== code ===" pour afficher des barres de synchronisation.

```
@startuml
(*) --> ===B1===
--> "Parallel Activity 1"
--> ===B2===

===B1=== --> "Parallel Activity 2"
--> ===B2===

--> (*)
@enduml
```



4.7 Long activity description

When you declare activities, you can span on several lines the description text. You can also add `\n` in the description. It is also possible to use few html tags like :

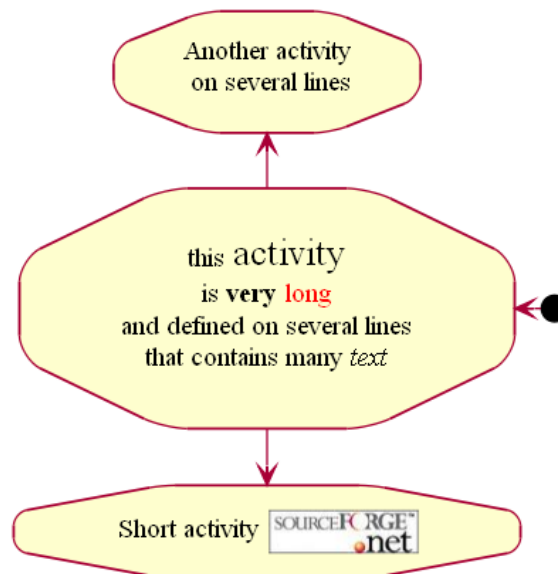
- ``
- `<i>`
- `` or `<size:nn>` to change font size
- `` or ``
- `<color:#AAAAAA>` or `<color:colorName>`
- `<img:file.png>` to include an image

You can also give a short code to the activity with the `as` keyword. This code can be used latter in the diagram description.

```
@startuml
(*) -left-> "this <size:20>activity</size>
    is <b>very</b> <color:red>long</color>
    and defined on several lines
    that contains many <i>text</i>" as A1

-up-> "Another activity\n on several lines"

A1 --> "Short activity <img:img/sourceforge.jpg>"
@enduml
```



4.8 Notes

Vous pouvez rajouter des notes sur une activités en utilisant les commandes :

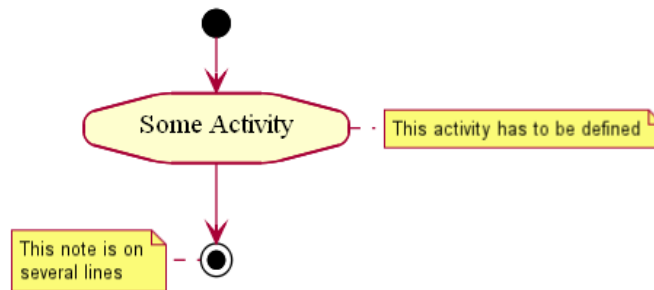
- `note left`,
- `note right`,
- `note top`,
- `note bottom`,

just after the description of the activity you want to note. If you want to put a note on the starting point, define the note at the very beginning of the diagram description.

You can also have a note on several lines, using the `end note` keywords.

```
@startuml
```

```
(*) --> "Some Activity"  
note right: This activity has to be defined  
"Some Activity" --> (*)  
note left  
  This note is on  
  several lines  
end note  
@enduml
```



4.9 Partition

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

When you declare activities, they are automatically put in the last used partition.

You can close the partition definition using the `end partition` keyword.

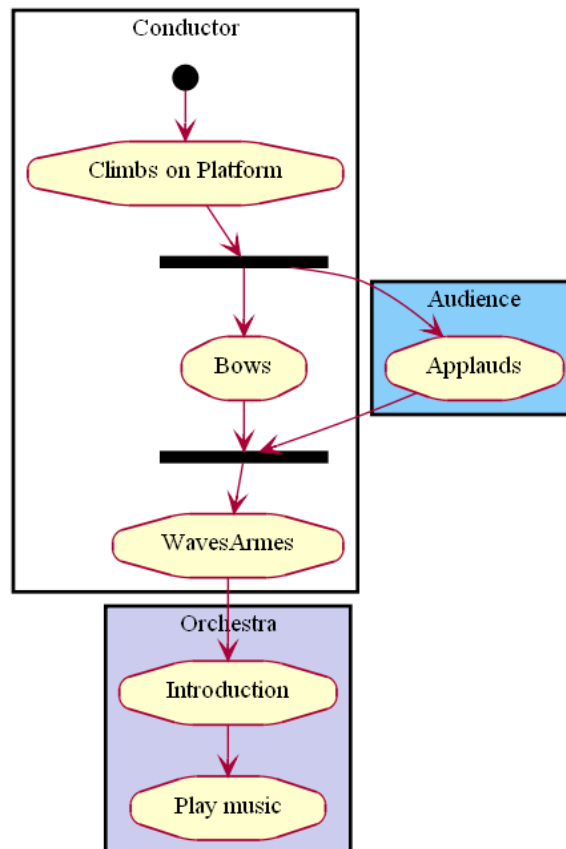
```
@startuml
partition Conductor
(*) --> "Climbs on Platform"
--> === S1 ===
--> Bows
end partition

partition Audience #LightSkyBlue
=== S1 === --> Applauds

partition Conductor
Bows --> === S2 ===
--> WavesArmes
Applauds --> === S2 ===
end partition

partition Orchestra #CCCCEE
WavesArmes --> Introduction
--> "Play music"
end partition

@enduml
```

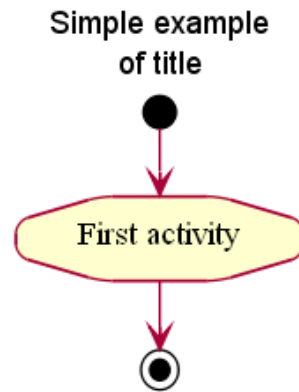


4.10 Title the diagram

The `title` keywords is used to put a title. You can use `title` and `end title` keywords for a longer title, as in sequence diagrams.

```
@startuml
title Simple example\nof title

(*) --> "First activity"
--> (*)
@enduml
```

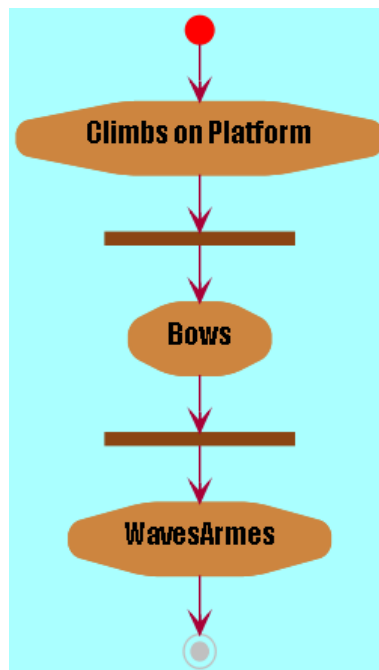


4.11 Skinparam

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 backgroundColor #AFFFFF
skinparam activity {
    StartColor red
    BarColor SaddleBrown
    EndColor Silver
    BackgroundColor Peru
    BorderColor Peru
    FontName Impact
}
(*) --> "Climbs on Platform"
--> === S1 ===
--> Bows
--> === S2 ===
--> WavesArmes
--> (*)
@enduml
```



4.12 Exemple complet

```

@startuml
'http://click.sourceforge.net/images/activity-diagram-small.png
title Moteur de Servlet

(*) --> "ClickServlet.handleRequest()"
--> "new Page"

if "Page.onSecurityCheck" then
->[true] "Page.onInit()"

    if "isForward?" then
->[no] "Process controls"

        if "continue processing?" then
-->[yes] ===RENDERING===
        else
-->[no] ===REDIRECT_CHECK===
        endif

    else
-->[yes] ===RENDERING===
    endif

    if "is Post?" then
-->[yes] "Page.onPost()"
--> "Page.onRender()" as render
--> ===REDIRECT_CHECK===
    else
-->[no] "Page.onGet()"
--> render
    endif

