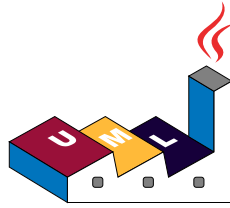


Dessiner de l'UML avec PlantUML



Guide de référence du langage (mardi 6 octobre 2015 18:46)

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

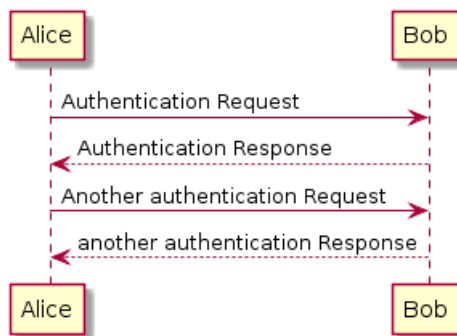
Le symbole ">" est utilisé pour dessiner un message entre deux participants. Les participants n'ont pas besoin d'être explicitement déclarés.

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.

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

Alice -> Bob: Another authentication Request
Alice <-- Bob: another authentication Response
@enduml
```



1.2 Commentaires

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

Vous pouvez aussi mettre des commentaires sur plusieurs lignes en utilisant '/' au début et '/' à la fin.

1.3 Déclaration de participants

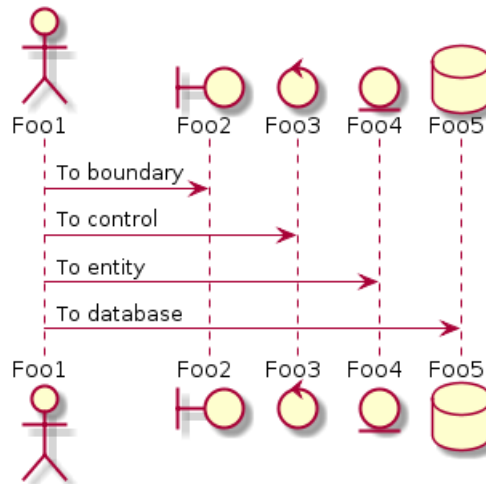
Il est possible de changer l'ordre des participants à l'aide du mot clé **participant**.

Il est aussi possible d'utiliser d'autres mot-clés pour déclarer un participant :

- actor
- boundary
- control
- entity
- database

```
@startuml
actor Foo1
boundary Foo2
control Foo3
entity Foo4
database Foo5
Foo1 -> Foo2 : To boundary
Foo1 -> Foo3 : To control
Foo1 -> Foo4 : To entity
Foo1 -> Foo5 : To database
@enduml
```



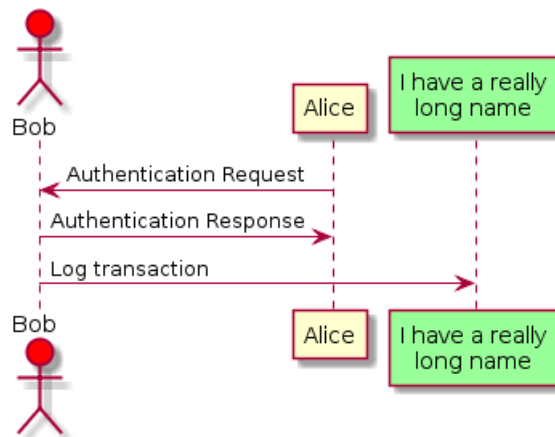


On peut aussi utiliser un nom court à l'aide grâce au mot-clé **as**.

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

```
@startuml
actor Bob #red
' The only difference between actor
' and participant is the drawing
participant Alice
participant "I have a really\nlong name" as L #99FF99
/' You can also declare:
participant L as "I have a really\nlong name" #99FF99
'/
```

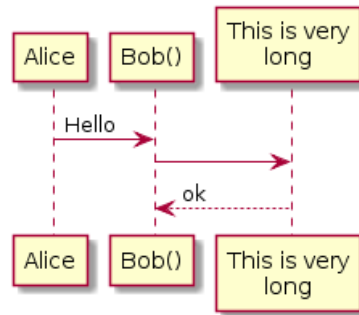
```
Alice->>Bob: Authentication Request
Bob->>Alice: Authentication Response
Bob->>L: Log transaction
@enduml
```



1.4 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()" ->>"This is very\nlong" as Long
' You can also declare:
' "Bob()" ->> Long as "This is very\nlong"
Long -->>"Bob()" : ok
@enduml
```



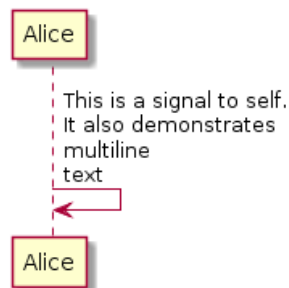
1.5 Message à soi-même

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: This is a signal to self.\nIt also demonstrates\nmultiline \ntext
@enduml
  
```



1.6 Autre style de flèches

Vous pouvez changer les flèches de plusieurs façons :

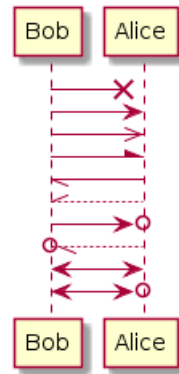
- Pour indiquer un message perdu, terminer la flèche avec `x`
- Utiliser `\` ou `/` à la place de `<` ou `>` pour avoir seulement la partie supérieure ou inférieure de la flèche.
- Doubler un des caractères (par exemple, `>>` ou `//`) pour avoir une flèche plus fine.
- Utiliser `--` à la place de `-` pour avoir des pointillés.
- Utiliser `o` après la flèche
- Utiliser une flèche bi-directionnelle

```

@startuml
Bob ->x Alice
Bob -> Alice
Bob ->> Alice
Bob -\ Alice
Bob \- Alice
Bob //-- Alice

Bob ->o Alice
Bob o\-- Alice

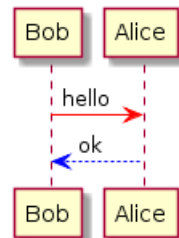
Bob <-> Alice
Bob <->o Alice
@enduml
  
```



1.7 Changer la couleur des flèches

Changer la couleur d'une flèche ainsi:

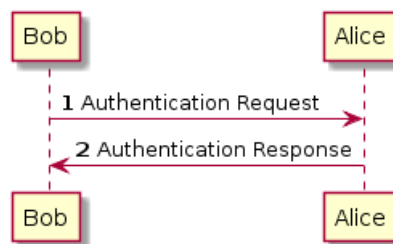
```
@startuml
Bob -[#red]> Alice : hello
Alice -[#0000FF]->Bob : ok
@enduml
```



1.8 Numérotation automatique des messages

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

```
@startuml
autonumber
Bob -> Alice : Authentication Request
Bob <- Alice : Authentication Response
@enduml
```



Spécifier le numéro de départ avec `autonumber 'start'`, et l'incrément avec `autonumber 'start' 'increment'`.

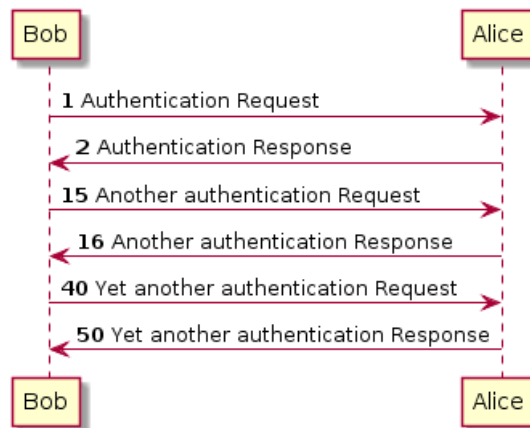
```
@startuml
autonumber
Bob -> Alice : Authentication Request
Bob <- Alice : Authentication Response

autonumber 15
Bob -> Alice : Another authentication Request
Bob <- Alice : Another authentication Response

autonumber 40 10
Bob -> Alice : Yet another authentication Request
Bob <- Alice : Yet another authentication Response

@enduml
```





Spécifier le format d'un nombre entre guillemets anglais.

Le formatage est fait par la classe `DecimalFormat` ('0' signifie un chiffre, '#' signifie un chiffre ou zéro si absent).

Des balises HTML sont permises dans le format.

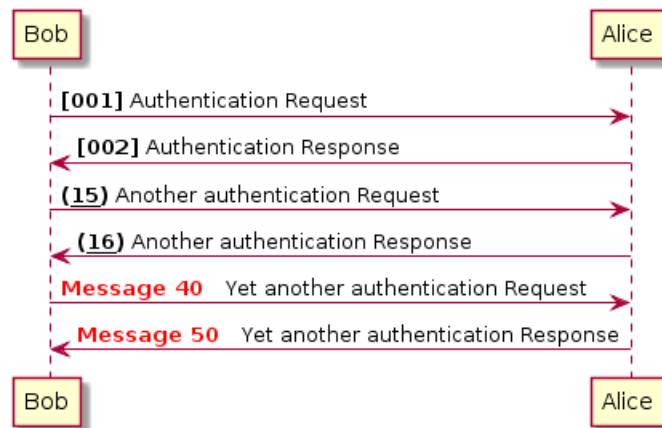
```

@startuml
autonumber "<b>[000]"
Bob -> Alice : Authentication Request
Bob <- Alice : Authentication Response

autonumber 15 "<b>(<u>##</u>)"
Bob -> Alice : Another authentication Request
Bob <- Alice : Another authentication Response

autonumber 40 10 "<font color=red><b>Message 0 "
Bob -> Alice : Yet another authentication Request
Bob <- Alice : Yet another authentication Response

@enduml
  
```



1.9 Titre

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

```

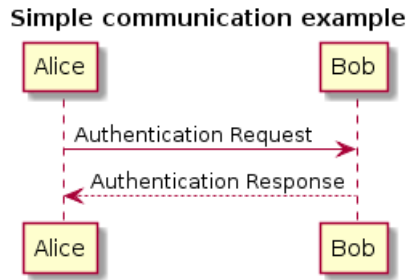
@startuml

title Simple communication example

Alice -> Bob: Authentication Request
Bob --> Alice: Authentication Response

@enduml
  
```





1.10 Spécifier la légende d'un diagramme

Les mots-clés `legend` et `end legend` délimitent la spécification d'une légende.

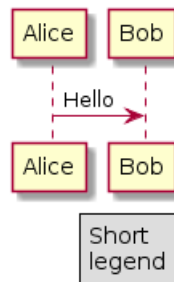
Utiliser `left`, `right` ou `center` pour aligner la légende à gauche, à droite ou au centre respectivement.

```

@startuml

Alice -> Bob : Hello
legend right
Short
legend
endlegend

@enduml
  
```



1.11 Découper un diagramme

Le mot clé `newpage` est utilisé pour découper un digramme 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 digrammes sur plusieurs pages.

```

@startuml

Alice -> Bob : message 1
Alice -> Bob : message 2

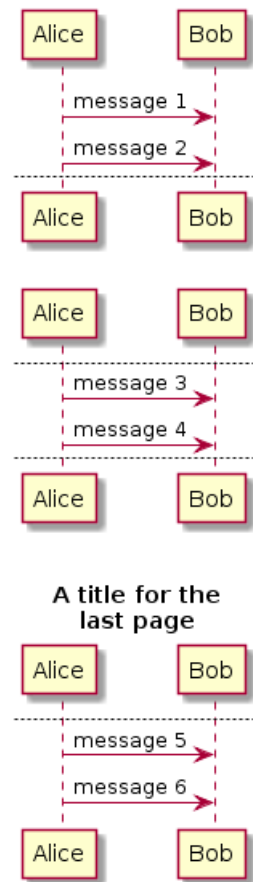
newpage

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

newpage A title for the\nlast page

Alice -> Bob : message 5
Alice -> Bob : message 6

@enduml
  
```



1.12 Regrouper les messages (cadres UML)

Il est possible de regrouper les messages dans un cadre UML 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.

Terminer le cadre avec le mot-clé `end`.

Il est possible d'imbriquer les cadres.

```

@startuml
Alice -> Bob: Authentication Request

alt successful case
    Bob -> Alice: Authentication Accepted
else some kind of failure
    Bob -> Alice: Authentication Failure
    group My own label
        Alice -> Log : Log attack start
        loop 1000 times
  
```




```

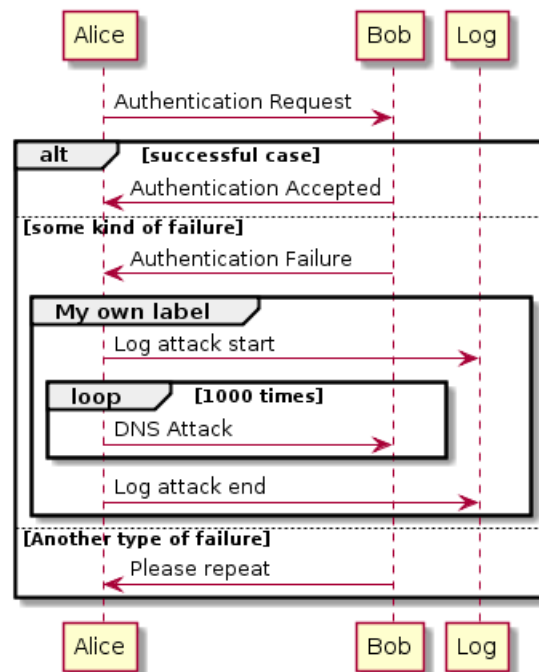
Alice -> Bob: DNS Attack
end
Alice -> Log : Log attack end
end

else Another type of failure

Bob -> Alice: Please repeat

end
@enduml

```



1.13 Note sur les messages

Pour attacher une note à un message, utiliser les mots-clés `note left` (pour une note à gauche) ou `note right` (pour une note à droite) *juste après le message*.

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

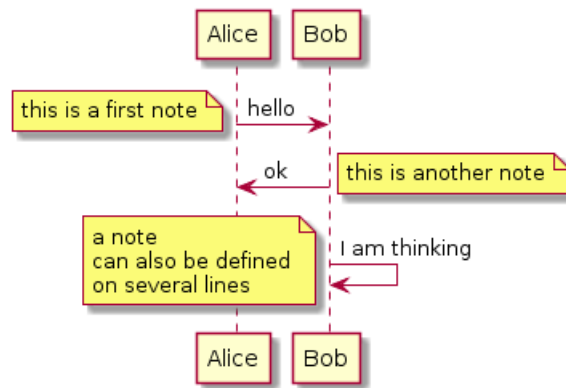
```

@startuml
Alice->>Bob : hello
note left: this is a first note

Bob->>Alice : ok
note right: this is another note

Bob->>Bob : I am thinking
note left
a note
can also be defined
on several lines
end note
@enduml

```



1.14 Encore plus de notes

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

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**.

```

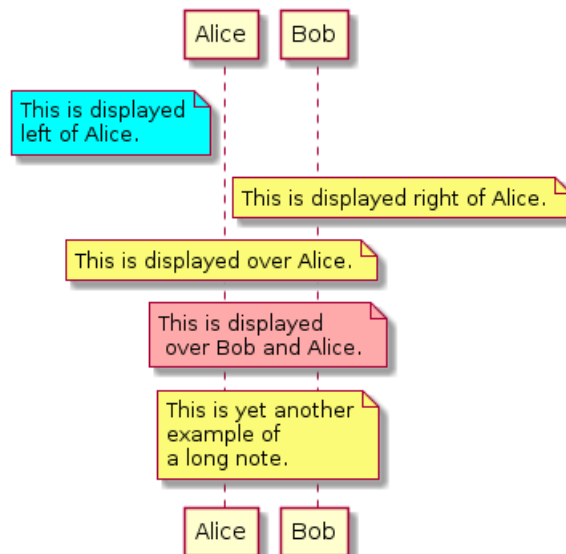
@startuml
participant Alice
participant Bob
note left of Alice #aqua
This is displayed
left of Alice.
end note

note right of Alice: This is displayed right of Alice.

note over Alice: This is displayed over Alice.

note over Alice, Bob #FFAAAA: This is displayed\n over Bob and Alice.

note over Bob, Alice
This is yet another
example of
a long note.
end note
@enduml
  
```



1.15 Changer l'aspect des notes

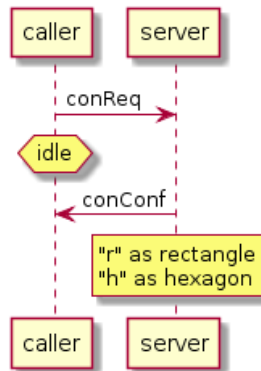
Vous pouvez préciser la forme géométrique des notes. (**rnote** : rectangulaire, ou **hnote** : hexagonale)



```

@startuml
caller -> server : conReq
note over caller : idle
caller <- server : conConf
note over server
"r" as rectangle
"h" as hexagon
endnote
@enduml

```



1.16 Créole (langage de balisage léger) et HTML

Il est également possible d'utiliser le formatage créole (langage de balisage léger):

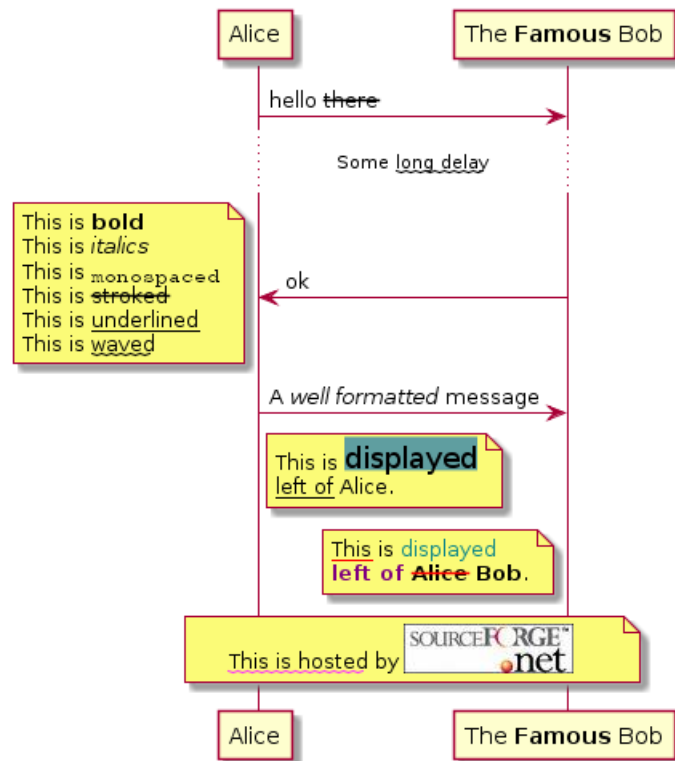
```

@startuml
participant Alice
participant "The Famous Bob" as Bob

Alice -> Bob : hello --there--
... Some ~~long delay~~ ...
Bob -> Alice : ok
note left
This is bold
This is italics
This is "monospaced"
This is --stroked--
This is underlined
This is ~waved~
end note

Alice -> Bob : A //well formatted// message
note right of Alice
This is <back:cadetblue><size:18>displayed</size></back>
__left of__ Alice.
end note
note left of Bob
<u:red>This</u> is <color #118888>displayed</color>
<color purple>left of</color> <s:red>Alice</strike> Bob.
end note
note over Alice, Bob
<w:#FF33FF>This is hosted</w> by <img sourceforge.jpg>
end note
@enduml

```



1.17 Séparation

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

```
@startuml
```

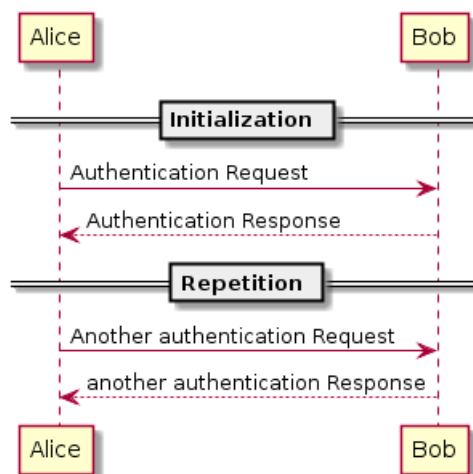
```
== Initialization ==
```

```
Alice -> Bob: Authentication Request
Bob --> Alice: Authentication Response
```

```
== Repetition ==
```

```
Alice -> Bob: Another authentication Request
Alice <-- Bob: another authentication Response
```

```
@enduml
```



1.18 Référence

You can use reference in a diagram, using the keyword **ref** over.