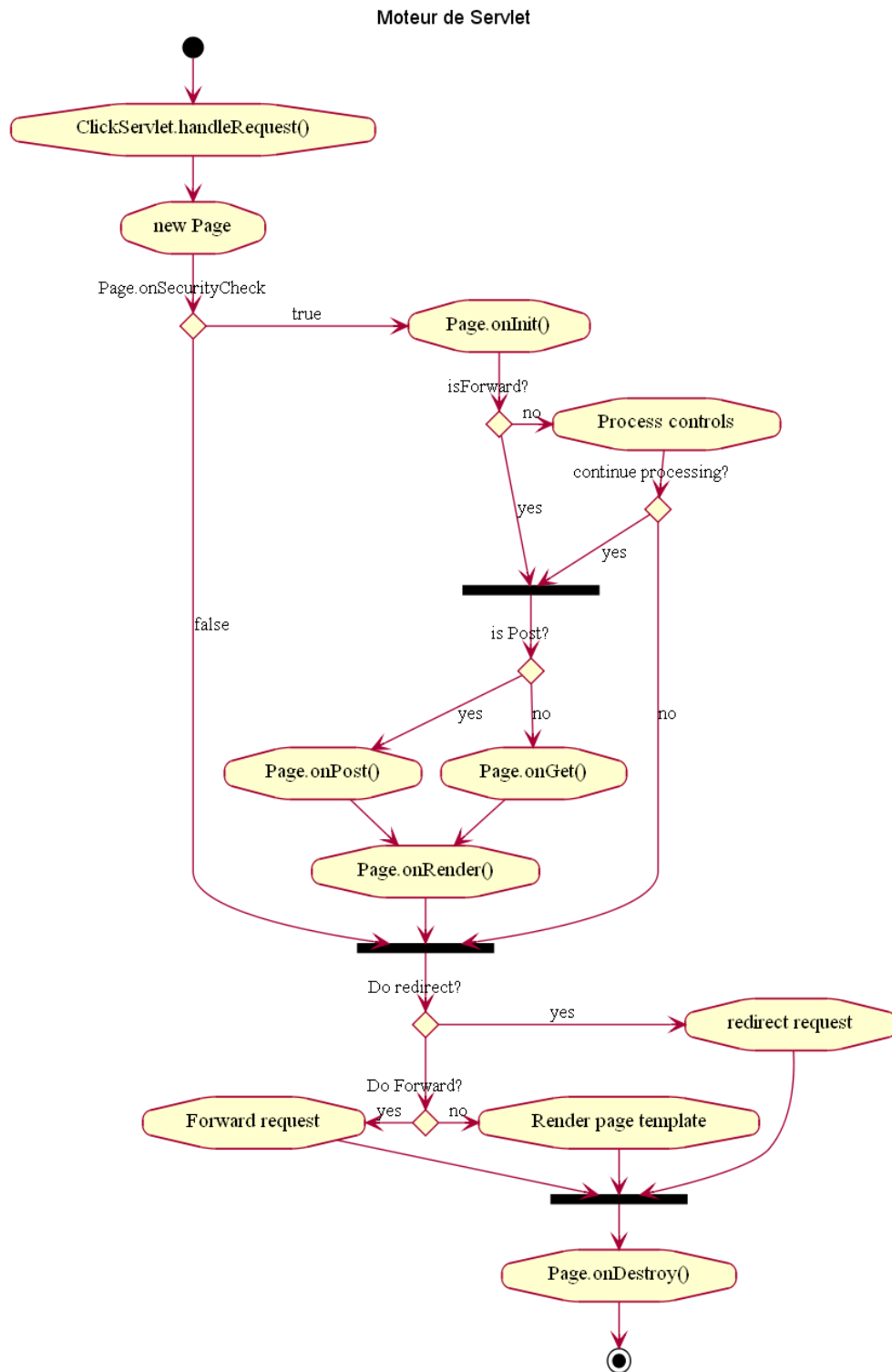
else
-->[false] ===REDIRECT_CHECK===
endif

if "Do redirect?" then
->[yes] "redirect request"
--> ==BEFORE_DESTROY===
else
    if "Do Forward?" then
-left->[yes] "Forward request"
--> ==BEFORE_DESTROY===
    else
-right->[no] "Render page template"
--> ==BEFORE_DESTROY===
    endif
endif

--> "Page.onDestroy()"
-->(*)

@enduml

```



5 Diagrammes de composants

5.1 Composants

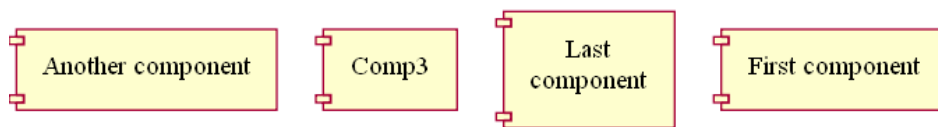
Les composants doivent être mis entre crochets.

Il est aussi possible d'utiliser le mot-clé `component` pour définir un composant.

Et vous pouvez définir un alias, grâce au mot-clé `as`.

Cet alias sera utile plus tard, pour définir des relations entre composants.

```
@startuml
[First component]
[Another component] as Comp2
component Comp3
component [Last\ncomponent] as Comp4
@enduml
```



5.2 Interfaces

Les interfaces sont définies à l'aide du symbole "()" (parce que cela ressemble à un cercle).

Vous pouvez aussi utiliser le mot-clé `interface` pour définir une interface.

And you can define an alias, using the `as` keyword.

This alias will be used latter, when defining relations.

We will see latter that interface declaration is optional.

```
@startuml
() "First Interface"
() "Another interface" as Interf2
interface Interf3
interface "Last\ninterface" as Interf4
@enduml
```



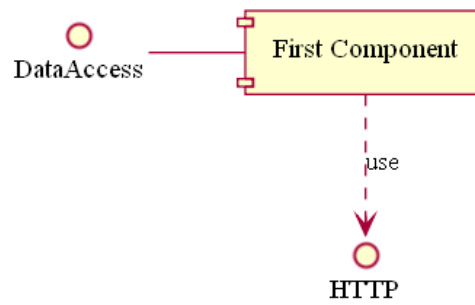
5.3 Basic example

Links between elements are made using combinations of dotted line “..”, straight line “--”, and arrows “-->” symbols.

```
@startuml
```

```
.DataAccess - [First Component]  
[First Component] ..> HTTP : use
```

```
@enduml
```



5.4 Using notes

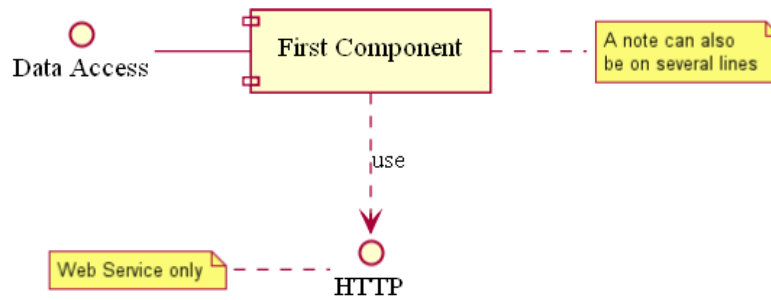
You can use the

- note left of,
- note right of,
- note top of,
- note bottom of,

keywords to define notes related to a single object.

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

```
@startuml
interface "Data Access" as DA
DA - [First Component]
[First Component] ..> HTTP : use
note left of HTTP : Web Service only
note right of [First Component]
  A note can also
  be on several lines
end note
@enduml
```

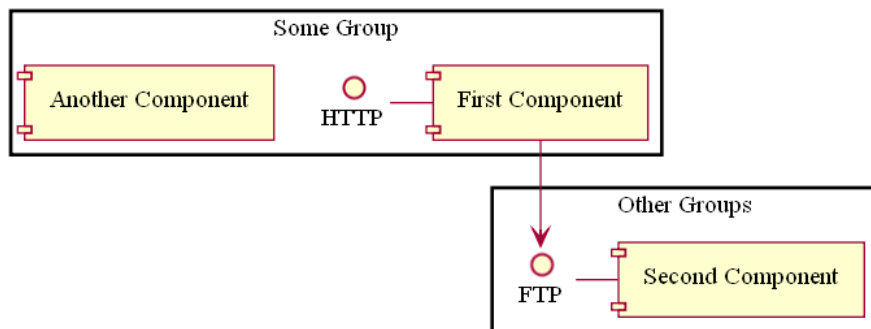


5.5 Regrouper des composants

Vous pouvez utiliser le mot-clé `package` pour regrouper des composants et des interfaces ensembles.