```

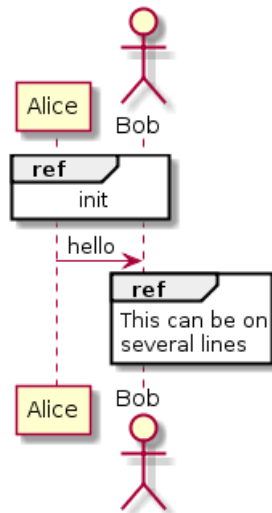
@startuml
participant Alice
actor Bob

ref over Alice, Bob : init

Alice -> Bob : hello

ref over Bob
This can be on
several lines
end ref
@enduml

```



1.19 Retard

Utiliser ... pour indiquer le passage de temps arbitraire dans le diagramme. Un message peut être associé à un retard.

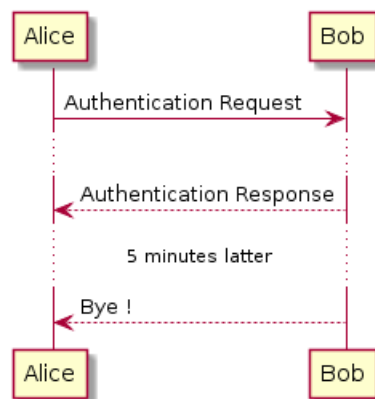
```

@startuml

Alice -> Bob: Authentication Request
...
Bob --> Alice: Authentication Response
...5 minutes latter...
Bob --> Alice: Bye !

@enduml

```



1.20 Séparation verticale

Utiliser ||| pour créer un espace vertical dans le diagramme.

Il est également possible de spécifier un nombre de pixels pour la séparation verticale.

```
@startuml
```

```
Alice -> Bob: message 1
Bob --> Alice: ok
|||
Alice -> Bob: message 2
Bob --> Alice: ok
||45||
Alice -> Bob: message 3
Bob --> Alice: ok
```

```
@enduml
```



1.21 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
```

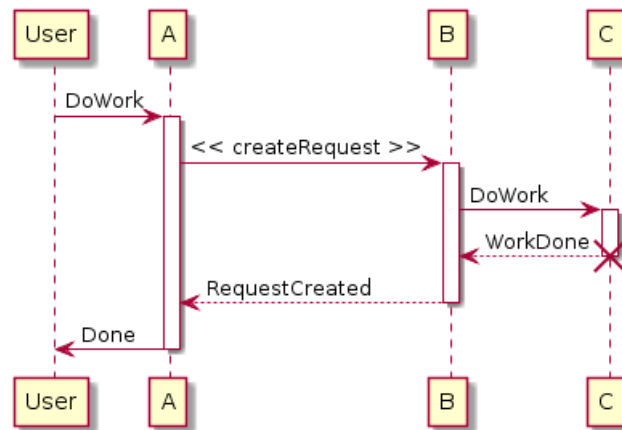
```
B -> C: DoWork
activate C
C --> B: WorkDone
destroy C
```

```
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

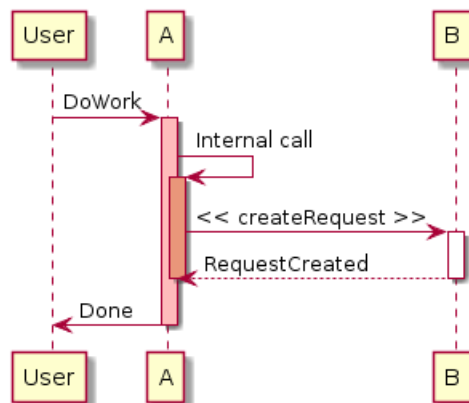
User -> A: DoWork
activate A #FFBBBB

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
```



1.22 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 nouveau objet.

```
@startuml
Bob -> Alice : hello

create Other
Alice -> Other : new

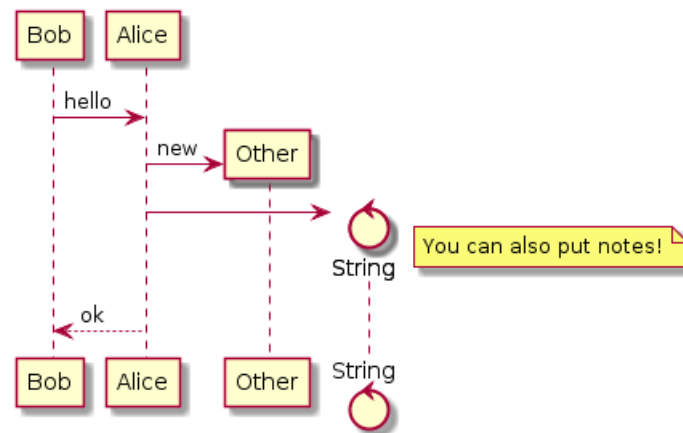
create control String
Alice -> String
note right : You can also put notes!
```



```

Alice --> Bob : ok
@enduml

```



1.23 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

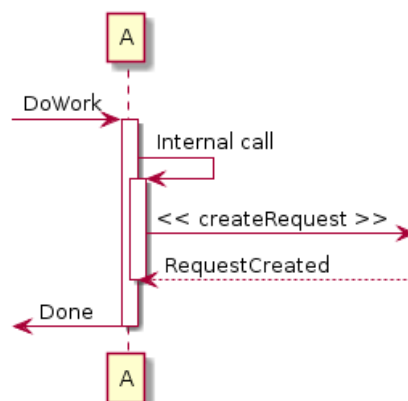
activate A

A -> A: Internal call
activate A

A ->] : << createRequest >>

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

```



Vous pouvez aussi utiliser la syntaxe suivante:

```

@startuml
[-> Bob
[o-> Bob
[o->o Bob
[x-> Bob

```




```

[<- Bob
[x<- Bob

Bob ->]
Bob ->o]
Bob o->o]
Bob ->x]

Bob <-]
Bob x<-]
@enduml

```



1.24 Stéréotypes et décoration

Il est possible de rajouter un stéréotype aux participants en utilisant "<<" et ">>".

Dans le stéréotype, vous pouvez ajouter un caractère entouré d'un cercle coloré en utilisant la syntaxe (X,couleur).

```

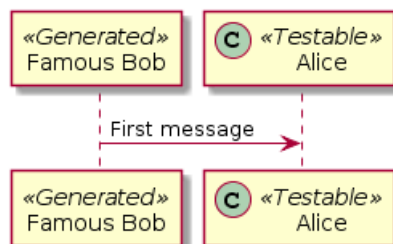
@startuml

participant "Famous Bob" as Bob << Generated >>
participant Alice << (C,#ADD1B2) Testable >>

Bob->>Alice: First message

@enduml

```



```

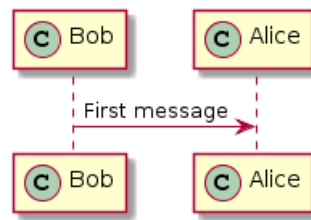
@startuml

participant Bob << (C,#ADD1B2) >>
participant Alice << (C,#ADD1B2) >>

Bob->>Alice: First message

@enduml

```



1.25 Plus d'information sur les titres

Vous pouvez utiliser le formatage creole dans le titre.

```

@startuml

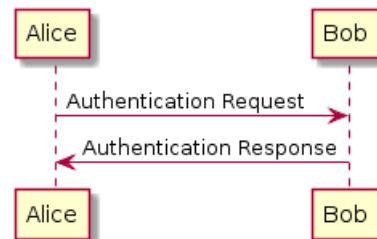
title __Simple__ **communication** example

Alice -> Bob: Authentication Request
Bob -> Alice: Authentication Response

@enduml

```

Simple communication example



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

```

@startuml

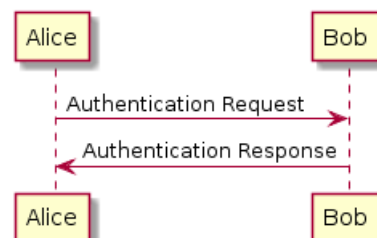
title __Simple__ communication example\nnon several lines

Alice -> Bob: Authentication Request
Bob -> Alice: Authentication Response

@enduml

```

**Simple communication example
on several lines**



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

```

@startuml

title
<u>Simple</u> communication example
on <i>several</i> lines and using <font color=red>html</font>
This is hosted by <img:sourceforge.jpg>
end title

Alice -> Bob: Authentication Request

```



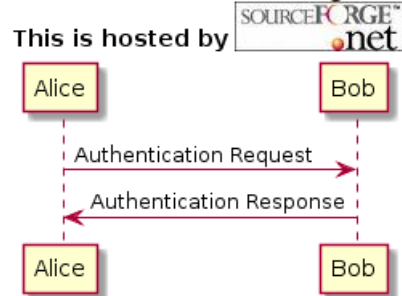
```

Bob -> Alice: Authentication Response

@enduml

```

**Simple communication example
on several lines and using **html****



1.26 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`.

```

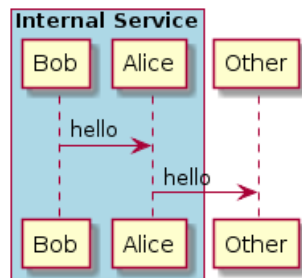
@startuml

box "Internal Service" #LightBlue
    participant Bob
    participant Alice
end box
participant Other

Bob -> Alice : hello
Alice -> Other : hello

@enduml

```



1.27 Supprimer les en-pieds

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

```

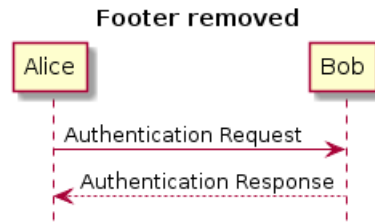
@startuml

hide footbox
title Footer removed

Alice -> Bob: Authentication Request
Bob --> Alice: Authentication Response

@enduml

```



1.28 Personnalisation

Vous pouvez utiliser la commande **skinparam** pour changer les couleurs 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 #EEEEBD

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 Apex
}

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

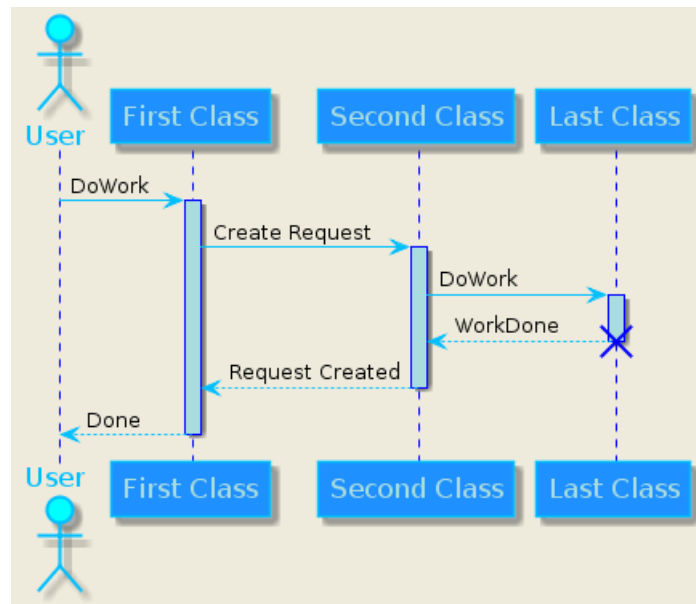
B --> A: Request Created
deactivate B

A --> User: Done
deactivate A

@enduml

```





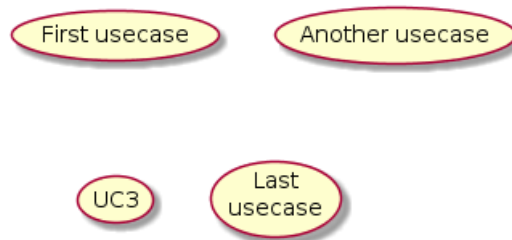
2 Diagramme de cas d'utilisation

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
```



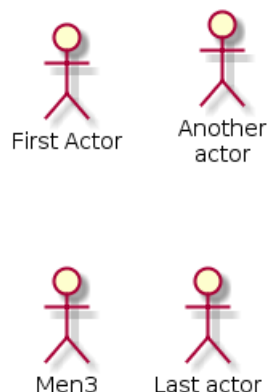
2.2 Acteurs

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 Description des cas d'utilisation

Si vous voulez une description sur plusieurs lignes, vous pouvez utiliser des guillemets.

Vous pouvez aussi utiliser les séparateurs suivants: -- .. == __. Et vous pouvez mettre un titre dans les séparateurs.



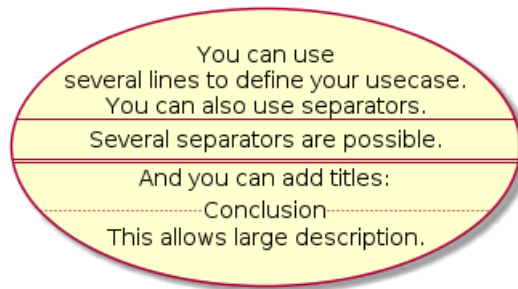
```

@startuml

usecase UC1 as "You can use
several lines to define your usecase.
You can also use separators.
--
Several separators are possible.
==
And you can add titles:
..Conclusion..
This allows large description."

@enduml

```



2.4 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. Vous pouvez ajouter un libellé sur la flèche, en ajoutant un caractère ":" dans la définition de la flèche.

Dans cet exemple, vous voyez que *User* n'a pas été défini préalablement, et qu'il est implicitement reconnu comme acteur.

```

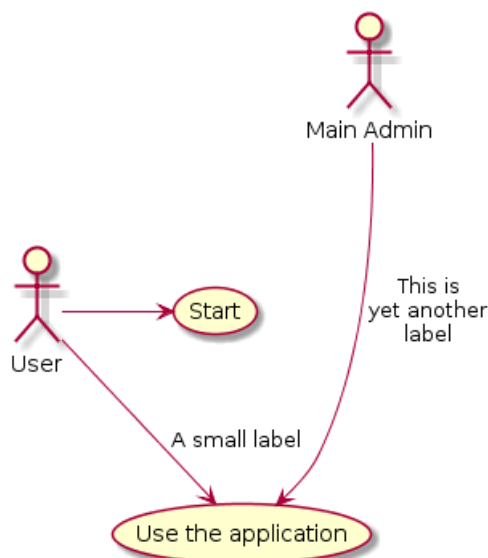
@startuml

User -> (Start)
User --> (Use the application) : A small label

:Main Admin: ---> (Use the application) : This is\nyet another\nlabel

@enduml

```



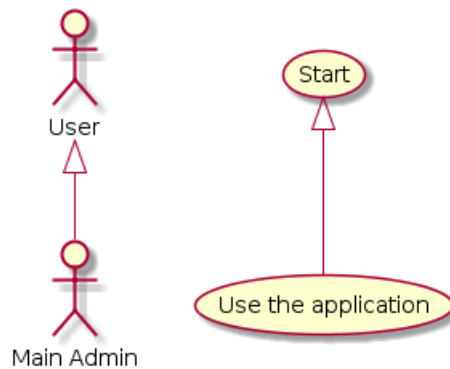
2.5 Héritage

Si un acteur ou un cas d'utilisation en étend un autre, vous pouvez utiliser le symbole

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

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

@enduml
```



2.6 Notes

Vous pouvez utiliser les mots clés `note left of` , `note right of` , `note top of` , `note bottom of` pour définir les notes en relation avec un objet.

Une note peut également être définie seule avec des mots-clés, puis liée à d'autres objets en utilisant le symbole `..` .

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

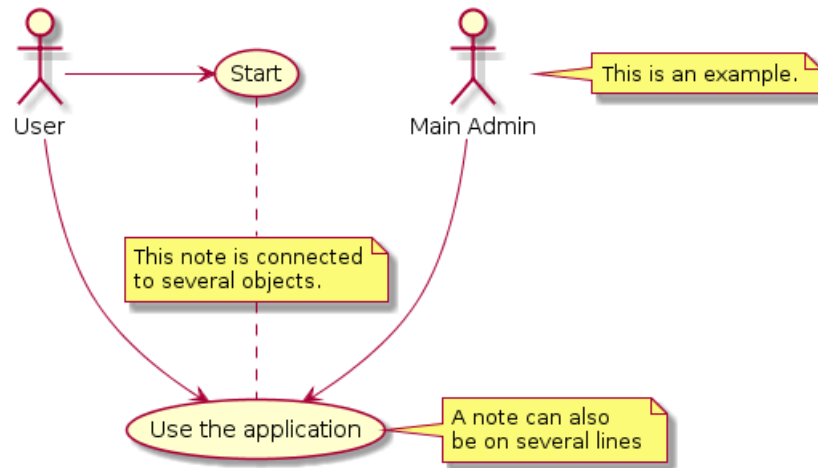
User -> (Start)
User --> (Use)

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 "This note is connected\nto several objects." as N2
(Start) .. N2
N2 .. (Use)
@enduml
```

2.7 Stéréotypes

Vous pouvez ajouter des stéréotypes à la définition des acteurs et des cas d'utilisation avec "<<" et ">>".

```

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

```

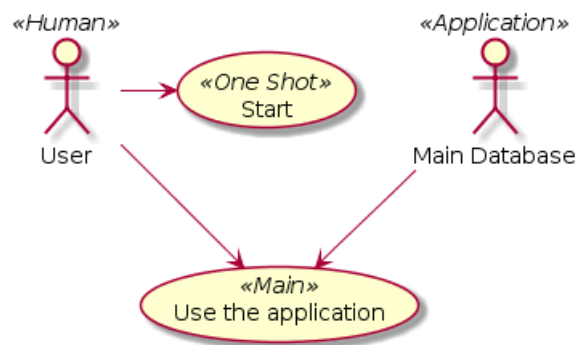
User -> (Start)
User --> (Use)
  
```

```

MySQL --> (Use)
  
```

```

@enduml
  
```

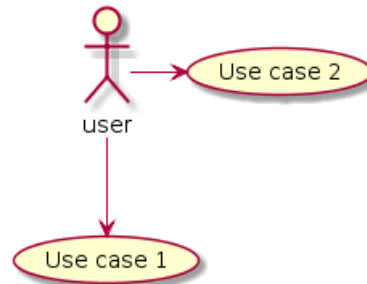


2.8 Changer les directions des flèches

Par défaut, les liens entre les classes ont deux tirets -- et sont orientés verticalement. Il est possible de mettre des liens horizontaux en mettant un seul tiret (ou un point) comme ceci:

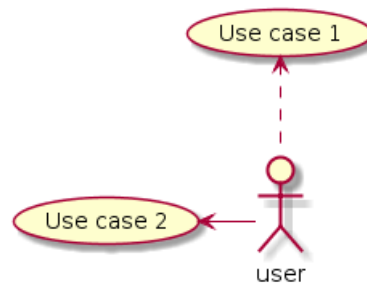
```

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



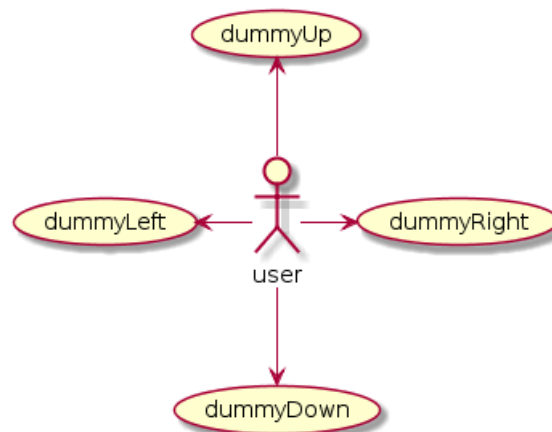
Vous pouvez aussi changer le sens en renversant le lien :

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



Il est possible de changer la direction d'une flèche en utilisant les mots-clé **left**, **right**, **up** ou **down** à l'intérieur de la flèche :

```
@startuml
:user: -left-> (dummyLeft)
:user: -right-> (dummyRight)
:user: -up-> (dummyUp)
:user: -down-> (dummyDown)
@enduml
```



Vous pouvez abréger les noms des flèches en indiquant seulement le premier caractère de la direction (par exemple **-d-** pour **-down-**) ou les deux premiers caractères (**-do-**).

Il est conseillé de ne pas abuser de cette fonctionnalité : *Graphviz* qui donne d'assez bon résultats quoique non

2.9 Titrer le diagramme

Le mot-clé **title** est utilisé pour mettre un titre.

Vous pouvez utiliser les commandes **title** et **end title** pour un titre plus long, comme dans les diagrammes de séquence.



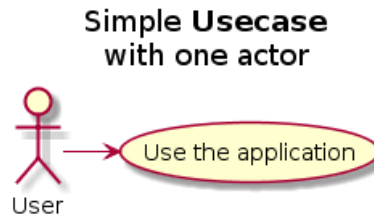
```

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

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

@enduml

```



2.10 Découper les diagrammes

Le mot-clé `newpage` est utilisé pour découper un diagramme en plusieurs images.

```

@startuml
:actor1: --> (Usecase1)
newpage
:actor2: --> (Usecase2)
@enduml

```



2.11 De droite à gauche

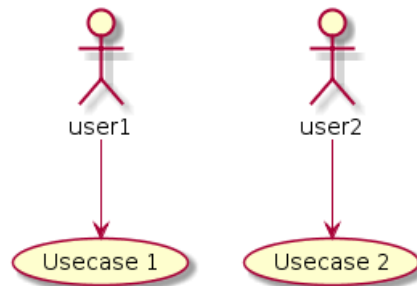
Le comportement général de construction des diagrammes est de haut en bas.

```

@startuml
'default
top to bottom direction
user1 --> (Usecase 1)
user2 --> (Usecase 2)

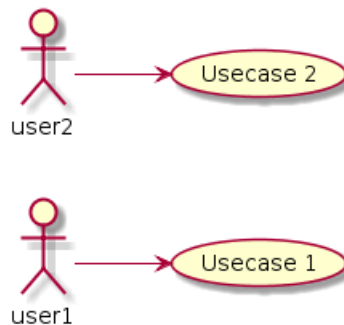
@enduml

```



Il est possible de changer pour aller plutôt de la droite vers la gauche avec la commande `left to right direction`. Le résultat est parfois meilleur dans ce cas.

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



2.12 La commande Skinparam

Utilisez la commande `skinparam` pour changer la couleur et la mise en forme du texte du schéma.

Vous pouvez utiliser cette commande :

- Dans la définition du diagramme, comme pour les autres commandes,
- Dans un fichier inclus,
- Dans un fichier de configuration, renseigné dans la ligne de commande ou la tâche ANT.

Vous pouvez aussi spécifier les polices et les couleurs pour les acteurs et cas d'utilisation avec des stéréotypes.

```
@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 >>
```



```

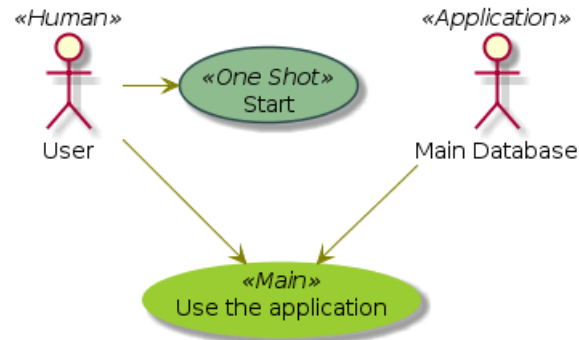
(Start) << One Shot >>
(Use the application) as (Use) << Main >>

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

MySQL --> (Use)

@enduml

```

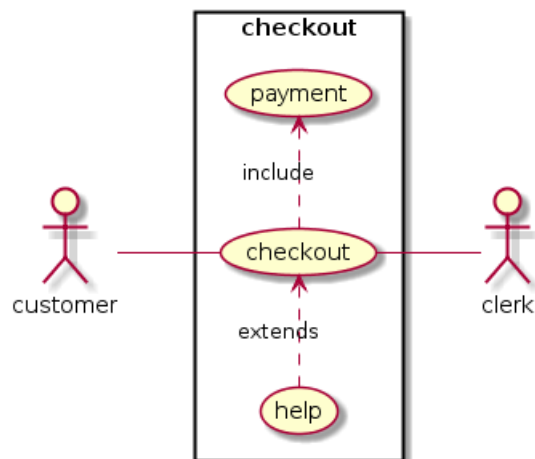


2.13 Exemple complet

```

@startuml
left to right direction
skinparam packageStyle rect
actor customer
actor clerk
rectangle checkout {
customer -- (checkout)
(checkout) .> (payment) : include
(help) .> (checkout) : extends
(checkout) -- clerk
}
@enduml




```



3 Diagramme de classes

3.1 Relations entre classes

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

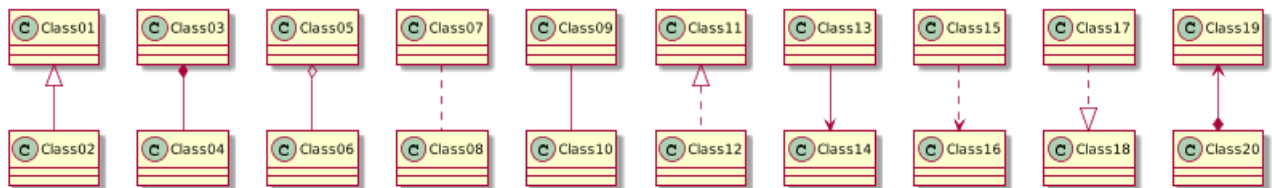
Extension	< --	
Composition	*--	
Agrégation	o--	

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

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

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

```
@startuml
scale 800 width
Class01 <|-- Class02
Class03 *-- Class04
Class05 o-- Class06
Class07 .. Class08
Class09 -- Class10
Class11 <|.. Class12
Class13 --> Class14
Class15 ..> Class16
Class17 ..|> Class18
Class19 <--* Class20
@enduml
```

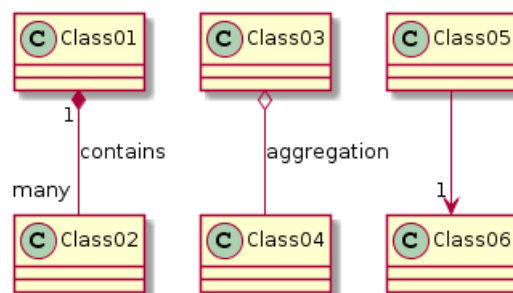


3.2 Libellés sur les relations

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

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

```
@startuml
Class01 "1" *-- "many" Class02 : contains
Class03 o-- Class04 : aggregation
Class05 --> "1" Class06
@enduml
```



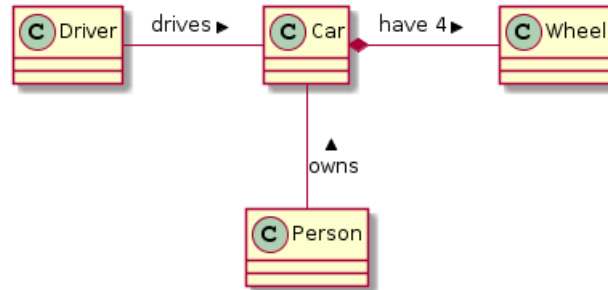
Vous pouvez ajouter une flèche désignant quel objet agit sur l'autre en utilisant < ou > au début ou à la fin du libellé.



```
@startuml
class Car

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

@enduml
```



3.3 Définir les méthodes

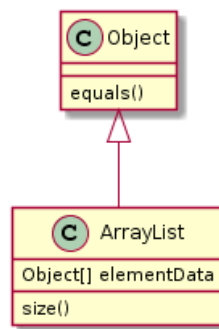
Pour déclarer des méthodes ou des champs, vous pouvez utiliser le caractère "deux-points" followed by the field's or method's name.

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

```
@startuml
Object <|-- ArrayList

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

@enduml
```



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

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

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

class Flight {
flightNumber : Integer
departureTime : Date
}

@enduml
```

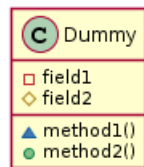


3.4 Définir les visibilité

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

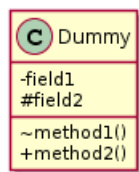
-	□	■	privé
#	◇	◆	protégé
~	△	▲	protégé package
+	○	●	publique

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



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

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

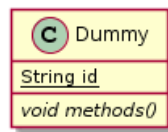


3.5 Abstrait et statique

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

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

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



3.6 Corps de classe avancé

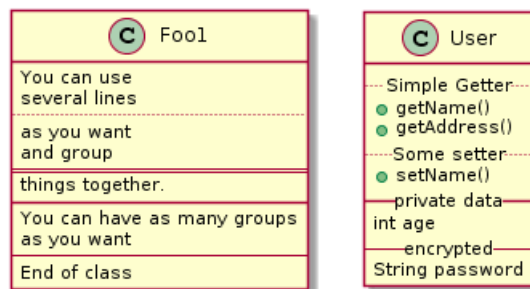
Par défaut, méthodes et champs sont automatiquement regroupé par PlantUML. Vous pouvez utiliser un séparateur pour définir votre propre manière d'ordonner les champs et les méthodes. Les séparateurs suivants sont possibles : -- .. == __.

Vous pouvez aussi utiliser les titres dans les séparateurs.

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

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

@enduml
```



3.7 Notes et stéréotypes

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

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

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

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

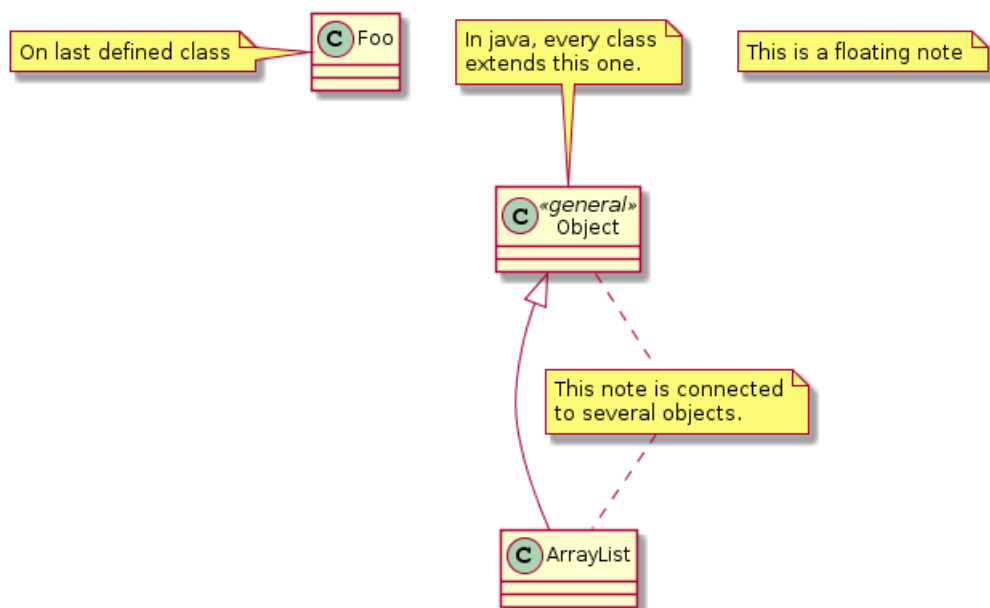
```
...
@startuml
class Object << general >>
Object <|--- ArrayList

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

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

class Foo
note left: On last defined class

@enduml
```



3.8 Encore des notes

Il est possible d'utiliser quelques tag HTML comme :

- ``
- `<u>`
- `<i>`
- `<s>`, ``, `<strike>`
- `` or ``
- `<color:AAAAAA>` or `<color:colorName>`
- `<size:nn>` to change font size
- `` 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.

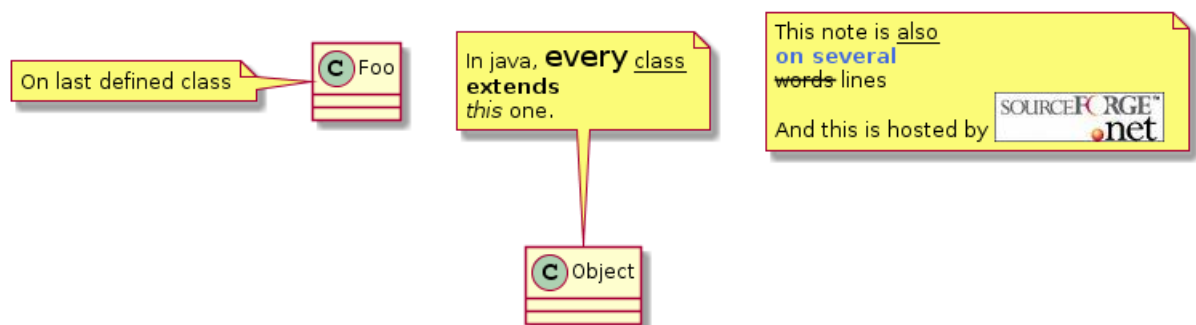
```
@startuml

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
```



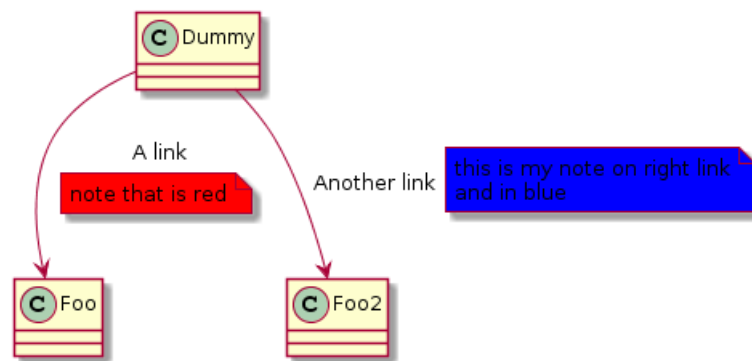
3.9 Note sur les liens

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

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

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

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



3.10 Classe abstraite et Interface

Vous pouvez déclarer une classe abstraite en utilisant "abstract" ou "abstract class". La classe sera alors é

Vous pouvez aussi utiliser interface, annotation et enum.

```
@startuml

abstract class AbstractList
abstract AbstractCollection
interface List
interface Collection

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

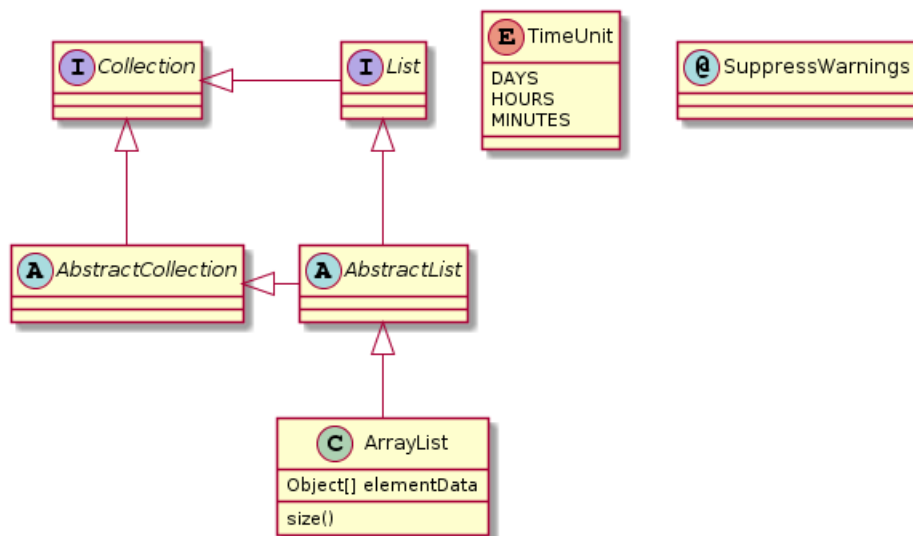
Collection <|-- List
AbstractCollection <|-- AbstractList
AbstractList <|-- ArrayList

class ArrayList {
Object[] elementData
size()
}

enum TimeUnit {
DAYS
HOURS
MINUTES
}

annotation SuppressWarnings

@enduml
```



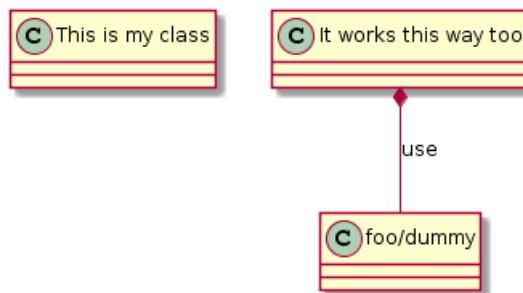
3.11 Caractères non alphabétiques

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

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

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

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



3.12 Masquer les attributs et les méthodes

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

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

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

- `empty fields` ou `empty attributes` pour des champs vides,
- `empty methods` pour des méthodes vides,
- `fields` or `attributes` qui masque les champs, même s'il y en a de définis,
- `methods` qui masque les méthodes, même s'il y en a de définies,
- `members` qui masque les méthodes ou les champs, même s'il y en a de définies,
- `circle` pour le caractère entouré en face du nom de la classe,
- `stereotype` pour le stéréotype.

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

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

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

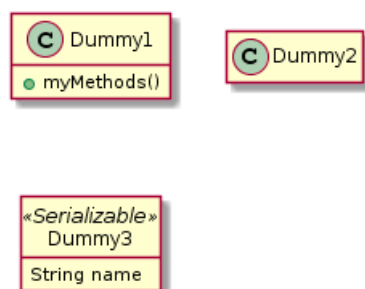
```
@startuml
class Dummy1 {
+myMethods()
}

class Dummy2 {
+hiddenMethod()
}

class Dummy3 <<Serializable>> {
String name
}

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

@enduml
```



3.13 Cacher des classes

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

Cela peut être utile si vous définissez un large fichier `!included`, et si vous voulez en cacher quelques unes après une inclusion de fichier.