```
@startuml
package "Some Group" {
  HTTP - [First Component]
  [Another Component]
}

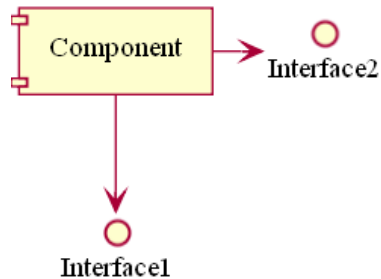
package "Other Groups" {
  FTP - [Second Component]
  [First Component] --> FTP
}
@enduml
```



5.6 Changing arrows direction

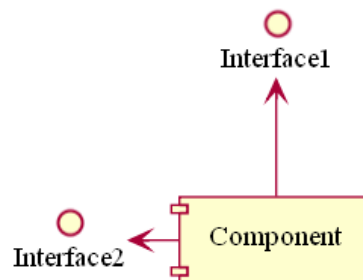
By default, links between classes have two dashes -- and are vertically oriented. It is possible to use horizontal link by putting a single dash (or dot) like this :

```
@startuml
[Component] --> Interface1
[Component] -> Interface2
@enduml
```



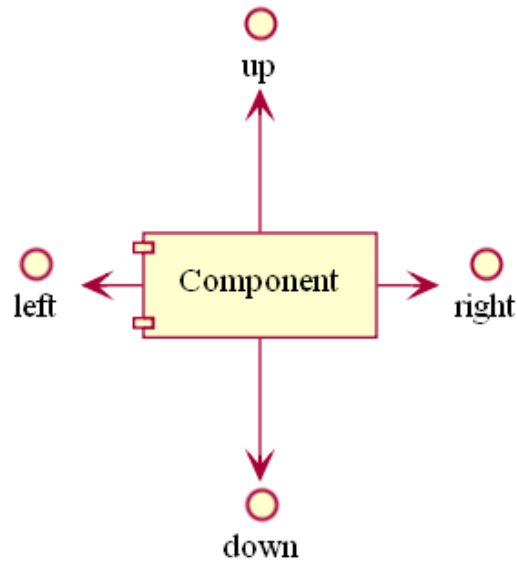
Vous pouvez aussi changer le sens en renversant le lien

```
@startuml
Interface1 <-- [Component]
Interface2 <- [Component]
@enduml
```



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

```
@startuml
[Component] -left-> left
[Component] -right-> right
[Component] -up-> up
[Component] -down-> down
@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 : *Graph Viz* gives usually good results without tweaking.

5.7 Ajouter un titre

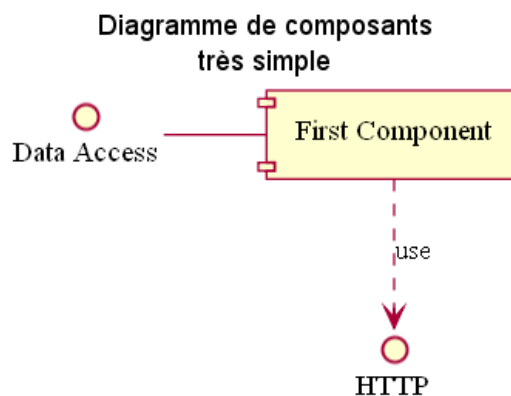
Le mot-clé `title` est utilisé pour rajouter un titre. Vous pouvez aussi utiliser `title` et `end title` pour avoir un titre plus long, comme dans les diagrammes de séquence.

```
@startuml
title Diagramme de composants\ntrès simple

interface "Data Access" as DA

DA - [First Component]
[First Component] ..> HTTP : use

@enduml
```



5.8 Skinparam

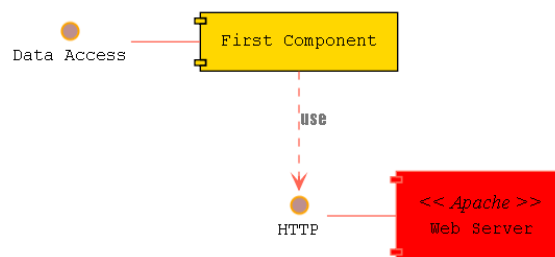
You can use the `skinparam` command to change colors and fonts for the drawing. You can use this command :

- 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 component {
  FontSize 13
  InterfaceBackgroundColor RosyBrown
  InterfaceBorderColor orange
  BackgroundColor<< Apache >> Red
  BorderColor<< Apache >> #FF6655
  FontName Courier
  BorderColor black
  BackgroundColor gold
  ArrowFontName Impact
  ArrowColor #FF6655
  ArrowFontColor #777777
}
```

```
() "Data Access" as DA
```

```
DA - [First Component]
[First Component] ..> () HTTP : use
HTTP - [Web Server] << Apache >>
@enduml
```



6 Diagrammes d'état

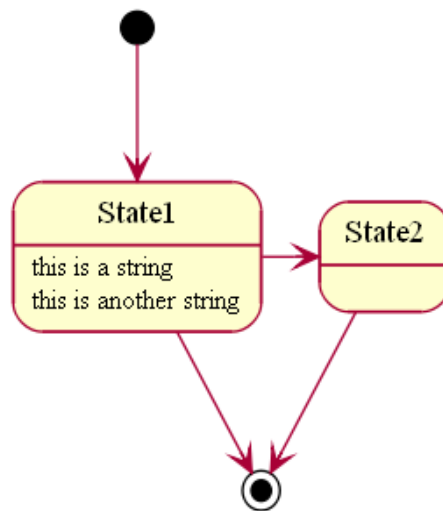
6.1 Exemple simple

Vous devez utiliser [*] pour le début et la fin du diagramme d'états.

Utilisez --> pour les flèches.

```
@startuml
[*] --> State1
State1 --> [*]
State1 : this is a string
State1 : this is another string

State1 -> State2
State2 --> [*]
@enduml
```



6.2 Etat composite

A state can also be composite. You have to define it using the **state** keywords and brackets.

```

@startuml
[*] --> NotShooting

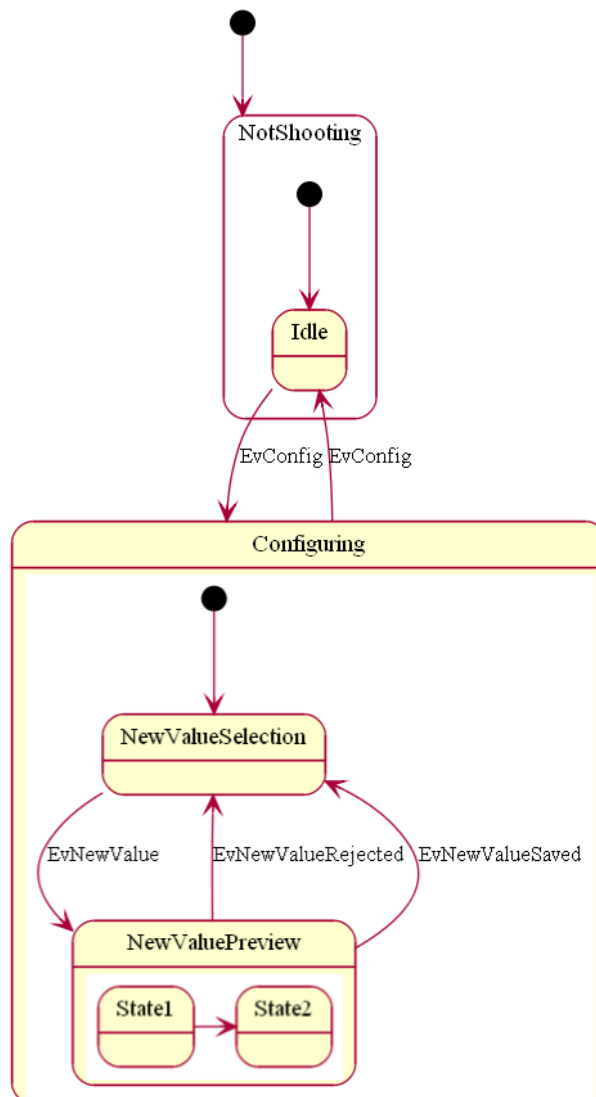
state NotShooting {
  [*] --> Idle
  Idle --> Configuring : EvConfig
  Configuring --> Idle : EvConfig
}

state Configuring {
  [*] --> NewValueSelection
  NewValueSelection --> NewValuePreview : EvNewValue
  NewValuePreview --> NewValueSelection : EvNewValueRejected
  NewValuePreview --> NewValueSelection : EvNewValueSaved

  state NewValuePreview {
    State1 -> State2
  }
}

}
@enduml

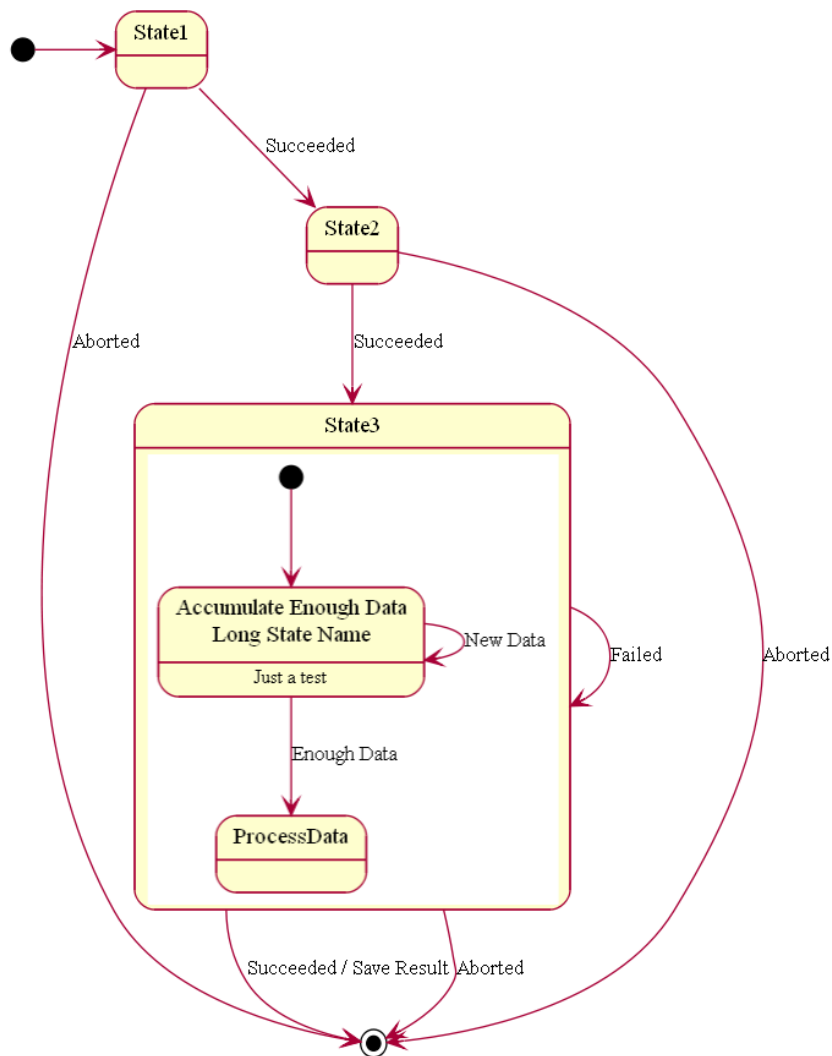
```



6.3 Long name

You can also use the `state` keyword to use long description for states.

```
@startuml
[*] -> State1
State1 --> State2 : Succeeded
State1 --> [*] : Aborted
State2 --> State3 : Succeeded
State2 --> [*] : Aborted
state State3 {
  state "Accumulate Enough Data\nLong State Name" as long1
  long1 : Just a test
  [*] --> long1
  long1 --> long1 : New Data
  long1 --> ProcessData : Enough Data
}
State3 --> State3 : Failed
State3 --> [*] : Succeeded / Save Result
State3 --> [*] : Aborted
@enduml
```



6.4 Concurrent state

You can define concurrent state into a composite state using the "--" symbol as separator.

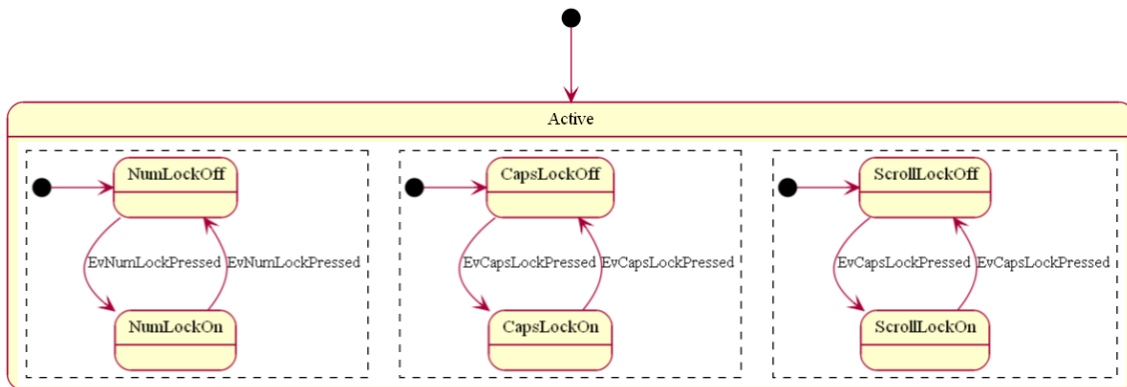
```

@startuml
[*] --> Active

state Active {
  [*] -> NumLockOff
  NumLockOff --> NumLockOn : EvNumLockPressed
  NumLockOn --> NumLockOff : EvNumLockPressed
  --
  [*] -> CapsLockOff
  CapsLockOff --> CapsLockOn : EvCapsLockPressed
  CapsLockOn --> CapsLockOff : EvCapsLockPressed
  --
  [*] -> ScrollLockOff
  ScrollLockOff --> ScrollLockOn : EvCapsLockPressed
  ScrollLockOn --> ScrollLockOff : EvCapsLockPressed
}

@enduml

```



6.5 Arrow direction

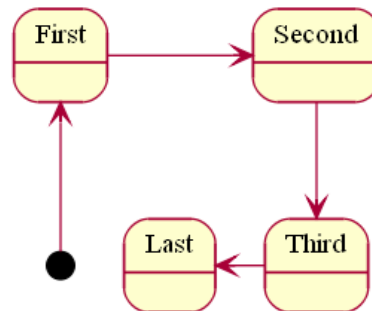
You can use `->` for horizontal arrows. It is possible to force arrow's direction using the following syntax :

- `-down->` (default arrow)
- `-right->` or `->`
- `-left->`
- `-up->`

```
@startuml
```

```
[*] -up-> First
First -right-> Second
Second --> Third
Third -left-> Last
```

```
@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 fonctionnality : *GraphViz* gives usually good results without tweaking.

6.6 Note

Vous pouvez définir des notes avec :

- note left of,
- note right of,
- note top of,
- note bottom of

Vous pouvez aussi définir des notes sur plusieurs lignes.