```
@startuml
class Foo1
class Foo2

Foo2 *-- Foo1

hide Foo2

@enduml
```

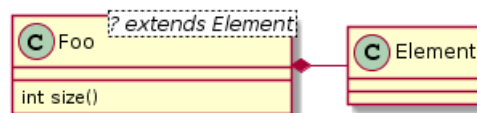


3.14 Utilisation de la généricité

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

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

@enduml
```

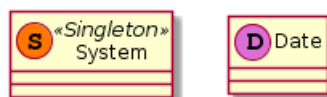


3.15 Caractère spécial

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

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

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



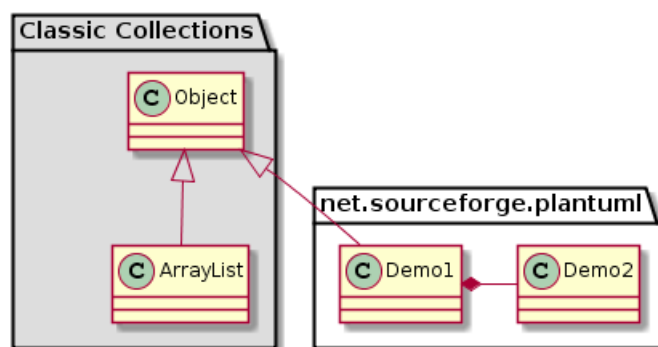
3.16 Packages

Vous pouvez définir un package en utilisant le mot-clé **package**, et optionnellement déclarer une couleur de fond pour le package. Notez que les définitions de packages peuvent être imbriquées.

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

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

@enduml
```



3.17 Modèle de paquet

Il y a différents styles de paquets disponibles.

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

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

package foo2 <<Rect>> {
class Class2
}

package foo3 <<Folder>> {
class Class3
}

package foo4 <<Frame>> {
class Class4
}

package foo5 <<Cloud>> {
class Class5
}

package foo6 <<Database>> {
class Class6
}

@enduml
```



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

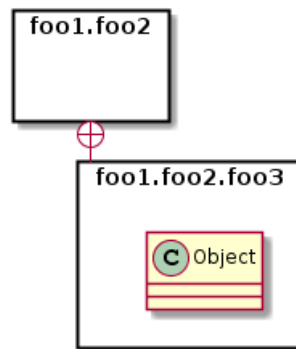
```
@startuml
skinparam packageStyle rect

package foo1.foo2 {

package foo1.foo2.foo3 {
class Object
}

foo1.foo2 +-- foo1.foo2.foo3

@enduml
```



3.18 Les espaces de noms

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.

Dans ce cas, vous pouvez utiliser les espaces de noms à la place des packages.

Vous pouvez faire référence à des classes d'autres espace de nom en les nommant complètement. Les classes de l'espace de nom par défaut sont nommées en commençant par un point.

Note that you don't have to explicitly create namespace : a fully qualified class est automatiquement ajouté au bon espace de nom.

```
@startuml

class BaseClass

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

.BaseClass <|-- Meeting
}

namespace net.foo {
```

```

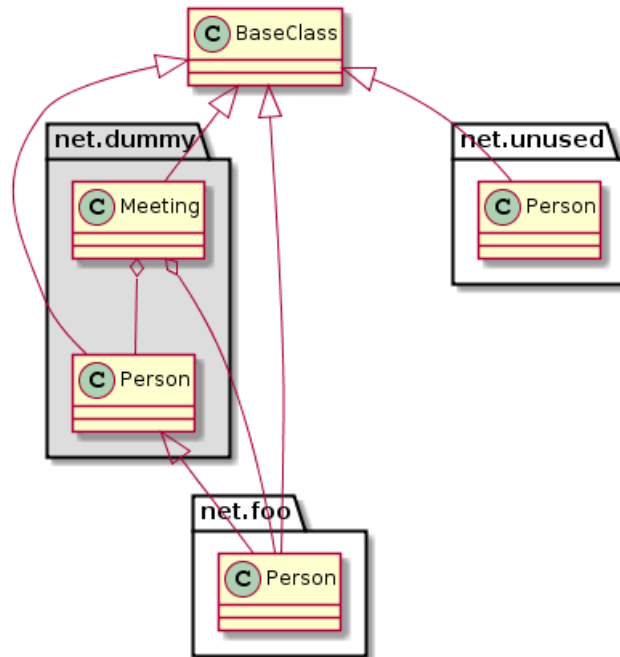
net.dummy.Person <|-- Person
.BaseClass <|-- Person

net.dummy.Meeting o-- Person
}

BaseClass <|-- net.unused.Person

@enduml

```



3.19 Creation automatique d'espace de nommage

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

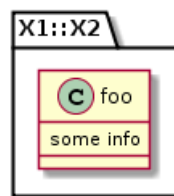
```
@startuml
```

```

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

```

```
@enduml
```



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

```
@startuml
```

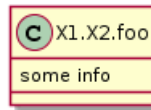
```

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

```

```
@enduml
```



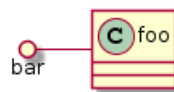


3.20 Interface boucle

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

```

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

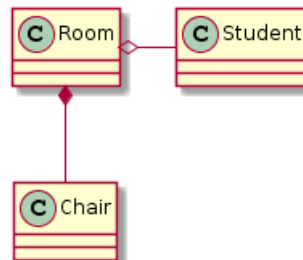


3.21 Changer la direction

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

```

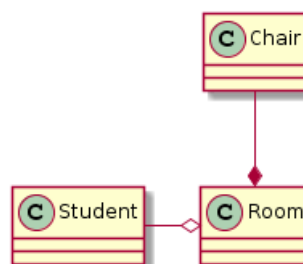
@startuml
Room o- Student
Room *-- Chair
@enduml
  
```



Vous pouvez aussi changer le sens en renversant le lien :

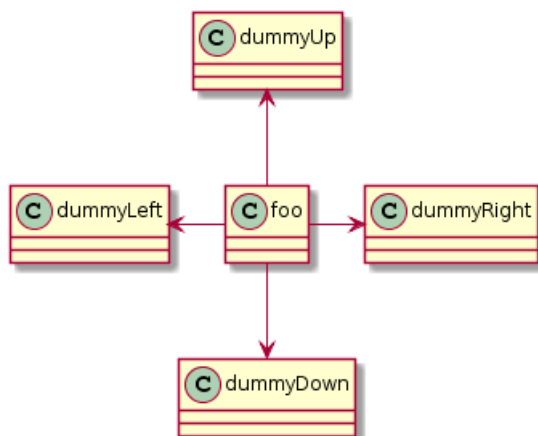
```

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



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

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



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

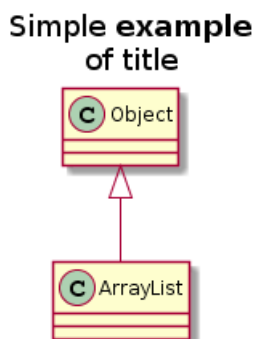
Attention à ne pas abuser de cette fonctionnalité : *Graph Viz* donne généralement de bons résultats sans trop

3.22 Titre de diagramme

Le mot clé `title` est à utiliser pour mettre un titre.

Vous pouvez utiliser les mots clés `title` et `end title` pour un titre plus long, comme dans un diagramme de séquence.

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



3.23 Diagramme de légende

Les mots clés `legend` et `end legend` sont utilisés pour mettre une légende.

Vous pouvez spécifier une option pour avoir `left`, `right` ou `center` pour aligner la légende.



```

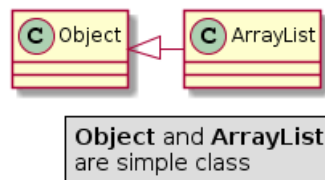
@startuml

Object <|-- ArrayList

legend right
<b>Object</b> and <b>ArrayList</b>
are simple class
endlegend

@enduml

```



3.24 Classes d'association

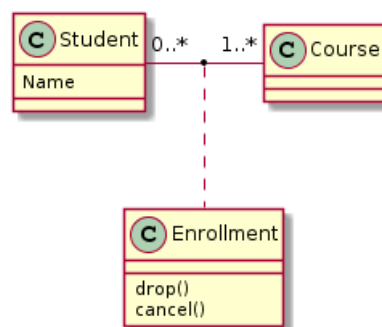
Vous pouvez définir une *classe d'association* après qu'une relation ait été définie entre deux classes, comme dans l'exemple suivant:

```

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

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

```



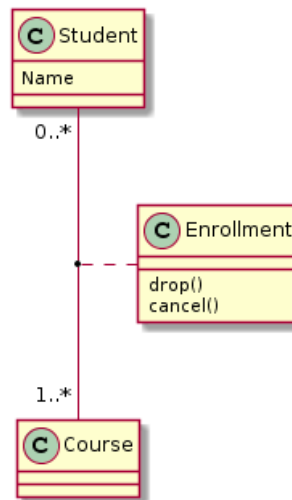
Vous pouvez la définir dans une autre direction :

```

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

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

```

3.25 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 précisé par la ligne de commande ou la tâche ANT.

```

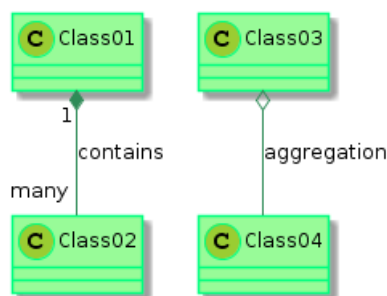
@startuml

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

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

Class03 o-- Class04 : aggregation

@enduml
  
```



3.26 Stéréotypes Personnalisés

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

```

@startuml

skinparam class {
  BackgroundColor PaleGreen
}
  
```



```

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

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

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

@enduml

```

```

[From /home/ec2-user/database/tmp/FR/classes.tex (line 1109) ]
... (skipping 12 lines) ...
Class01 << Foo >>
Class01 "1" *-- "many" Class02 : contains

Class03<<Foo>> o-- Class04 : aggregation
Syntax Error?

```

3.27 Dégradé de couleur

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

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

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

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

en fonction de la direction du dégradé

Par exemple, vous pouvez avoir :

```

@startuml

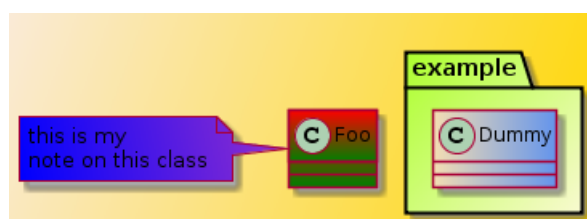
skinparam backgroundcolor AntiqueWhite/Gold
skinparam classBackgroundColor Wheat|CornflowerBlue

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

package example #GreenYellow/LightGoldenRodYellow {
class Dummy
}

@enduml

```



3.28 Découper les grands diagrammes

Parfois, vous obtiendrez des images de taille importante.

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

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

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

class BaseClass

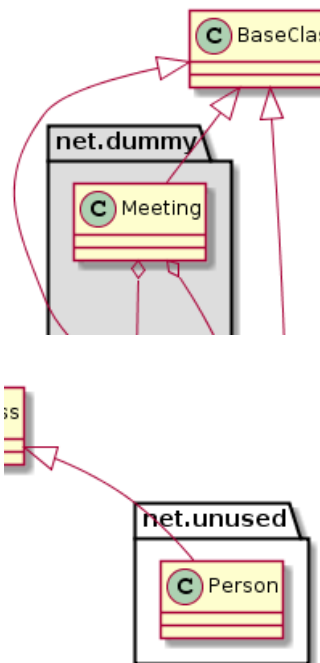
namespace net.dummy #DDDDDD {
  .BaseClass <|-- Person
  Meeting o-- Person

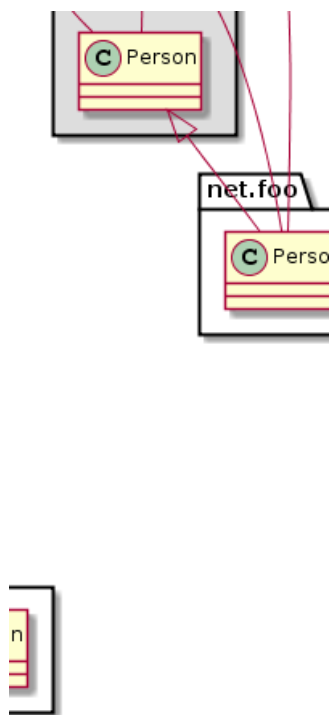
  .BaseClass <|-- Meeting
}

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

  net.dummy.Meeting o-- Person
}

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





4 Diagrammes d'activité

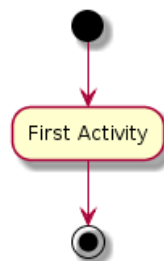
4.1 Exemple de base

Vous devez utiliser (*) pour le début et la fin du diagramme d'activité.

Dans certaines occasions, vous pourriez vouloir utiliser (*top) pour forcer le début à être en haut du diagramme.

Utiliser --> pour les flèches.

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

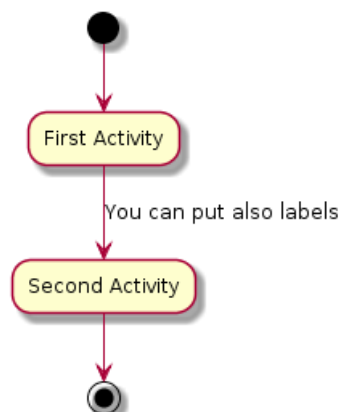


4.2 Texte sur les flèches.

Par défaut, une flèche commence à partir de la dernière activité définie.

Vous pouvez rajouter un libellé sur une flèche en mettant des crochets [et] juste après la définition de la flèche.

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



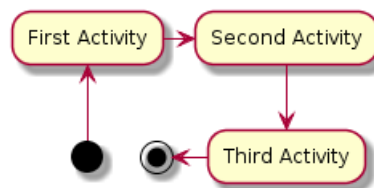
4.3 Changer la direction des flèches

Vous pouvez utiliser -> pour les flèches horizontales. Il est aussi possible de forcer la direction d'une flèche en utilisant la syntaxe suivante :



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

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

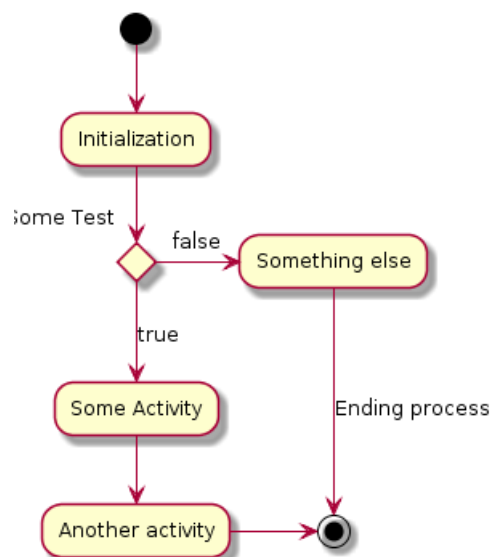


4.4 Branches

Vous pouvez utiliser le mot clé `if/then/else` pour définir une branche.

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

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

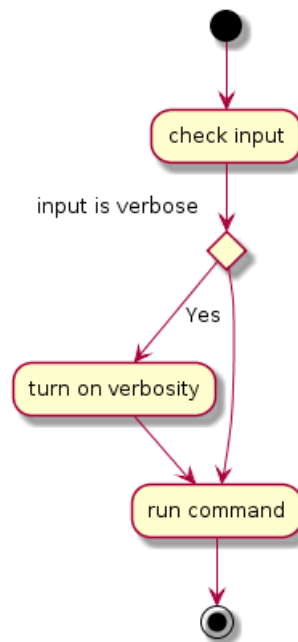


Malheureusement, vous devez parfois avoir à répéter la même activité dans le diagramme de texte.

```

@startuml
(*) --> "check input"
If "input is verbose" then
--> [Yes] "turn on verbosity"
--> "run command"
else
--> "run command"
Endif
-->(*)
@enduml

```



4.5 Encore des branches

Par défaut, une branche commence à la dernière activité définie, mais il est possible de passer outre et de définir un lien avec le mot clé `if`.

Il est aussi possible d'imbriquer les 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

else
->[false] "activity 2"

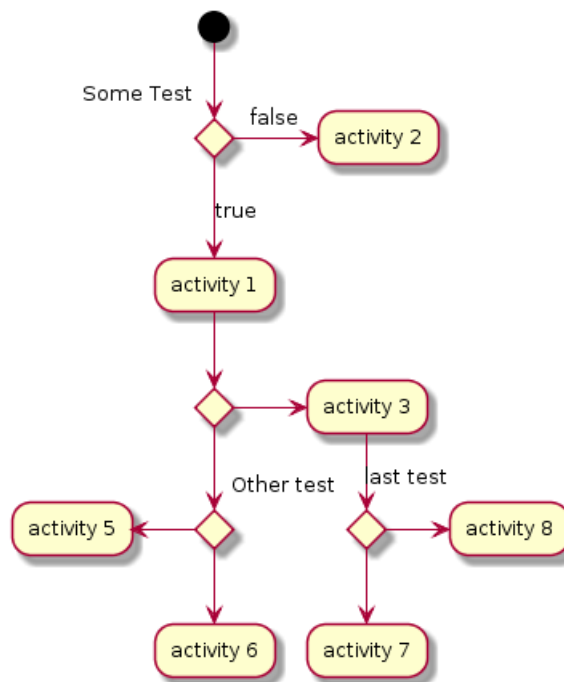
endif

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

```



```
endif
@enduml
```



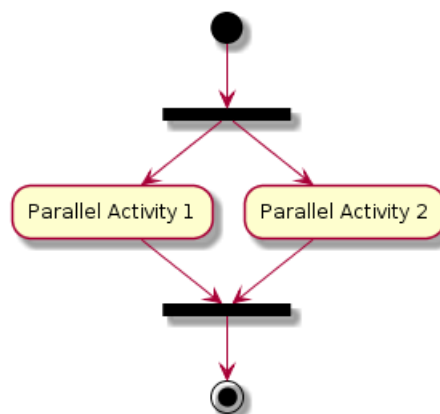
4.6 Synchronisation

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 Description détaillée

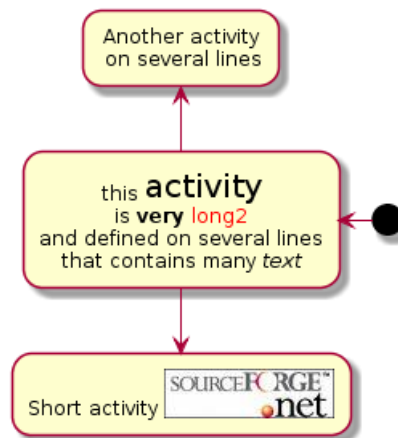
Lorsque vous déclarez des activités, vous pouvez positionner sur plusieurs lignes le texte de description. Vous pouvez également ajouter `\n` dans la description. Il est également possible d'utiliser quelques tags HTML tels que :

Vous pouvez aussi donner un court code à l'activité avec le mot clé `as`. Ce code peut être utilisé plus tard dans le diagramme de description.

```
@startuml
(*) -left-> "this <size:20>activity</size>
is <b>very</b> <color:red>long2</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:sourceforge.jpg>"
@enduml
```



4.8 Notes

Vous pouvez rajouter des notes sur une activités en utilisant les commandes: `note left`, `note right`, `note top` or `note bottom`, juste après la définition de l'activité concernée.

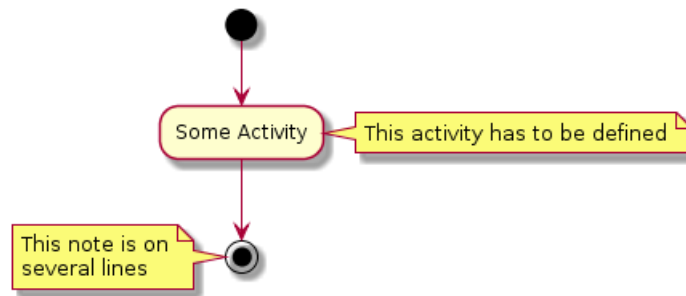
Si vous voulez mettre une note sur le démarrage du diagramme, définissez la note au tout début du diagramme.

Vous pouvez aussi avoir une note sur plusieurs lignes, en utilisant les mots clés `endnote`.

```
@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

Vous pouvez définir une partition en utilisant le mot clé **partition**, et optionnellement déclarer un fond de couleur pour votre partition (En utilisant un code couleur html ou un nom)

Quand vous déclarez les activités, ils sont automatiquement mis dans la dernière partition utilisée.