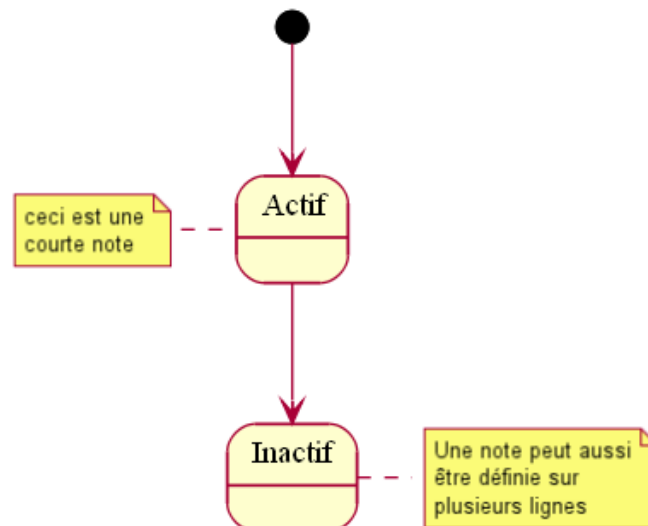
```
@startuml
```

```
[*] --> Actif
Actif --> Inactif
```

```
note left of Actif : ceci est une\ncourte note
```

```
note right of Inactif
  Une note peut aussi
  être définie sur
  plusieurs lignes
end note
```

```
@enduml
```



6.7 More in notes

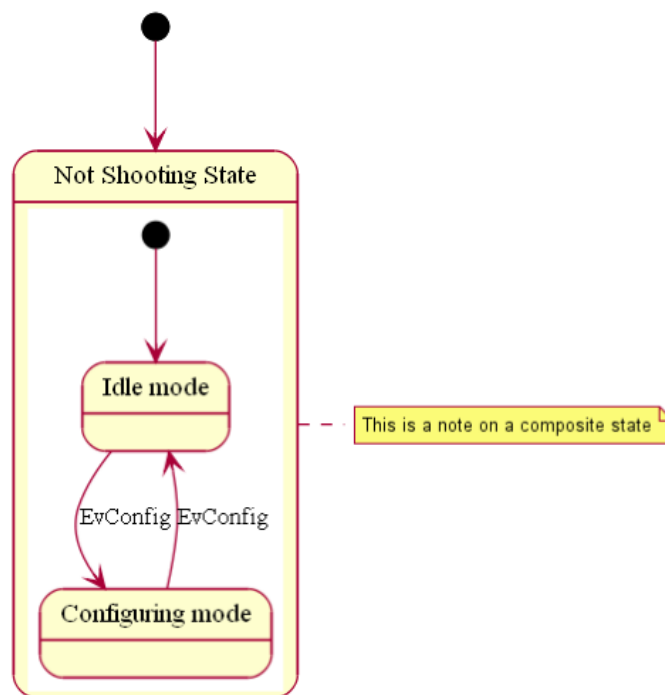
You can put notes on composite states.

```
@startuml
[*] --> NotShooting

state "Not Shooting State" as NotShooting {
  state "Idle mode" as Idle
  state "Configuring mode" as Configuring
  [*] --> Idle
  Idle --> Configuring : EvConfig
  Configuring --> Idle : EvConfig
}

note right of NotShooting : This is a note on a composite state

@enduml
```

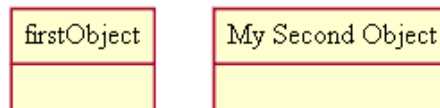


7 Diagrammes d'objets

7.1 Définition des objets

You define instance of objects using the object keywords.

```
@startuml
object firstObject
object "My Second Object" as o2
@enduml
```



7.2 Relations entre les objets

Relations between objects are defined using the following symbols :

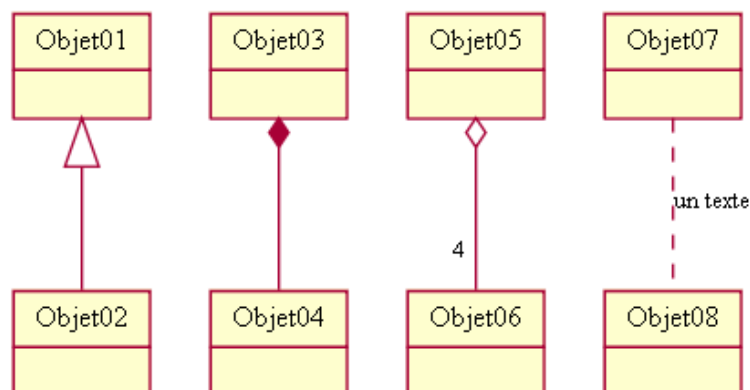
Extension	< --	
Composition	*--	
Agregation	o--	

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

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

```
@startuml
object Objet01
object Objet02
object Objet03
object Objet04
object Objet05
object Objet06
object Objet07
object Objet08

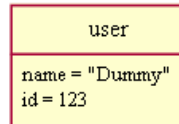
Objet01 <|-- Objet02
Objet03 *-- Objet04
Objet05 o-- "4" Objet06
Objet07 .. Objet08 : un texte
@enduml
```



7.3 Ajout de champs

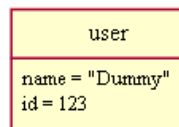
To declare fields, you can use the symbol ":" followed by the field's name.

```
@startuml
object user
user : name = "Dummy"
user : id = 123
@enduml
```



It is also possible to ground between brackets {} all fields.

```
@startuml
object user {
    name = "Dummy"
    id = 123
}
@enduml
```



8 Commandes communes

8.1 Footer and header

You can use the commands `header` or `footer` to add a footer or a header on any generated diagram.

You can optionally specify if you want a `center`, `left` or `right` footer/header, by adding a keyword.

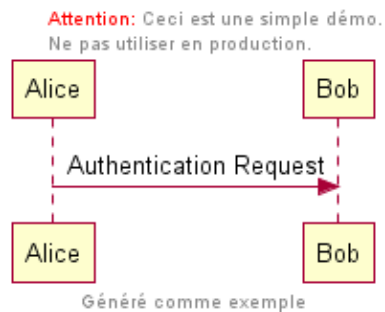
As for title, it is possible to define a header or a footer on several lines.

It is also possible to put some HTML into the header or footer

```
@startuml
Alice -> Bob: Authentication Request

header
<font color=red>Attention:</font> Ceci est une simple démo.
Ne pas utiliser en production.
endheader

center footer Généré comme exemple
@enduml
```

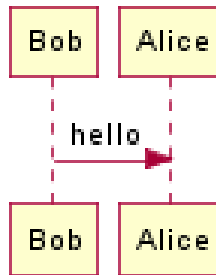


8.2 Zoom

You can use the scale command to zoom the generated image. You can use either a number or a fraction to define the scale factor. You can also specify either width or height (in pixel). And you can also give both width and height : the image is scaled to fit inside the specified dimension.

- scale 1.5,
- scale 2/3,
- scale 200 width,
- scale 200 height,
- scale 200*100

```
@startuml
scale 180*90
Bob->Alice : hello
@enduml
```



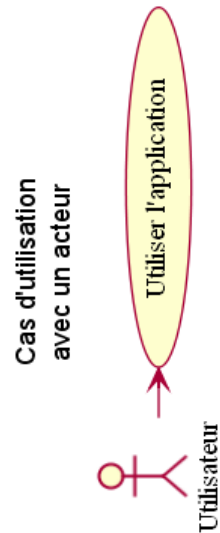
8.3 Rotation

Sometimes, and especially for printing, you may want to rotate the generated image, so that it fits better in the page. You can use the `rotate` command for this.

```
@startuml
rotate

title Cas d'utilisation\avec un acteur
"Utiliser l'application" as (Utiliser)
Utilisateur -> (Utiliser)

@enduml
```



9 Changing fonts and colors

9.1 Usage

You can change colors and font of the drawing using the `skinparam` command. Example :

```
skinparam backgroundColor yellow
```

Vous pouvez utiliser cette commande :

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

9.2 Imbrication

Pour éviter les répétitions, il est possible d'imbriquer les définitions. Ainsi, le texte suivant :

```
skinparam xxxxParam1 value1
skinparam xxxxParam2 value2
skinparam xxxxParam3 value3
skinparam xxxxParam4 value4
```

is strictly equivalent to :

```
skinparam xxxx {
  Param1 value1
  Param2 value2
  Param3 value3
  Param4 value4
}
```

9.3 Color

You can use either standard color name or RGB code.