Vous pouvez fermer la partition de définition en utilisant les crochets fermants }.

```

@startuml

partition Conductor {
(*) --> "Climbs on Platform"
--> === S1 ===
--> Bows
}

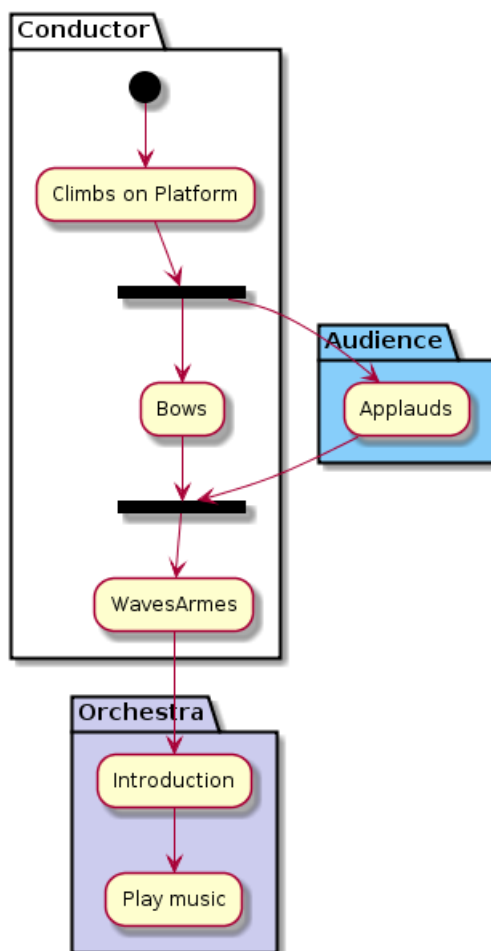
partition Audience LightSkyBlue {
=== S1 === --> Applauds
}

partition Conductor {
Bows --> === S2 ===
--> WavesArmes
Applauds --> === S2 ===
}

partition Orchestra #CCCCEE {
WavesArmes --> Introduction
--> "Play music"
}

@enduml

```



4.10 Mettre un titre

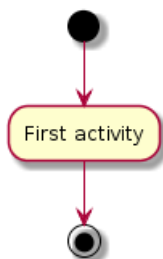
Le mot-clé `title` est utilisé pour mettre un titre.

Vous pouvez utiliser les mot-clés `title` et `end title` pour des titres plus long, comme dans les diagrammes de séquence.

```

@startuml
title Simple example\nof title
(*) --> "First activity"
--> (*)
@enduml
  
```

Simple example
of title



4.11 Paramètre de thème

Vous pouvez utiliser la commande `skinparam` pour changer la couleur et la police d'écriture pour dessiner.

Vous pouvez utiliser cette commande :

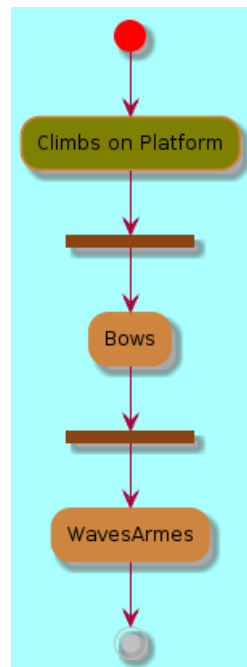
- Dans le diagramme de définition, comme n'importe quelle autre commande,
- Dans un fichier inclus,
- Dans un fichier de configuration, à condition que la ligne de commande ou la tâche ANT.

Vous pouvez spécifier une couleur et une police d'écriture dans les stéréotypes d'activités.

```
@startuml
skinparam backgroundColor #AAFFFF
skinparam activity {
  StartColor red
  BarColor SaddleBrown
  EndColor Silver
  BackgroundColor Peru
  BackgroundColor<< Begin >> Olive
  BorderColor Peru
  FontName Impact
}

(*) --> "Climbs on Platform" << Begin >>
--> == S1 ==
--> Bows
--> == S2 ==
--> WavesArmes
--> (*)

@enduml
```



4.12 Octogone

Vous pouvez changer la forme des activités en octogone en utilisant la commande `skinparam activityShape octagon`

```
@startuml
'Default is skinparam activityShape roundBox
skinparam activityShape octagon
```



```
(*) --> "First Activity"
"First Activity" --> (*)

@enduml
```



4.13 Exemple complet

```
@startuml
title Servlet Container

(*) --> "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

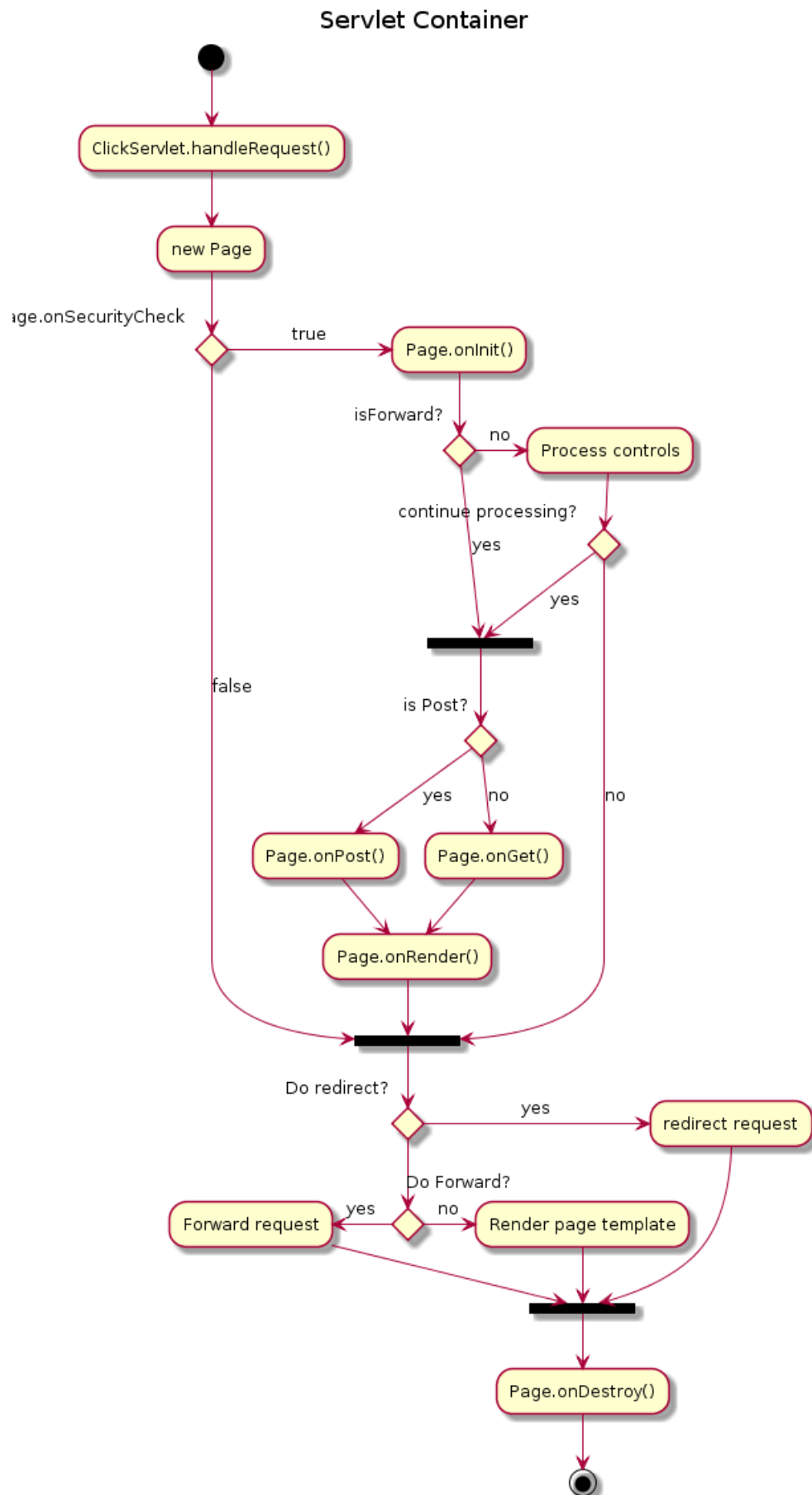
else
-->[false] ===REDIRECT_CHECK===
endif

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 d'activité (bêta)

La syntaxe courante pour les diagrammes d'activité a plusieurs limitations et inconvénients (par exemple, c'est difficile à maintenir).

Une complète nouvelle syntaxe et implémentation est proposé avec **beta version** aux utilisateurs (commence avec V7947), ainsi cela permet de définir un meilleur nouveau format et syntaxe.

Un autre avantage de cette nouvelle implémentation est qu'il n'y a pas besoin d'avoir Graphviz d'installé (comme pour les diagrammes de séquences).

La nouvelle syntaxe remplace l'ancienne. Cependant, pour des raisons de compatibilité, l'ancienne syntaxe reste reconnu, pour s'en assurer *ascending compatibility*.

Les utilisateurs sont simplement encouragés à migrer vers une nouvelle syntaxe.

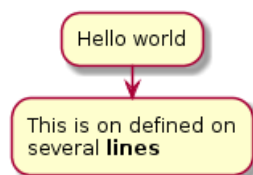
5.1 Activité simple

Les étiquettes d'activités commencent avec `:` et finissent avec `;`.

Le formatage de texte peut être fait en utilisant la syntaxe creole wiki.

Ils sont implicitement liés à leur ordre de définition.

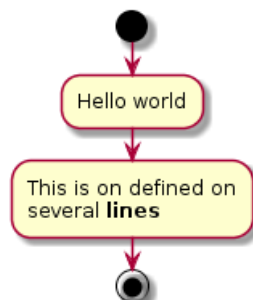
```
@startuml
:Hello world;
:This is on defined on
several lines;
@enduml
```



5.2 Départ/Arrêt

Vous pouvez utiliser les mots clés `start` et `stop` pour indiquer le début et la fin du diagramme.

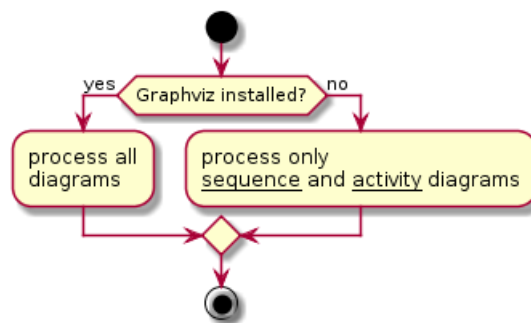
```
@startuml
start
:Hello world;
:This is on defined on
several lines;
stop
@enduml
```



5.3 Conditionnel

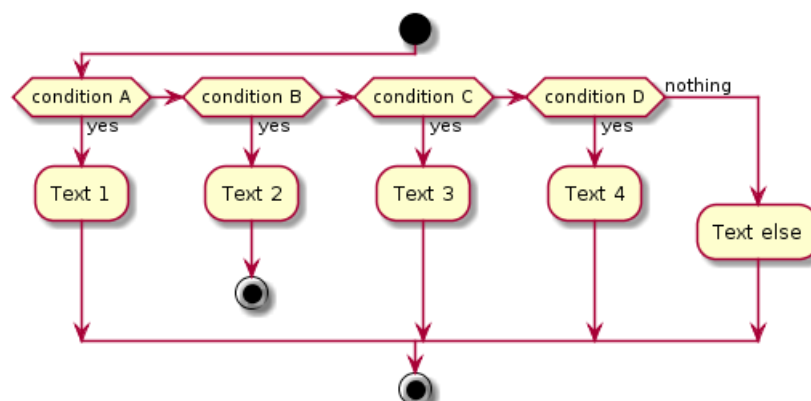
Vous pouvez utiliser les mots clés **if**, **then** et **else** pour mettre des tests si votre diagramme. Les étiquettes peuvent être fournies entre parenthèse.

```
@startuml
start
if (Graphviz installed?) then (yes)
:process all\ndiagrams;
else (no)
:process only
__sequence__ and __activity__ diagrams;
endif
stop
@enduml
```



Vous pouvez utiliser le mot clé **elseif** pour avoir plusieurs tests :

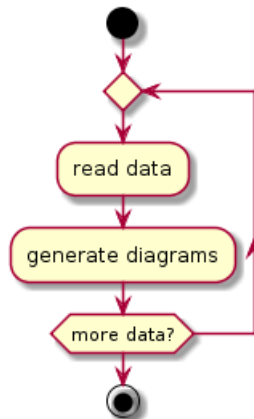
```
@startuml
start
if (condition A) then (yes)
:Text 1;
elseif (condition B) then (yes)
:Text 2;
stop
elseif (condition C) then (yes)
:Text 3;
elseif (condition D) then (yes)
:Text 4;
else (nothing)
:Text else;
endif
stop
@enduml
```



5.4 Boucle de répétition

Vous pouvez utiliser les mots clés `repeat` et `repeatwhile` pour créer une boucle.

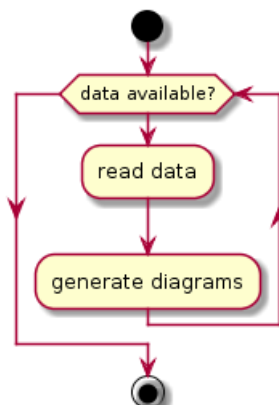
```
@startuml
start
repeat
:read data;
:generate diagrams;
repeat while (more data?)
stop
@enduml
```



5.5 Boucle While

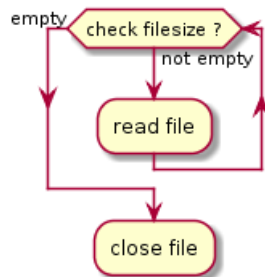
Vous pouvez utiliser les mots clés `while` et `end while` pour définir une boucle.

```
@startuml
start
while (data available?)
:read data;
:generate diagrams;
endwhile
stop
@enduml
```



Il est possible de mettre un libellé après le mot clé **endwhile** ou bien avec le mot clé **is**.

```
@startuml
while (check filesize ?) is (not empty)
:read file;
endwhile (empty)
:close file;
@enduml
```



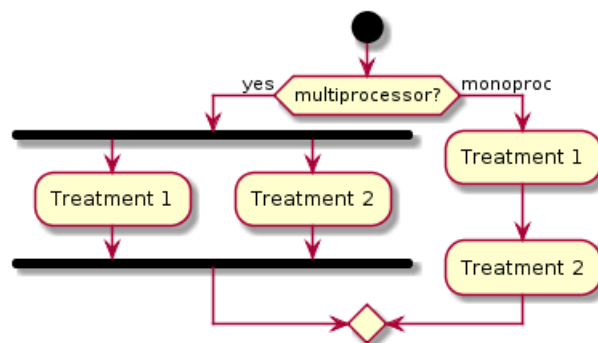
5.6 Processus parallèle

Vous pouvez utiliser les mots clés **fork**, **fork again** et **end fork** pour indiquer un processus parallèle.

```
@startuml
start

if (multiprocessor?) then (yes)
fork
:Treatment 1;
fork again
:Treatment 2;
end fork
else (monoproc)
:Treatment 1;
:Treatment 2;
endif

@enduml
```



5.7 Notes

Le formatage de texte peut être fait en utilisant la syntaxe créole wiki.

```
@startuml
start
:foo1;
```

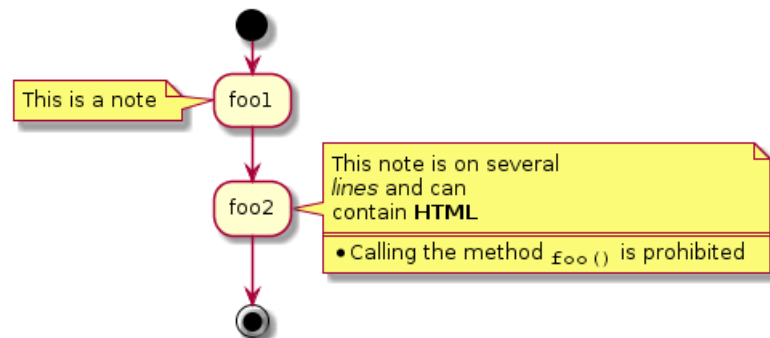


```

note left: This is a note
:foo2;
note right
This note is on several
//lines// and can
contain <b>HTML</b>
====
* Calling the method "foo()" is prohibited
end note
stop

@enduml

```



5.8 Titre et légende

Vous pouvez ajouter un titre, une en-tête, un pied-de-page, une légende à un diagramme.

```

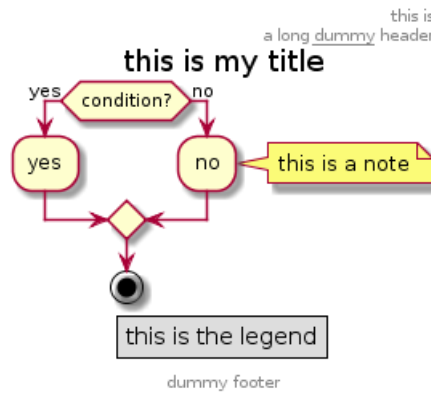
@startuml
title this is my title
if (condition?) then (yes)
:yes;
else (no)
:no;
note right
this is a note
end note
endif
stop

legend
this is the legend
endlegend

footer dummy footer
header
this is
a long __dummy__ header
end header

@enduml

```



5.9 Couleurs

Vous pouvez spécifier une couleur pour certaines activités.

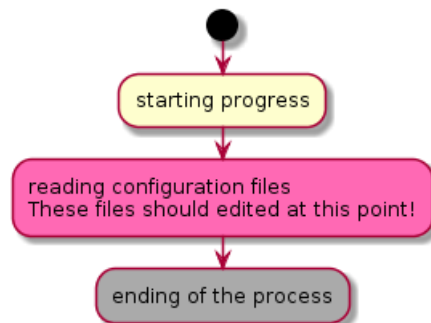
```
@startuml
```

```

start
:starting progress;
#HotPink:reading configuration files
These files should edited at this point!;
#AAAAAA:ending of the process;

```

```
@enduml
```



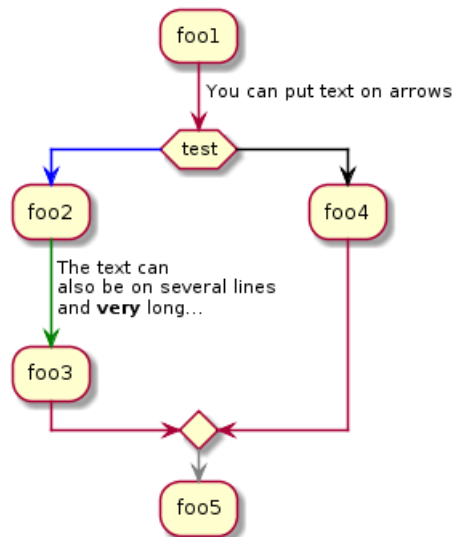
5.10 flèches

Utiliser la notation `->`, vous pouvez ajouter le texte à la flèche, et changer leur couleur.

```

@startuml
:foo1;
-> You can put text on arrows;
if (test) then
-[#blue]->
:foo2;
-[#green]-> The text can
also be on several lines
and very long...;
:foo3;
else
-[#black]->
:foo4;
endif
-[#gray]->
:foo5;
@enduml

```



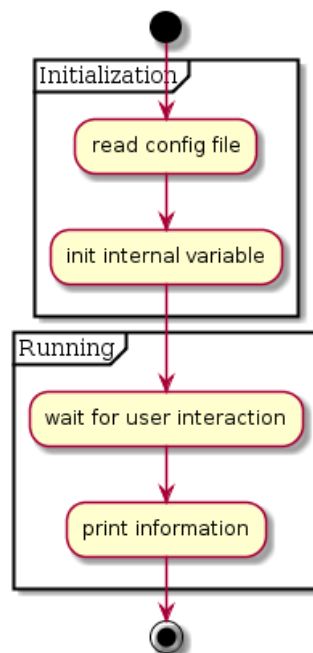
5.11 Groupement

Vous pouvez grouper les activités ensemble en définissant les partitions.