Parameter name	Default Value	Color	Comment
backgroundColor	white		Background of the page
activityArrowColor	#A80036		Color of arrows in activity diagrams
activityBackgroundColor	#FEFECE		Background of activities
activityBorderColor	#A80036		Color of activity borders
activityStartColor	black		Starting circle in activity diagrams
activityEndColor	black		Ending circle in activity diagrams
activityBarColor	black		Synchronization bar in activity diagrams
usecaseArrowColor	#A80036		Color of arrows in usecase diagrams
usecaseActorBackgroundColor	#FEFECE		Head's color of actor in usecase diagrams
usecaseActorBorderColor	#A80036		Color of actor borders in usecase diagrams
usecaseBackgroundColor	#FEFECE		Background of usecases
usecaseBorderColor	#A80036		Color of usecase borders in usecase diagrams
classArrowColor	#A80036		Color of arrows in class diagrams
classBackgroundColor	#FEFECE		Background of classes/interface/enum in class diagrams
classBorderColor	#A80036		Borders of classes/interface/enum in class diagrams
packageBackgroundColor	#FEFECE		Couleur de fond des packages dans les diagrammes de classes
packageBorderColor	#A80036		Borders of packages in class diagrams
stereotypeCBackgroundColor	#ADD1B2		Background of class spots in class diagrams
stereotypeABackgroundColor	#A9DCDF		Background of abstract class spots in class diagrams
stereotypeIBackgroundColor	#B4A7E5		Background of interface spots in class diagrams
stereotypeEBackgroundColor	#EB937F		Background of enum spots in class diagrams
componentArrowColor	#A80036		Color of arrows in component diagrams
componentBackgroundColor	#FEFECE		Background of components
componentBorderColor	#A80036		Borders of components
componentInterfaceBackgroundColor	#FEFECE		Background of interface in component diagrams
componentInterfaceBorderColor	#A80036		Border of interface in component diagrams
noteBackgroundColor	#FBFB77		Background of notes
noteBorderColor	#A80036		Border of notes
stateBackgroundColor	#FEFECE		Background of states in state diagrams
stateBorderColor	#A80036		Border of states in state diagrams
stateArrowColor	#A80036		Colors of arrows in state diagrams
stateStartColor	black		Starting circle in state diagrams
stateEndColor	black		Ending circle in state diagrams
sequenceArrowColor	#A80036		Couleurs des flèches dans les diagramme de séquence
sequenceActorBackgroundColor	#FEFECE		Head's color of actor in sequence diagrams
sequenceActorBorderColor	#A80036		Border of actor in sequence diagrams
sequenceGroupBackgroundColor	#EEEEEE		Header color of alt/opt/loop in sequence diagrams
sequenceLifeLineBackgroundColor	white		Background of life line in sequence diagrams
sequenceLifeLineBorderColor	#A80036		Border of life line in sequence diagrams
sequenceParticipantBackgroundColor	#FEFECE		Couleur de fond des participants dans les diagrammes de séquence
sequenceParticipantBorderColor	#A80036		Border of participant in sequence diagrams

9.4 Font color, name and size

You can change the font for the drawing using `xxxFontColor`, `xxxFontSize` and `xxxFontName` parameters.

Example :

```
skinparam classFontColor red
skinparam classFontSize 10
skinparam classFontName Aapex
```

You can also change the default font for all fonts using `skinparam defaultFontName`.

Example :

```
skinparam defaultFontName Aapex
```

Please note the fontname is highly system dependant, so do not over use it, if you look for portability.

Parameter Name	Default Value	Comment
activityFontColor activityFontSize activityFontStyle activityFontName	black 14 plain	Used for activity box
activityArrowFontColor activityArrowFontSize activityArrowFontStyle activityArrowFontName	black 13 plain	Used for text on arrows in activity diagrams
circledCharacterFontColor circledCharacterFontSize circledCharacterFontStyle circledCharacterFontName circledCharacterRadius	black 17 bold Courier 11	Used for text in circle for class, enum and others
classArrowFontColor classArrowFontSize classArrowFontStyle classArrowFontName	black 10 plain	Used for text on arrows in class diagrams
classAttributeFontColor classAttributeFontSize classAttributeIconSize classAttributeFontStyle classAttributeFontName	black 10 10 plain	Class attributes and methods
classFontColor classFontSize classFontStyle classFontName	black 12 plain	Used for classes name
classStereotypeFontColor classStereotypeFontSize classStereotypeFontStyle classStereotypeFontName	black 12 italic	Used for stereotype in classes
componentFontColor componentFontSize componentFontStyle componentFontName	black 14 plain	Used for components name
componentStereotypeFontColor componentStereotypeFontSize componentStereotypeFontStyle componentStereotypeFontName	black 14 italic	Used for stereotype in components
componentArrowFontColor componentArrowFontSize componentArrowFontStyle componentArrowFontName	black 13 plain	Used for text on arrows in component diagrams

noteFontColor noteFontSize noteFontStyle noteFontName	black 13 plain	Used for notes in all diagrams but sequence diagrams
packageFontColor packageFontSize packageFontStyle packageFontName	black 14 plain	Used for package and partition names
sequenceActorFontColor sequenceActorFontSize sequenceActorFontStyle sequenceActorFontName	black 13 plain	Used for actor in sequence diagrams
sequenceDividerFontColor sequenceDividerFontSize sequenceDividerFontStyle sequenceDividerFontName	black 13 bold	Used for text on dividers in sequence diagrams
sequenceArrowFontColor sequenceArrowFontSize sequenceArrowFontStyle sequenceArrowFontName	black 13 plain	Used for text on arrows in sequence diagrams
sequenceGroupingFontColor sequenceGroupingFontSize sequenceGroupingFontStyle sequenceGroupingFontName	black 11 plain	Used for text for "else" in sequence diagrams
sequenceGroupingHeaderFontColor sequenceGroupingHeaderFontSize sequenceGroupingHeaderFontStyle sequenceGroupingHeaderFontName	black 13 plain	Used for text for "alt/opt/loop" headers in sequence diagrams
sequenceParticipantFontColor sequenceParticipantFontSize sequenceParticipantFontStyle sequenceParticipantFontName	black 13 plain	Used for text on participant in sequence diagrams
sequenceTitleFontColor sequenceTitleFontSize sequenceTitleFontStyle sequenceTitleFontName	black 13 plain	Used for titles in sequence diagrams
titleFontColor titleFontSize titleFontStyle titleFontName	black 18 plain	Used for titles in all diagrams but sequence diagrams
stateFontColor stateFontSize stateFontStyle stateFontName	black 14 plain	Used for states in state diagrams
stateArrowFontColor stateArrowFontSize stateArrowFontStyle stateArrowFontName	black 13 plain	Used for text on arrows in state diagrams
stateAttributeFontColor stateAttributeFontSize stateAttributeFontStyle stateAttributeFontName	black 12 plain	Used for states description in state diagrams
usecaseFontColor usecaseFontSize usecaseFontStyle usecaseFontName	black 14 plain	Used for usecase labels in usecase diagrams

usecaseStereotypeFontColor usecaseStereotypeFontSize usecaseStereotypeFontStyle usecaseStereotypeFontName	black 14 italic	Used for stereotype in usecase
usecaseActorFontColor usecaseActorFontSize usecaseActorFontStyle usecaseActorFontName	black 14 plain	Used for actor labels in usecase diagrams
usecaseActorStereotypeFontColor usecaseActorStereotypeFontSize usecaseActorStereotypeFontStyle usecaseActorStereotypeFontName	black 14 italic	Used for stereotype for actor
usecaseArrowFontColor usecaseArrowFontSize usecaseArrowFontStyle usecaseArrowFontName	black 13 plain	Used for text on arrows in usecase diagrams
footerFontColor footerFontSize footerFontStyle footerFontName	black 10 plain	Used for footer
headerFontColor headerFontSize headerFontStyle headerFontName	black 10 plain	Used for header

9.5 Black and White

You can force the use of a blackwhite output using the `skinparam monochrome true` command.

```
@startuml
skinparam monochrome true

actor User
participant "First Class" as A
participant "Second Class" as B
participant "Last Class" as C

User -> A: DoWork
activate A

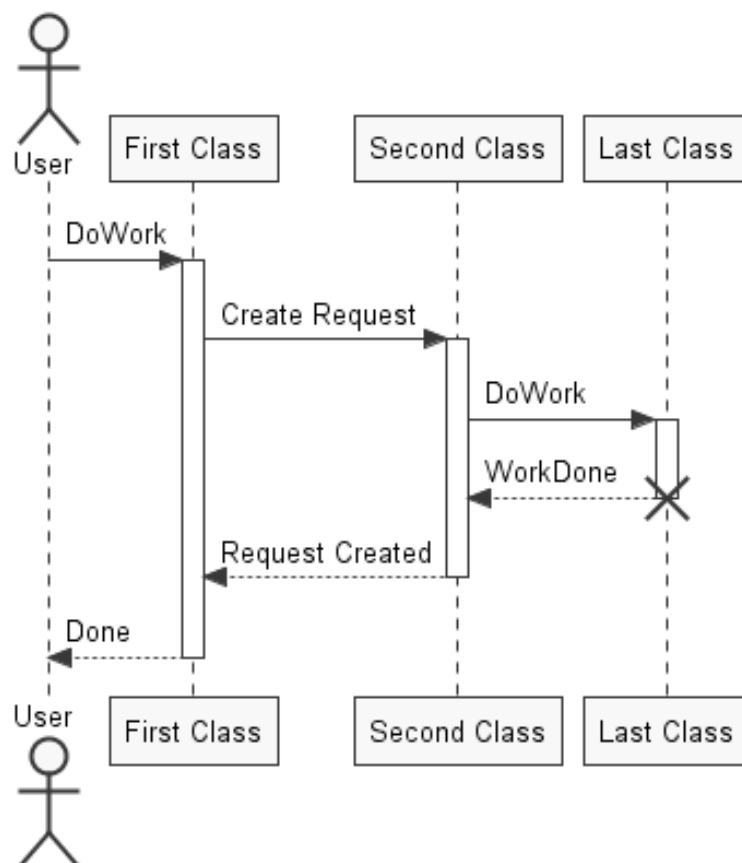
A -> B: Create Request
activate B

B -> C: DoWork
activate C
C --> B: WorkDone
destroy C

B --> A: Request Created
deactivate B

A --> User: Done
deactivate A

@enduml
```



10 Préprocesseur

Quelques fonctionnalités de préprocesseur sont présentes dans PlantUML, et disponibles pour tous les diagrammes.

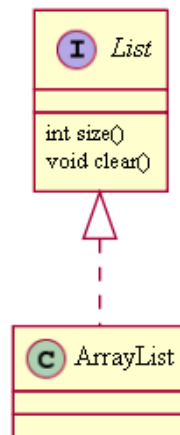
Ces fonctionnalités sont très proches du langage C, sauf que le point d'exclamation "!" a été utilisé à la place du caractère "#" pour les directives.

10.1 Inclusion de fichier

Il faut utiliser la directive `!include` pour inclure des fichiers dans les diagrammes.

Supposons que vous avez la même classe qui apparaît dans de nombreux diagrammes. Au lieu de dupliquer la description de la classe, vous pouvez créer un fichier unique qui contient cette description.

```
@startuml
!include List.iuml
List <|.. ArrayList
@enduml
```



File List.iuml :

```
interface List
List : int size()
List : void clear()
```

Le fichier `List.iuml` peut ainsi être inclus dans plusieurs fichiers, et tout changement dans ce fichier modifiera l'ensemble des diagrammes qui l'utilise.

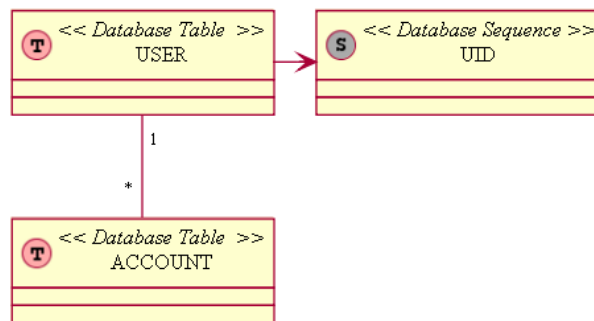
10.2 Définition de constantes

You can define constant using the `!define` directive. As in **C** language, a constant name can only use alphanumeric and underscore characters, and cannot start with a digit.

```
@startuml
!define SEQUENCE (S,#AAAAAA) Database Sequence
!define TABLE (T,#FFAAAA) Database Table

class USER << TABLE >>
class ACCOUNT << TABLE >>
class UID << SEQUENCE >>
USER "1" -- "*" ACCOUNT
USER -> UID

@enduml
```



Of course, you can use the `!include` directive to define all your constants in a single file that you include in your diagram.

Constant can be undefined with the `!undef XXX` directive.

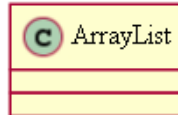
10.3 Conditions

Vous pouvez utiliser les directives `!ifdef XXX` et `!endif` pour avoir des parties optionnelles.

Les lignes entre les deux directives ne seront mis que si la constante après la directive `!ifdef` a été définie auparavant.

Il est aussi possible d'utiliser un `!else` qui ne sera mis que si la constante n'a pas été définie.

```
@startuml
!include ArrayList.iuml
@enduml
```

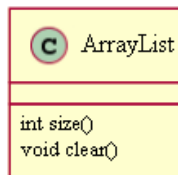


File `ArrayList.iuml` :

```
class ArrayList
!ifdef SHOW_METHODS
ArrayList : int size()
ArrayList : void clear()
!endif
```

You can then use the `!define` directive to activate the conditionnal part of the diagram.

```
@startuml
!define SHOW_METHODS
!include ArrayList.iuml
@enduml
```

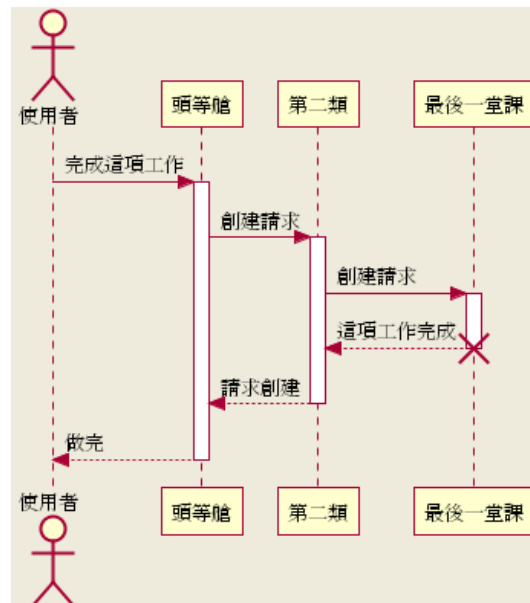


You can also use the `!ifndef` directive that includes lines if the provided constant has *NOT* been defined.