```

@startuml
start
partition Initialization {
:read config file;
:init internal variable;
}
partition Running {
:wait for user interaction;
:print information;
}

stop
@enduml
  
```

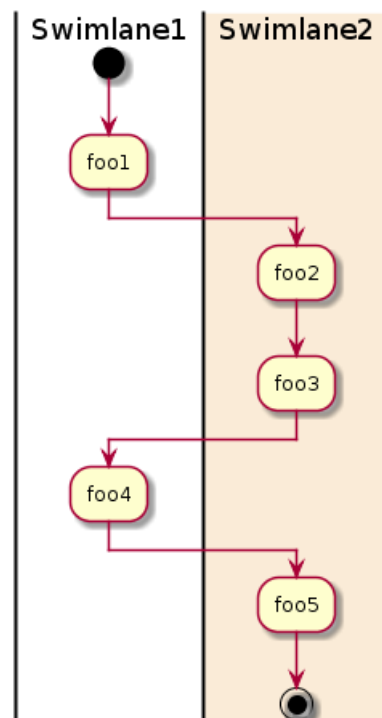


5.12 Couloirs

A l'aide du symbole `|`, il est possible de définir des couloirs d'exécution.

Il est aussi possible de changer la couleur d'un couloir.

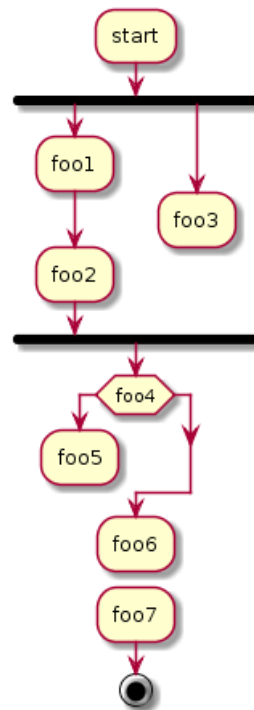
```
@startuml
|Swimlane1|
start
:foo1;
|#AntiqueWhite|Swimlane2|
:foo2;
:foo3;
|Swimlane1|
:foo4;
|Swimlane2|
:foo5;
stop
@enduml
```



5.13 Détacher

Il est possible de supprimer un utilisant le mot clé `detach`.

```
@startuml
:start;
fork
:foo1;
:foo2;
fork again
:foo3;
detach
endfork
if (foo4) then
:foo5;
detach
endif
:foo6;
detach
:foo7;
stop
@enduml
```



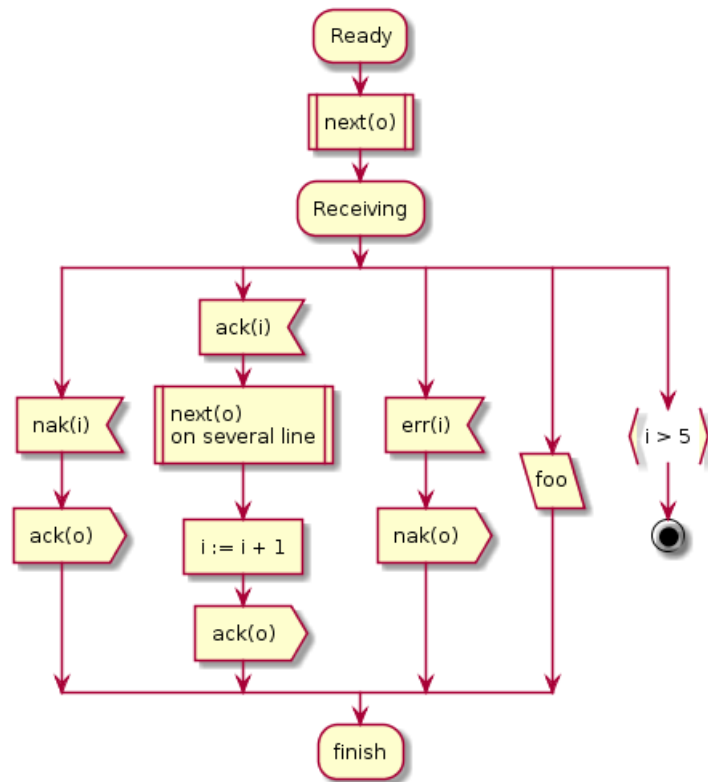
5.14 SDL

En changeant le séparateur final ;, vous pouvez déterminer différents rendu pour l'activité

- |
- <
- >
- /
-]
- }

```

@startuml
:Ready;
:next(o)|
:Receiving;
split
:nak(i)<
:ack(o)>
split again
:ack(i)<
:next(o)
on several line|
:i := i + 1]
:ack(o)>
split again
:err(i)<
:nak(o)>
split again
:foo/
split again
:i > 5}
stop
end split
:finish;
@enduml
  
```

5.15 Exemple complet

```

@startuml

start
:ClickServlet.handleRequest();
:new page;
if (Page.onSecurityCheck) then (true)
:Page.onInit();
if (isForward?) then (no)
:Process controls;
if (continue processing?) then (no)
stop
endif
endif

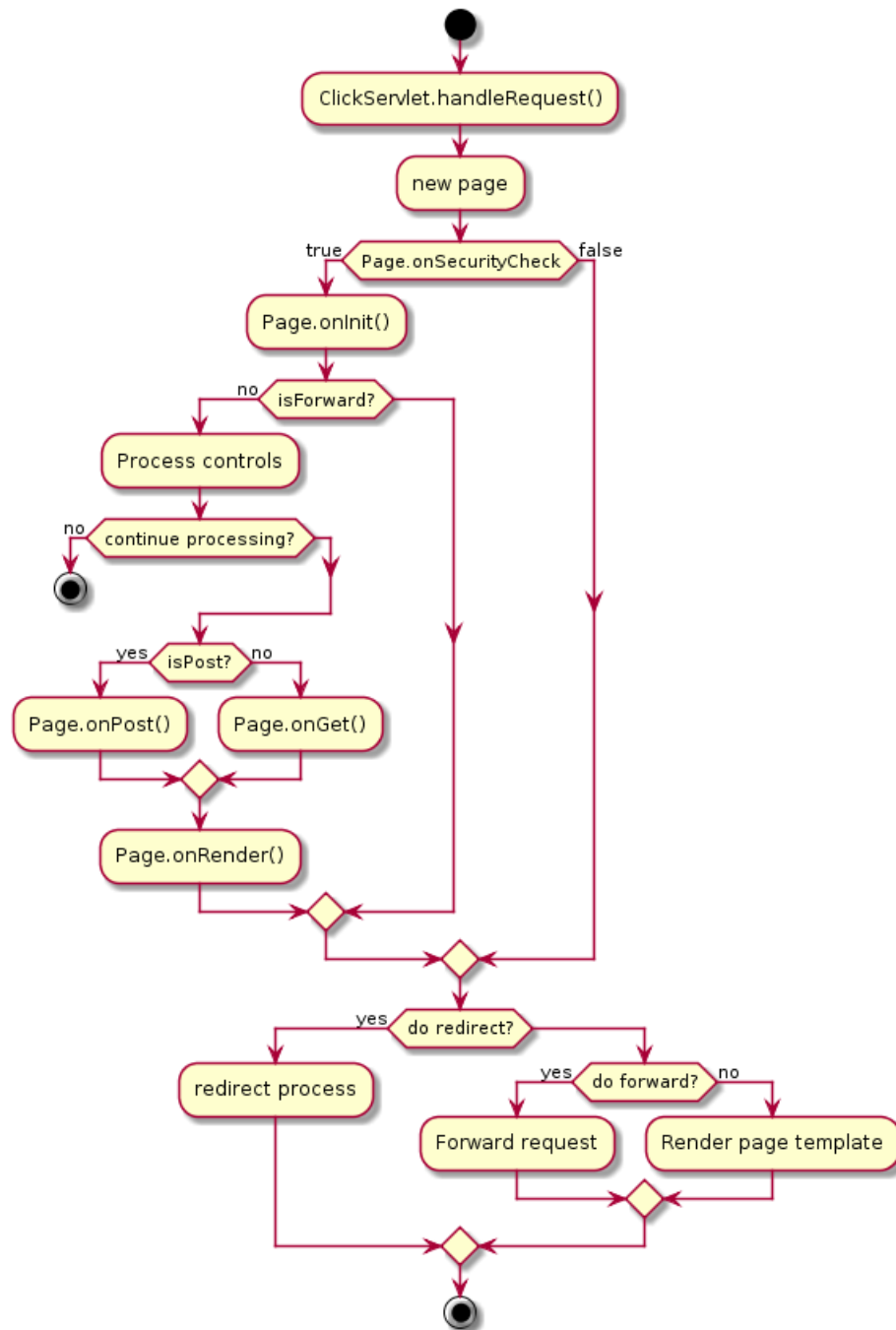
if (isPost?) then (yes)
:Page.onPost();
else (no)
:Page.onGet();
endif
:Page.onRender();
endif
else (false)
endif

if (do redirect?) then (yes)
:redirect process;
else
if (do forward?) then (yes)
:Forward request;
else (no)
:Render page template;
endif
endif

stop

@enduml

```



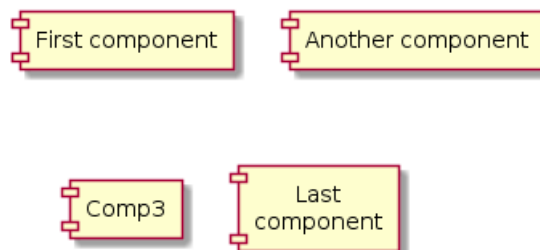
6 Diagrammes de composants

6.1 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
```



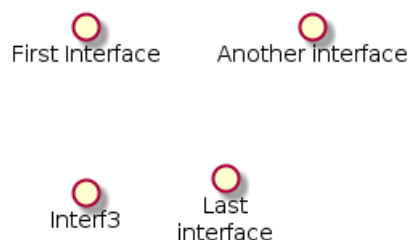
6.2 Interfaces

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. Vous pouvez aussi définir un alias, à l'aide du mot-clé **as**. Cet alias pourrait être utilisé plus tard, lors de la définition des relations.

Nous verrons plus tard qu'il n'est pas obligatoire de définir les interfaces.

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



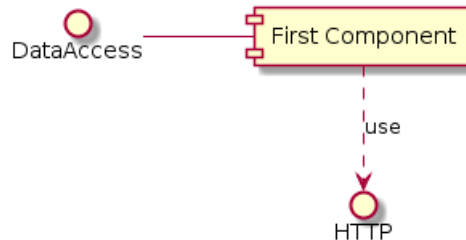
6.3 Exemple simple

Les liens entre les éléments sont à utiliser avec des combinaisons de lignes pointillées (.), lignes droites(--), et de flèches (-->).

```

@startuml
DataAccess - [First Component]
[First Component] ..> HTTP : use
@enduml

```



6.4 Mettre des notes

Vous pouvez utiliser les commandes suivantes : `note left of` , `note right of` , `note top of` , `note bottom of` keywords to define notes related to a single object.

Une note peut aussi être e alone with the `note` keywords, then linked to other objects using the `..` symbol.

```

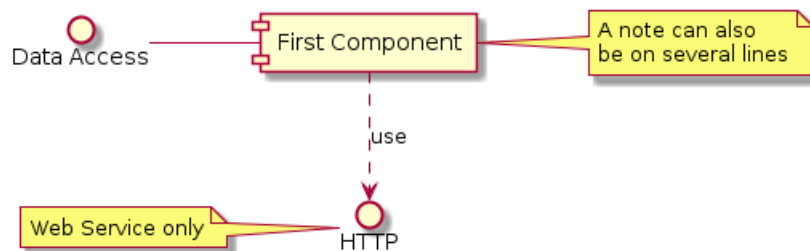
@startuml
interface "Data Access" as DA

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

```



6.5 Regrouper des composants

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

- `package`
- `node`
- `folder`
- `frame`
- `cloud`
- `database`



```

@startuml

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

node "Other Groups" {
  FTP - [Second Component]
  [First Component] --> FTP
}

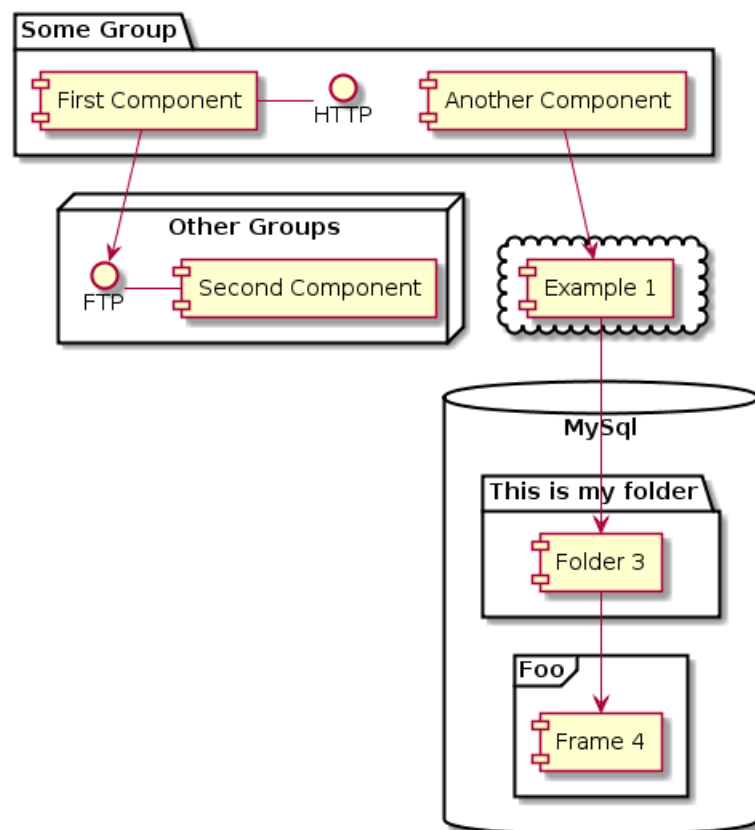
cloud {
  [Example 1]
}

database "MySQL" {
  folder "This is my folder" {
    [Folder 3]
  }
  frame "Foo" {
    [Frame 4]
  }
}

[Another Component] --> [Example 1]
[Example 1] --> [Folder 3]
[Folder 3] --> [Frame 4]

@enduml

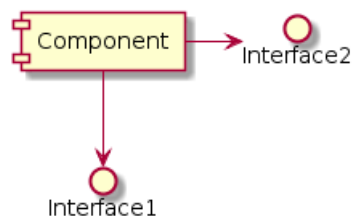
```



6.6 Changer la direction des flèches

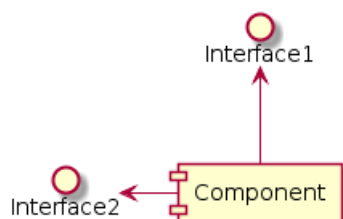
Par défaut, les liens entre classes ont deux tirets -- et sont orientées verticalement. C'est possible d'utiliser horizontalement un lien en mettant un simple tiret (ou point) comme ceci :

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



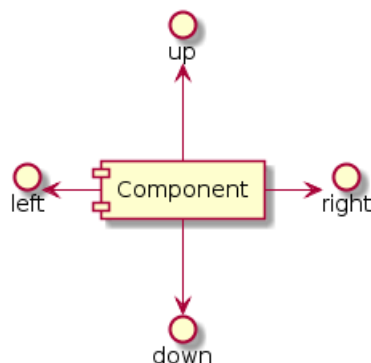
Vous pouvez aussi changer le sens en renversant le lien

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



Il est aussi possible de changer la direction des flèches e, ajoutant les mots clés **left**, **right**, **up** ou **down** à l'intérieur des flèches :

```
@startuml
[Component] -left-> left
[Component] -right-> right
[Component] -up-> up
[Component] -down-> down
@enduml
```



Vous pouvez raccourcir les flèches en utilisant seulement les premiers caractères de la direction (par exemple, **-d-** instead of **-down-**) ou les deux premiers caractères (**-do-**).

Veuillez noter qu'il ne faut pas abuser de cette fonctionnalité : *Graphviz* donne généralement de bon résultat sans modification.



6.7 Ajouter un titre

Le mot-clé `title` est utilisé pour rajouter un titre.

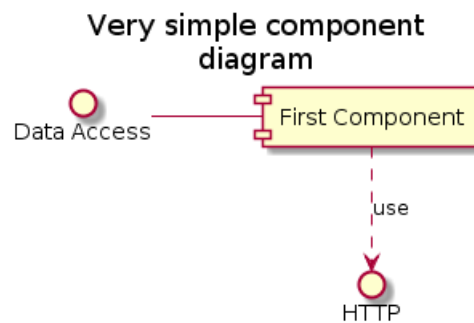
Vous pouvez aussi utiliser `title` et `end title` pour avoir un titre plus long, comme dans les diagrammes de séquence.

```
@startuml
title Very simple component\ndiagram

interface "Data Access" as DA

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

@enduml
```



6.8 Utiliser la notation UML2

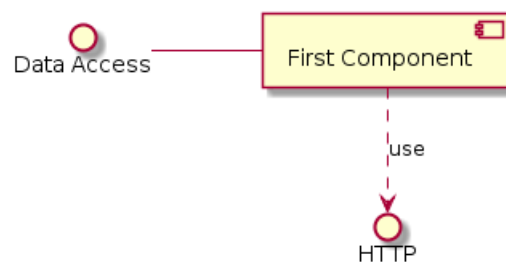
La commande `skinparam componentStyle uml2` est utilisée pour changer vers la notation UML2.

```
@startuml
skinparam componentStyle uml2

interface "Data Access" as DA

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

@enduml
```



6.9 Couleurs individuelles

Il est possible de spécifier une couleur après la définition du composant.

```
@startuml
component [Web Server] #Yellow

@enduml
```



6.10 Skinparam

Vous pouvez utiliser la commande `skinparam` pour changer les couleurs et les polices du dessin.

Vous pouvez utiliser cette commande :

- Dans la définition du diagramme, comme n'importe qu'elle autre commandes.
- Dans un fichier inclus,
- Dans le fichier de configuration, fournit par la ligne de commande ou la tâche ANT.

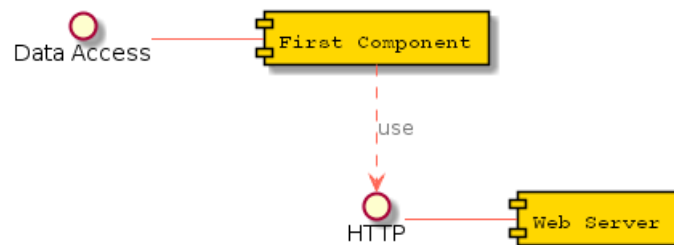
Vous pouvez définir des couleurs et des fontes spécifiques pour les composants et interfaces stéréotypés.

```
@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
```



```
@startuml
[AA] <<static lib>>
[BB] <<shared lib>>
[CC] <<static lib>>

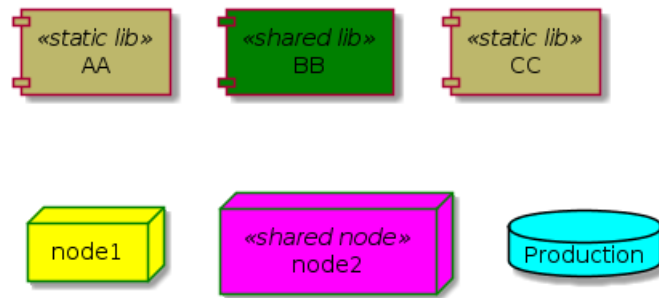
node node1
node node2 <<shared node>>
database Production

skinparam component {
backgroundColor<<static lib>> DarkKhaki
backgroundColor<<shared lib>> Green
}

skinparam node {
borderColor Green
backgroundColor Yellow
backgroundColor<<shared node>> Magenta
}
skinparam databaseBackgroundColor Aqua

@enduml
```





7 Diagrammes d'état

7.1 Exemple simple

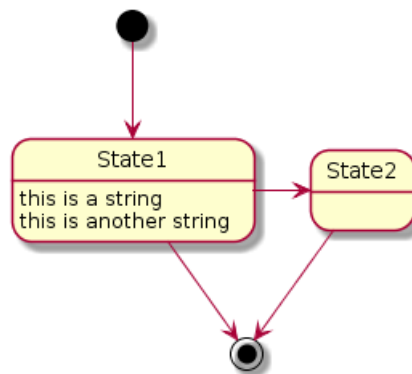
Vous devez utiliser [*] pour le début et la fin du diagramme d'états.

Utilisez --> pour les flèches.

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

State1 -> State2
State2 --> [*]

@enduml
```



7.2 Etat composite

Un état peut également être composite. Vous devez alors le définir avec le mot-clé **state** et des accolades.

```
@startuml
scale 350 width
[*] --> NotShooting

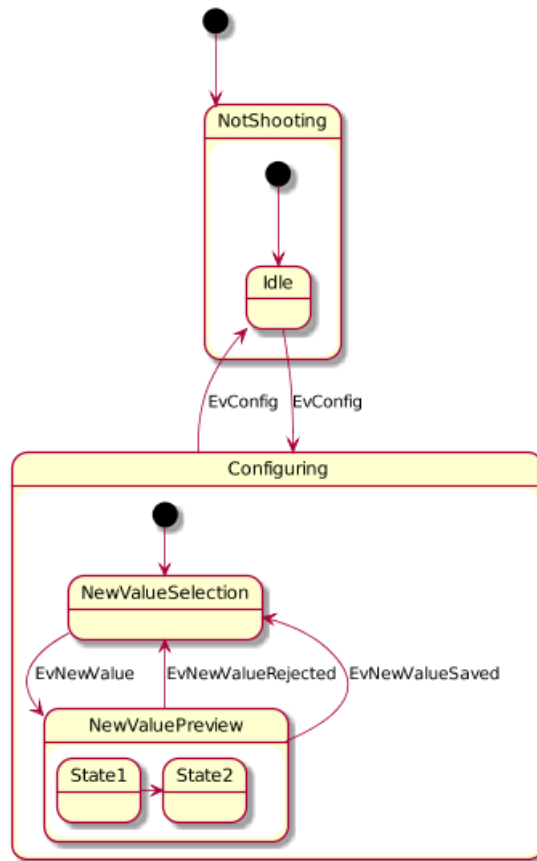
state NotShooting {
    [*] --> Idle
    Idle --> Configuring : EvConfig
    Configuring --> Idle : EvConfig
}

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

state NewValuePreview {
    State1 -> State2
}

}

@enduml
```



7.3 Nom long

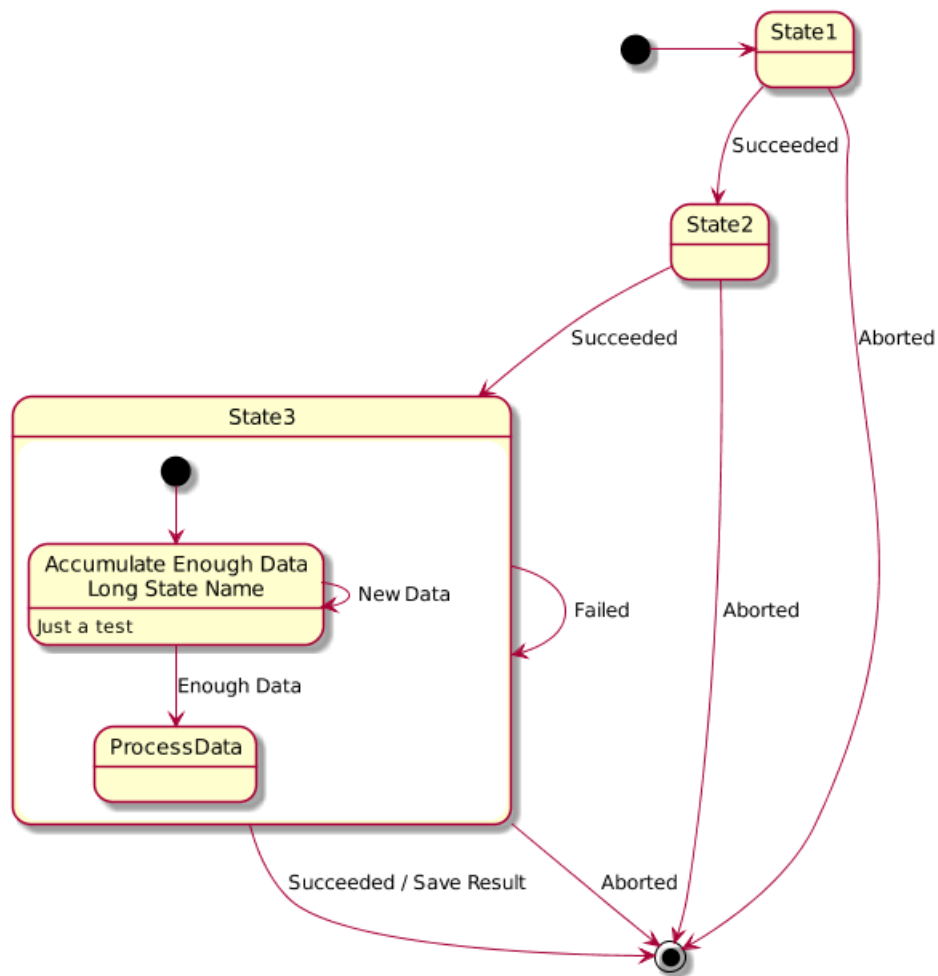
Vous pouvez aussi utiliser le mot-clé **state** pour donner un nom avec des espaces à un état.

```

@startuml
scale 600 width

[*] -> 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
  
```



7.4 Etat concurrent

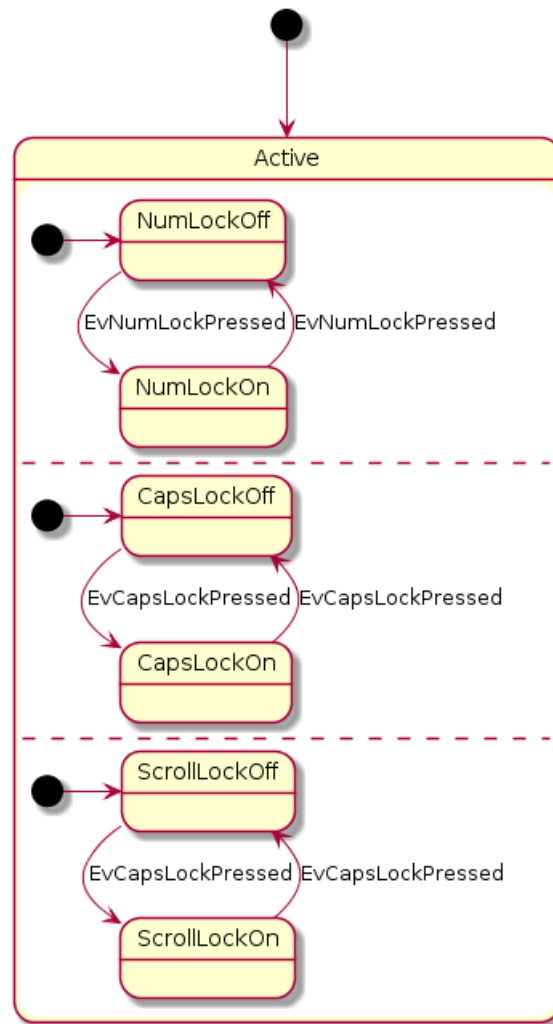
Vous pouvez définir un état concurrent dans un état composé en utilisant le symbole `--` comme séparateur.

```

@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
  
```



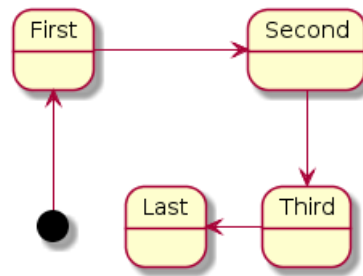
7.5 Direction des flèches

Vous pouvez utiliser `->` pour les flèches horizontales. Il est aussi possible de forcer la direction de la flèche avec la syntaxe suivante:

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

```

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



Vous pouvez aussi utiliser une notation abrégée, avec soit le premier caractère de la direction (par exemple `-d-` à la place de `-down-`) ou bien les deux premiers caractères (`-do-`).

Veillez noter qu'il ne faut pas abuser de cette fonction : *Graphviz* donne généralement de bons résultats sans peaufinage.

7.6 Note

Vous pouvez définir des notes avec : `note left of`, `note right of`, `note top of`, `note bottom of` Mots clés.

Vous pouvez aussi définir des notes sur plusieurs lignes.

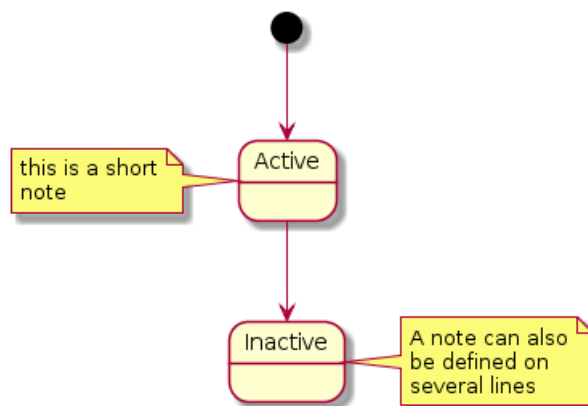
```

@startuml
[*] --> Active
Active --> Inactive

note left of Active : this is a short\nnote

note right of Inactive
A note can also
be defined on
several lines
end note

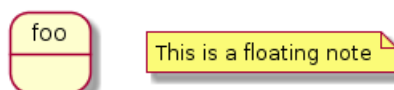
@enduml
  
```



Vous pouvez aussi avoir des notes flottantes.

```

@startuml
state foo
note "This is a floating note" as N1
@enduml
  
```



7.7 Plus de notes

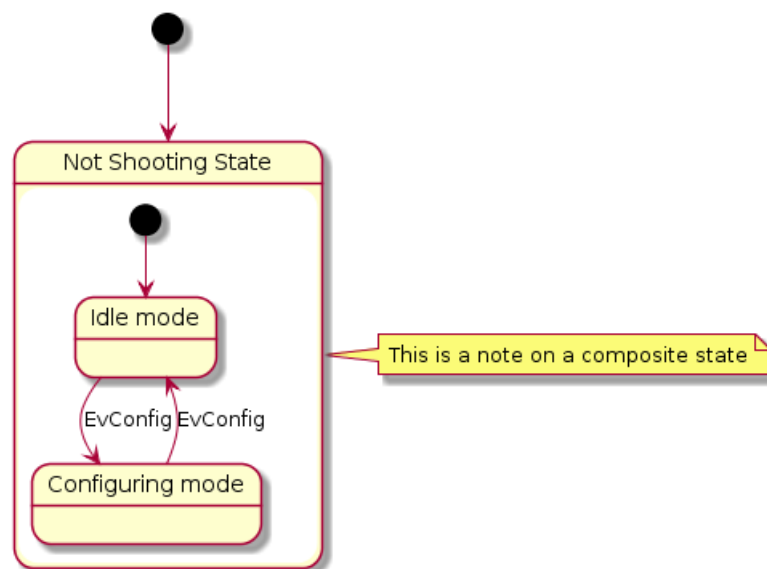
Vous pouvez mettre des notes sur les états de composite

```
@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
```



7.8 Skinparam

Vous pouvez utiliser la commande `skinparam` pour changer les couleurs et les polices pour le dessin.

Vous pouvez utiliser cette commande :

- Dans la définition du diagramme, comme n'importe quelle autre commande,
- Dans un fichier inclus,
- Dans un fichier de configuration, à condition que dans la ligne de commande ou la tâche ANT.

Vous pouvez définir une couleur spécifique et une police d'écriture pour les états stéréotypés.

```
@startuml
skinparam backgroundColor LightYellow
skinparam state {
StartColor MediumBlue
EndColor Red
BackgroundColor Peru
BackgroundColor<<Warning>> Olive
BorderColor Gray
FontName Impact
}

[*] --> NotShooting

state "Not Shooting State" as NotShooting {
```

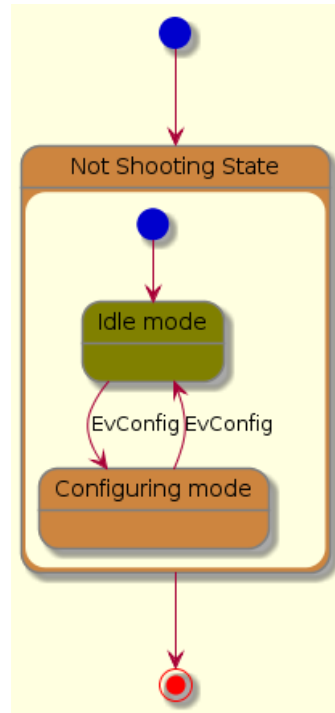


```

state "Idle mode" as Idle <<Warning>>
state "Configuring mode" as Configuring
[*] --> Idle
Idle --> Configuring : EvConfig
Configuring --> Idle : EvConfig
}

NotShooting --> [*]
@enduml

```

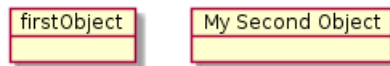


8 Diagrammes d'objets

8.1 Définition des objets

Les instances d'objets sont définies avec le mot clé `object`.

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



8.2 Relations entre les objets

Les relations entre objets sont définies à l'aide des symboles suivants :

Héritage	< --	
Composition	*--	
Agrégation	o--	

Il est possible de remplacer `--` par `..` pour avoir des pointillés.

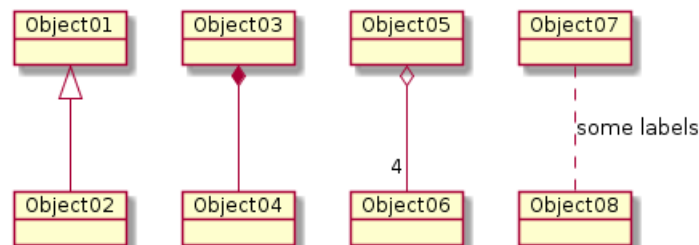
Grâce à ces règles, on peut avoir les dessins suivants:

Il est possible d'ajouter une étiquette sur la relation, en utilisant `" : "`, suivi par le texte de l'étiquette.

Pour les cardinalités, vous pouvez utiliser les doubles quotes `"` sur chaque côté de la relation.

```
@startuml
object Object01
object Object02
object Object03
object Object04
object Object05
object Object06
object Object07
object Object08

Object01 <|-- Object02
Object03 *-- Object04
Object05 o-- "4" Object06
Object07 .. Object08 : some labels
@enduml
```

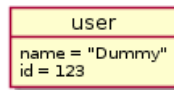


8.3 Ajout de champs

Pour déclarer un champ, vous pouvez utiliser le symbole `:"` suivi par le nom du champs.

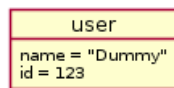
```
@startuml
object user
```

```
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
```



8.4 Caractéristiques communes avec les diagrammes de classes

- Visibilité
- Ajout de notes
- Utilisation de packages
- Titre de diagramme
- Personnalisation de l'affichage
- Découpage de l'image

9 Commandes communes

9.1 Entête et pied de page

Vous pouvez utiliser les commandes **header** ou **footer** pour ajouter un entête ou pied de page sur un diagramme généré.

Vous pouvez éventuellement spécifier la position de l'élément en utilisant les mots clés **center**, **left** ou **right**.

Comme pour le titre, il est possible de définir un entête ou pied de page sur plusieurs lignes.

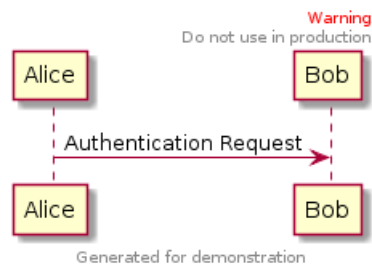
Il est aussi possible de mettre du HTML dans l'entête ou dans le pied de page

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

header
<font color=red>Warning:</font>
Do not use in production.
endheader

center footer Generated for demonstration

@enduml
```



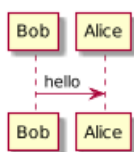
9.2 Zoom

Vous pouvez utiliser la commande **scale** pour zoomer l'image générée.

Vous pouvez utiliser un nombre ou une fraction pour définir le facteur d'échelle. Vous pouvez aussi spécifier la largeur et la hauteur (en pixel). Et vous pouvez aussi donner la largeur et la hauteur : l'image est mis à l'échelle pour s'adapter aux dimensions spécifiées.

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

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



10 Modifier les polices et couleurs de caractères

10.1 Usage

Vous pouvez changer les couleurs et la police avec la commande `skinparam`. Exemple:

```
skinparam backgroundColor yellow
```

Vous pouvez utiliser cette commande :

- Dans la définition du diagramme, comme les autres commandes,
- Dans un fichier inclus (voir *Preprocessing*),
- Dans un fichier de configuration, donné dans la ligne de commande ou dans la tâche ANT.

10.2 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
```

est absolument équivalent à :

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

10.3 Couleur

Vous pouvez utiliser au choix le nom d'une couleur ou son code RVB.