11 Internationalisation

Le langage PlantUML utilise des *lettres* pour définir des acteurs, des cas d'utilisation et d'autres entités. Mais les *lettres* ne sont pas simplement les caractères A à Z de l'alphabet latin, cela peut être *n'importe quelle lettre dans n'importe quelle langue*.

```
@startuml
skinparam backgroundColor #EEEEBD
actor 使用者
participant 頭等艙" as A
participant 第二類" as B
participant 最後一堂課" as 別的東西使用者
-> A: 完成這項工作
activate A
A -> B: 創建請求
activate B
B -> 別的東西: 創建請求
activate 別的東西別的東西
-> B: 這項工作完成
destroy 別的東西
B -> A: 請求創建
deactivate B
A -> 使用者: 做完
deactivate A
@enduml
```



11.1 Jeux de caractères

Le jeu de caractères utilisé par défaut pour la lecture des fichiers texte contenant la description UML dépend du système. Normalement, cela devrait convenir, mais dans certains cas, vous voudrez utiliser un autre jeu de caractères. Par exemple, en ligne de commande :

```
java -jar plantuml.jar -charset UTF-8 files.txt
```

Ou, avec la tâche ant :

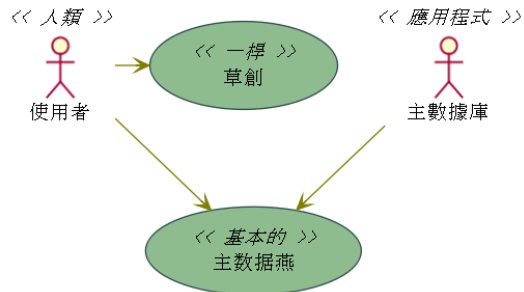
```
<!-- Put images in c:/images directory -->
<target name="main">
<plantuml dir="./src" charset="UTF-8" />
</target>
```

En fonction de l'installation de Java, les encodages suivant devraient être présents sur votre système : ISO-8859-1, UTF-8, UTF-16BE, UTF-16LE, UTF-16.

11.2 Problèmes de Police

Quand vous utilisez des polices de caractères asiatiques, vous pouvez rencontrer quelques problèmes, parce que les polices par défaut de *Graphviz* ne contiennent pas ces caractères. Ainsi, vous pouvez être obligé d'ajouter les lignes suivantes, afin de forcer l'utilisation de polices systèmes qui contiennent ces caractères :

```
skinparam defaultFontName MS Mincho
```



12 Nom des couleurs

Voici la liste des couleurs reconnues par PlantUML. Notez que les noms de couleur ne prennent pas en compte les majuscules/minuscules.

	AliceBlue		GhostWhite		NavajoWhite
	AntiqueWhite		GoldenRod		Navy
	Aquamarine		Gold		OldLace
	Aqua		Gray		OliveDrab
	Azure		GreenYellow		Olive
	Beige		Green		OrangeRed
	Bisque		HoneyDew		Orange
	Black		HotPink		Orchid
	BlanchedAlmond		IndianRed		PaleGoldenRod
	BlueViolet		Indigo		PaleGreen
	Blue		Ivory		PaleTurquoise
	Brown		Khaki		PaleVioletRed
	BurlyWood		LavenderBlush		PapayaWhip
	CadetBlue		Lavender		PeachPuff
	Chartreuse		LawnGreen		Peru
	Chocolate		LemonChiffon		Pink
	Coral		LightBlue		Plum
	CornflowerBlue		LightCoral		PowderBlue
	Cornsilk		LightCyan		Purple
	Crimson		LightGoldenRodYellow		Red
	Cyan		LightGreen		RosyBrown
	DarkBlue		LightGrey		RoyalBlue
	DarkCyan		LightPink		SaddleBrown
	DarkGoldenRod		LightSalmon		Salmon
	DarkGray		LightSeaGreen		SandyBrown
	DarkGreen		LightSkyBlue		SeaGreen
	DarkKhaki		LightSlateGray		SeaShell
	DarkMagenta		LightSteelBlue		Sienna
	DarkOliveGreen		LightYellow		Silver
	DarkOrchid		LimeGreen		SkyBlue
	DarkRed		Lime		SlateBlue
	DarkSalmon		Linen		SlateGray
	DarkSeaGreen		Magenta		Snow
	DarkSlateBlue		Maroon		SpringGreen
	DarkSlateGray		MediumAquaMarine		SteelBlue
	DarkTurquoise		MediumBlue		Tan
	DarkViolet		MediumOrchid		Teal
	Darkorange		MediumPurple		Thistle
	DeepPink		MediumSeaGreen		Tomato
	DeepSkyBlue		MediumSlateBlue		Turquoise
	DimGray		MediumSpringGreen		Violet
	DodgerBlue		MediumTurquoise		Wheat
	FireBrick		MediumVioletRed		WhiteSmoke
	FloralWhite		MidnightBlue		White
	ForestGreen		MintCream		YellowGreen
	Fuchsia		MistyRose		Yellow
	Gainsboro		Moccasin		

Table des matières

1	Diagramme de séquence	1
1.1	Exemples de base	1
1.2	Déclaration de participants	2
1.3	Caractères non alphanumérique dans les participants	3
1.4	Message à soi-même	3
1.5	Autre style de flèches	4
1.6	Numérotation automatique	5
1.7	Titre	7
1.8	Découper un diagramme	8
1.9	Regrouper les messages	9
1.10	Note sur les messages	10
1.11	Encore plus de notes	11
1.12	Formattage grâce au HTML	12
1.13	Séparation	13
1.14	Lignes de vie	14
1.15	Création de participants.	16
1.16	Messages entrant et sortant	17
1.17	Stéréotypes et décoration	18
1.18	Plus d'information sur les titres	19
1.19	Cadre pour les participants	21
1.20	Supprimer les en-pieds	22
1.21	Personnalisation	23
1.22	Thème	24
2	Diagramme de cas d'utilisation	25
2.1	Cas d'utilisation	25
2.2	Acteurs	26
2.3	Exemples très simples	27
2.4	Extension	28
2.5	Using notes	29
2.6	Stereotypes	30
2.7	Changing arrows direction	31
2.8	Title the diagram	32
2.9	Left to right direction	33
2.10	Skinparam	34
3	Diagramme de classes	35
3.1	Relations entre classes	35
3.2	Label on relations	36
3.3	Adding methods	37
3.4	Defining visibility	38
3.5	Notes and stereotypes	39
3.6	Encore des notes	40
3.7	Abstract class and interface	41
3.8	Using non-letters	42
3.9	Hide attributes, methods...	43
3.10	Specific Spot	44
3.11	Packages	45
3.12	Namespaces	46
3.13	Changing arrows direction	47
3.14	Lollipop interface	48
3.15	Title the diagram	48
3.16	Association classes	49
3.17	Skinparam	50
3.18	Skinned Stereotypes	51
3.19	Splitting large files	52

4 Diagrammes d'activité	53
4.1 Exemple de base	53
4.2 Label on arrows	53
4.3 Changing arrow direction	54
4.4 Branches	55
4.5 Encore des branches	56
4.6 Synchronisation	57
4.7 Long activity description	58
4.8 Notes	59
4.9 Partition	60
4.10 Title the diagram	61
4.11 Skinparam	62
4.12 Exemple complet	63
5 Diagrammes de composants	65
5.1 Composants	65
5.2 Interfaces	66
5.3 Basic example	67
5.4 Using notes	68
5.5 Regrouper des composants	69
5.6 Changing arrows direction	70
5.7 Ajouter un titre	71
5.8 Skinparam	72
6 Diagrammes d'état	73
6.1 Exemple simple	73
6.2 Etat composite	74
6.3 Long name	75
6.4 Concurrent state	76
6.5 Arrow direction	77
6.6 Note	78
6.7 More in notes	79
7 Diagrammes d'objets	80
7.1 Définition des objets	80
7.2 Relations entre les objets	80
7.3 Ajout de champs	81
8 Commandes communes	82
8.1 Footer and header	82
8.2 Zoom	83
8.3 Rotation	84
9 Changing fonts and colors	85
9.1 Usage	85
9.2 Imbrication	85
9.3 Color	86
9.4 Font color, name and size	87
9.5 Black and White	90
10 Préprocesseur	91
10.1 Inclusion de fichier	91
10.2 Définition de constantes	92
10.3 Conditions	93
11 Internationalisation	94
11.1 Jeux de caractères	94
11.2 Problèmes de Police	95
12 Nom des couleurs	96