Nom du paramètre	Défaut Valeur	Couleur	Commentaire
backgroundColor	white		Fond de la page
activityArrowColor	#A80036		Couleur des flèches des diagrammes d'activité
activityBackgroundColor	#FEFECE		Couleur du fond des diagrammes d'activité
activityBorderColor	#A80036		Couleur des bordures des diagrammes d'activité
activityStartColor	black		Etat (cercle) de départ dans les diagrammes d'activité
activityEndColor	black		Etat (cercle) de fin dans les diagrammes d'activité
activityBarColor	black		Barre de synchronisation dans les diagrammes d'activité
usecaseArrowColor	#A80036		Couleurs des flèches dans les diagrammes de cas d'utilisation
usecaseActorBackgroundColor	#FEFECE		Couleurs des acteurs dans les diagrammes de cas d'utilisation
usecaseActorBorderColor	#A80036		Couleur de la bordure d'un acteur dans un diagramme de cas d'utilisation
usecaseBackgroundColor	#FEFECE		Couleur de fond des cas d'utilisation
usecaseBorderColor	#A80036		Couleur de la bordure d'un cas d'utilisation dans un diagramme
classArrowColor	#A80036		Couleurs des flèches dans les diagrammes de classe
classBackgroundColor	#FEFECE		Couleur de fond des classes dans les diagrammes de classe
classBorderColor	#A80036		Bordure des classes/interface/enum dans les diagrammes de classe
packageBackgroundColor	#FEFECE		Couleur de fond des packages dans les diagrammes de classe
packageBorderColor	#A80036		Bordures des packages dans les diagrammes de classe
stereotypeCBackgroundColor	#ADD1B2		Couleur de fond des points des classes dans les diagrammes de classe
stereotypeABBackgroundColor	#A9DCDF		Couleur de fond des points des classes abstraites dans les diagrammes de classe
stereotypeIBBackgroundColor	#B4A7E5		Couleur de fond des points des interfaces dans les diagrammes de classe
stereotypeEBackgroundColor	#EB937F		Couleur de fond des points des enum dans les diagrammes de classe
componentArrowColor	#A80036		Couleur des flèches dans les diagrammes de composants
componentBackgroundColor	#FEFECE		Couleur de fond des composants
componentBorderColor	#A80036		Bordure des composants
componentInterfaceBackgroundColor	#FEFECE		Couleur de fond des interfaces dans les diagrammes de composants
componentInterfaceBorderColor	#A80036		Bordure des interfaces dans les diagrammes de composants
noteBackgroundColor	#FBFB77		Couleur de fond des notes
noteBorderColor	#A80036		Bordure des notes
stateBackgroundColor	#FEFECE		Fond d'état d'un diagramme d'état
stateBorderColor	#A80036		Bordure d'état d'un diagramme d'état
stateArrowColor	#A80036		Couleurs de flèches dans les diagrammes d'état
stateStartColor	black		Etat démarrage dans les diagrammes d'état
stateEndColor	black		Fin d'un cercle dans les diagrammes d'état
sequenceArrowColor	#A80036		Couleurs des flèches dans les diagrammes de séquence
sequenceActorBackgroundColor	#FEFECE		Couleur de la tête des acteurs dans les diagrammes de séquence
sequenceActorBorderColor	#A80036		Contour des acteurs dans les diagrammes de séquence
sequenceGroupBackgroundColor	#EEEEEE		Couleur de la tête de page de alt/opt/loop dans un diagramme de séquence
sequenceLifeLineBackgroundColor	white		Fond d'une ligne de vie dans des diagrammes de séquence
sequenceLifeLineBorderColor	#A80036		Bordure de ligne de vie dans un diagramme de séquence
sequenceParticipantBackgroundColor	#FEFECE		Couleur de fond des participants dans les diagrammes de séquence
sequenceParticipantBorderColor	#A80036		Bordure des participants dans les diagrammes de séquence



10.4 Couleur de police, nom et taille

Vous pouvez changer la police d'écriture en utilisant `xxxFontColor`, paramètres `xxxFontSize` et `xxxFontName`.

Exemple:

```
skinparam classFontColor red
skinparam classFontSize 10
skinparam classFontName Apex
```

Vous pouvez aussi changer la police par défaut pour toutes les polices utilisées `skinparam defaultFontName`.

Exemple :

```
skinparam defaultFontName Apex
```

Veuillez noter que le nom de la police est hautement dépendant, alors ne l'utilisez pas trop, si c'est ce que vous recherchez portabilité.

Paramètre Nom	Défaut Valeur	Commentaire
activityFontColor activityFontSize activityFontStyle activityFontName	black 14 plain	Utilisé pour une boîte d'activité
activityArrowFontColor activityArrowFontSize activityArrowFontStyle activityArrowFontName	black 13 plain	Utilisé pour le texte et les flèches dans les diagrammes d'activité.
circledCharacterFontColor circledCharacterFontSize circledCharacterFontStyle circledCharacterFontName circledCharacterRadius	black 17 bold Courier 11	Utilisé pour le texte en cercle pour les classes, énumérations et autres.
classArrowFontColor classArrowFontSize classArrowFontStyle classArrowFontName	black 10 plain	Utilisé pour le texte sur les flèches dans les diagrammes de classe.
classAttributeFontColor classAttributeFontSize classAttributeIconSize classAttributeFontStyle classAttributeFontName	black 10 10 plain	Attribut et méthode de classes
classFontColor classFontSize classFontStyle classFontName	black 12 plain	Utilisé pour les noms de classe
classStereotypeFontColor classStereotypeFontSize classStereotypeFontStyle classStereotypeFontName	black 12 italic	Utilisé pour les stéréotypes de classes
componentFontColor componentFontSize componentFontStyle componentFontName	black 14 plain	Utilisé pour les noms de composant
componentStereotypeFontColor componentStereotypeFontSize componentStereotypeFontStyle componentStereotypeFontName	black 14 italic	Utilisé pour stéréotypes de composant

componentArrowFontColor componentArrowFontSize componentArrowFontStyle componentArrowFontName	black 13 plain	Utilisé pour du texte sur les flèches dans un diagramme de com
noteFontColor noteFontSize noteFontStyle noteFontName	black 13 plain	Utilisé pour des notes dans tous les diagrammes mais le diagram
packageFontColor packageFontSize packageFontStyle packageFontName	black 14 plain	Utilisé pour le package les noms de partition
sequenceActorFontColor sequenceActorFontSize sequenceActorFontStyle sequenceActorFontName	black 13 plain	Utilisé pour les acteurs dans le diagramme de séquence
sequenceDividerFontColor sequenceDividerFontSize sequenceDividerFontStyle sequenceDividerFontName	black 13 bold	Utilisé pour le texte sur les séparateurs dans les diagrammes de séq
sequenceArrowFontColor sequenceArrowFontSize sequenceArrowFontStyle sequenceArrowFontName	black 13 plain	Texte sur les flèches dans les diagrammes de séquence
sequenceGroupingFontColor sequenceGroupingFontSize sequenceGroupingFontStyle sequenceGroupingFontName	black 11 plain	Utilisé pour le texte pour "else" dans les diagrammes de séquence.
sequenceGroupingHeaderFontColor sequenceGroupingHeaderFontSize sequenceGroupingHeaderFontStyle sequenceGroupingHeaderFontName	black 13 plain	Utilisé pour le texte pour "alt/opt/loop" dans les diagrammes de s
sequenceParticipantFontColor sequenceParticipantFontSize sequenceParticipantFontStyle sequenceParticipantFontName	black 13 plain	Utilisé pour le textetur les participants dans les diagrammes de séq
sequenceTitleFontColor sequenceTitleFontSize sequenceTitleFontStyle sequenceTitleFontName	black 13 plain	Utilisé pour les titres dans les diagrammes de séquence
titleFontColor titleFontSize titleFontStyle titleFontName	black 18 plain	Utilisé pour les titres de tous les diagrammes sauf diagrammes de s
stateFontColor stateFontSize stateFontStyle stateFontName	black 14 plain	Utilisé pour les états dans les diagrammes d'état.
stateArrowFontColor stateArrowFontSize stateArrowFontStyle stateArrowFontName	black 13 plain	Utilisé pour le texte sur les flèches dans les diagrammes d'état.
stateAttributeFontColor stateAttributeFontSize stateAttributeFontStyle stateAttributeFontName	black 12 plain	Utilisé pour les descriptions d'états dans les diagrammes d'état.

usecaseFontColor usecaseFontSize usecaseFontStyle usecaseFontName	black 14 plain	Utilisé pour les labels de cas d'utilisation dans les diagrammes de cas d'utilisation
usecaseStereotypeFontColor usecaseStereotypeFontSize usecaseStereotypeFontStyle usecaseStereotypeFontName	black 14 italic	Utilisé pour les stéréotypes dans les cas d'utilisation
usecaseActorFontColor usecaseActorFontSize usecaseActorFontStyle usecaseActorFontName	black 14 plain	Utilisé pour les labels d'acteurs dans les diagrammes de cas d'utilisation
usecaseActorStereotypeFontColor usecaseActorStereotypeFontSize usecaseActorStereotypeFontStyle usecaseActorStereotypeFontName	black 14 italic	Utilisé pour les stéréotypes d'acteurs
usecaseArrowFontColor usecaseArrowFontSize usecaseArrowFontStyle usecaseArrowFontName	black 13 plain	Texte des flèches dans les diagrammes de cas d'utilisation
footerFontColor footerFontSize footerFontStyle footerFontName	black 10 plain	Utilisé pour le pied de page
headerFontColor headerFontSize headerFontStyle headerFontName	black 10 plain	Utilisé pour l'entête de page

10.5 Noir et blanc

Vous pouvez forcer l'utilisation du noir et blanc en utilisant la commande `skinparam monochrome true`.

```
@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

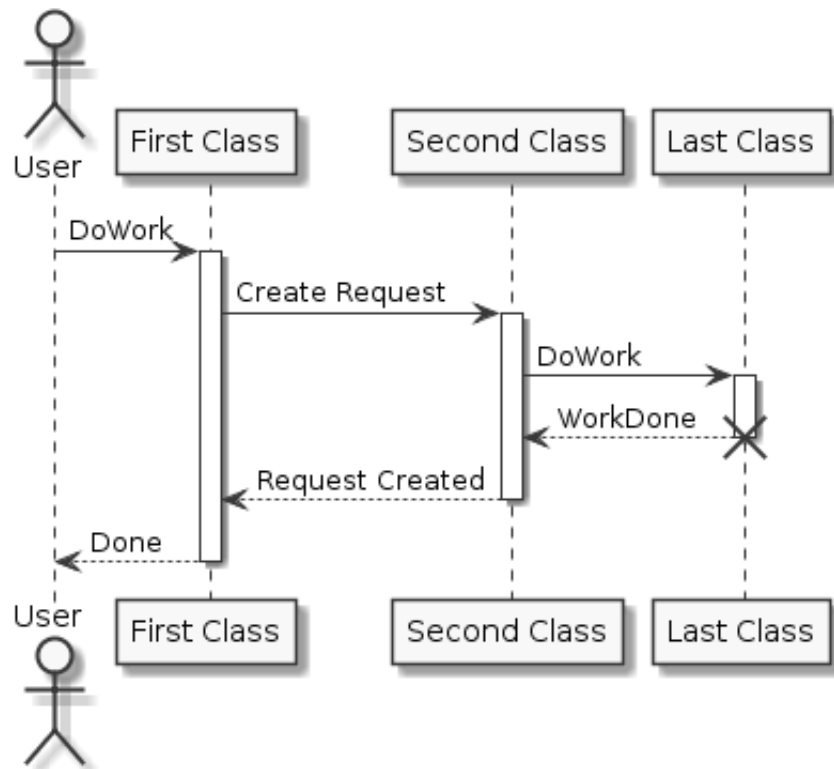
A -> B: Create Request
activate B

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

B --> A: Request Created
deactivate B

A --> User: Done
deactivate A

@enduml
```



11 Préprocesseur

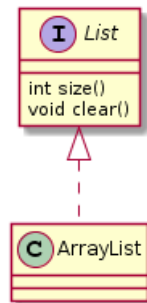
Quelques fonctionnalités de préprocesseur sont présentes dans PlantUML, et disponibles pour tous les diagrammes. Ces fonctionnalités sont très proches du langage C, sauf que le point d'exclamation "!" a été utilisé à la place du caractère "#" pour les directives.

11.1 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 inclu dans plusieurs fichiers, et tout changement dans ce fichier modifiera tous les diagrammes qui l'incluent.

Vous pouvez aussi mettre plusieurs `@startuml/@enduml` bloque de texte dans un fichier inclus et ensuite spécifier le bloque que vous voulez inclure en ajoutant `!0` où 0 est le numéro du bloque.

Par exemple, si vous utilisez `!include foo.txt!1`, le second `@startuml/@enduml` bloque à l'intérieur `foo.txt` sera inclus.

11.2 Inclusion d'URL

Utiliser la directive `!includeurl` pour inclure dans votre diagramme un fichier depuis Internet ou un Intranet.

Utilisez `!includeurl http://someurl.com/mypath!0` pour spécifier quel `@startuml/@enduml` bloque vous voulez inclure à partir de `http://someurl.com/mypath`. La notation `!0` désigne le premier diagramme.

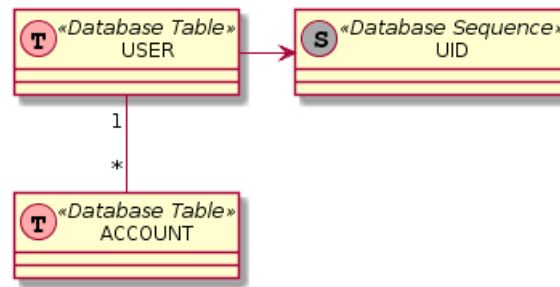
11.3 Définition de constantes

Vous pouvez définir les constantes en utilisant l'instruction `!define`. Comme en c, le nom d'une constante peut être n'importe quel mot-clé.

```
@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
```





Bien sur vous pouvez utiliser l'instruction `!include` pour définir toute les constantes dans un fichier simple que vous pouvez inclure dans votre diagramme.

Les constantes peuvent être définies avec l'instruction `!undef XXX`.

Vous pouvez aussi définir les constantes en ligne de commande, avec le "flag" `-D`.

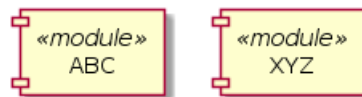
```
java -jar plantuml.jar -DTITLE="My title" atest1.txt
```

Notez que l'option `-D` doit être mise après la partie `-jar plantuml.jar`.

11.4 Macro définition

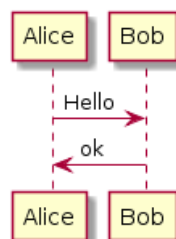
Vous pouvez alors définir des macros avec des arguments

```
@startuml
!define module(x) component x <<module>>
module(ABC)
module(XYZ)
@enduml
```



Les macros peuvent avoir plusieurs arguments

```
@startuml
!define send(a,b,c) a->b : c
send(Alice, Bob, Hello)
send(Bob, Alice, ok)
@enduml
```



11.5 Macro sur plusieurs lignes

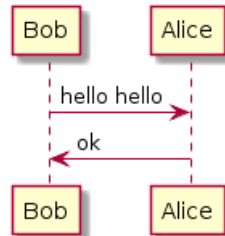
Vous pouvez alors définir des macros sur plusieurs lignes en utilisant `!definelong` et `!enddefinelong`.

```

@startuml
!define DOUBLE(x) x x
!definelong AUTHEN(x,y)
x -> y : DOUBLE(hello)
y -> x : ok
!enddefinelong

AUTHEN(Bob,Alice)
@enduml

```



11.6 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.

Les lignes qui sont entre les deux directives doivent être incluses seulement si la constante après `!ifdef` ont été définie avant.

Vous pouvez aussi utiliser `!else` pour une partie qui doit être incluse 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

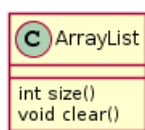
```

Vous pouvez alors utiliser la directive `!define` pour activer la part conditionnel du diagramme.

```

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

```



Vous pouvez alors utiliser la directive `!ifndef` qui inclut les lignes à condition que la constante NOT n'est pas été définie.



11.7 Chemin de recherche

Vous pouvez spécifier la propriété java "plantuml.include.path" en ligne de commande

Par exemple :

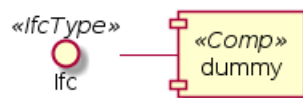
```
java -Dplantuml.include.path="c:/mydir" -jar plantuml.jar atest1.txt
```

Notez que l'option -D doit être utilisée avant l'option -jar. L'option -D après l'option -jar sera utilisée pour définir des constantes du pré-processeur de plantuml.

11.8 Caractéristiques avancées

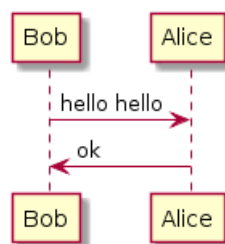
Il est possible d'ajouter du texte à une macro en utilisant la syntaxe ##.

```
@startuml
!definelong COMP_TEXTGENCOMP(name)
[name] << Comp >>
interface Ifc << IfcType >> AS name##Ifc
name##Ifc - [name]
!enddefinelong
COMP_TEXTGENCOMP(dummy)
@enduml
```



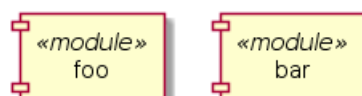
Une macro peut être défini par une autre macro.

```
@startuml
!define DOUBLE(x) x x
!definelong AUTHEN(x,y)
x -> y : DOUBLE(hello)
y -> x : ok
!enddefinelong
AUTHEN(Bob,Alice)
@enduml
```



Une macro peut être polymorphe avec un nombre d'argument

```
@startuml
!define module(x) component x <<module>>
!define module(x,y) component x as y <<module>>
module(foo)
module(bar, barcode)
@enduml
```



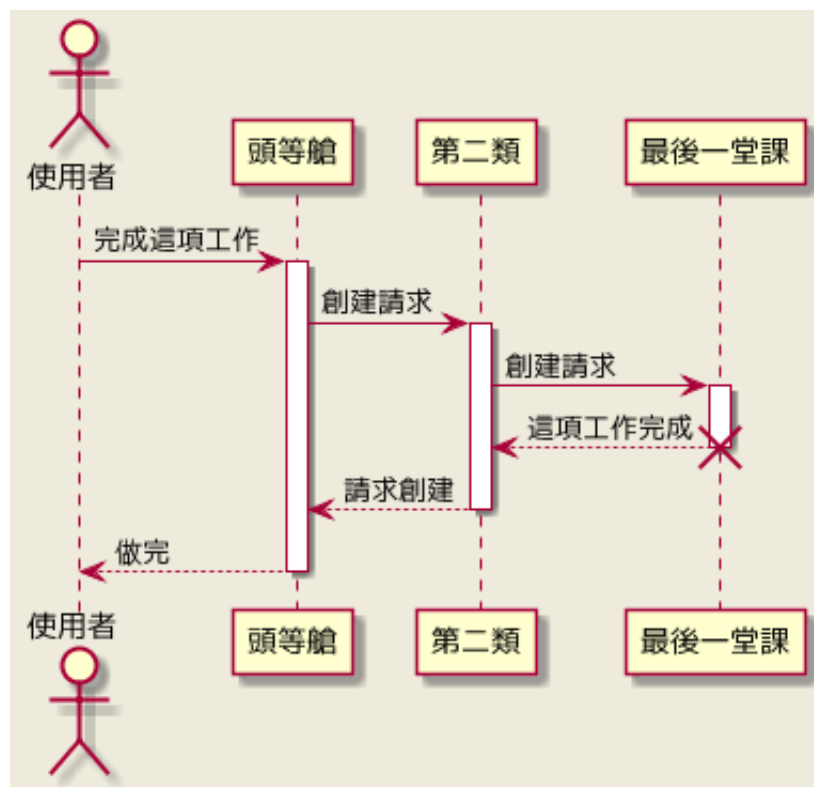
Vous pouvez utiliser une variable d'environnement ou définir une constante lorsque include est utilisé.

```
!include %windir%/test1.txt
!define PLANTUML_HOME /home/foo
!include PLANTUML_HOME/test1.txt
```

12 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
```



12.1 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:

```
<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.

13 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		



Contents

1	Diagramme de séquence	1
1.1	Exemples de base	1
1.2	Commentaires	1
1.3	Déclaration de participants	1
1.4	Caractères non alphanumérique dans les participants	2
1.5	Message à soi-même	3
1.6	Autre style de flèches	3
1.7	Changer la couleur des flèches	4
1.8	Numérotation automatique des messages	4
1.9	Titre	5
1.10	Spécifier la légende d'un diagramme	6
1.11	Découper un diagramme	6
1.12	Regrouper les messages (cadres UML)	7
1.13	Note sur les messages	8
1.14	Encore plus de notes	9
1.15	Changer l'aspect des notes	9
1.16	Créole (langage de balisage léger) et HTML	10
1.17	Séparation	11
1.18	Référence	11
1.19	Retard	12
1.20	Séparation verticale	12
1.21	Lignes de vie	13
1.22	Création de participants.	14
1.23	Messages entrant et sortant	15
1.24	Stéréotypes et décoration	16
1.25	Plus d'information sur les titres	17
1.26	Cadre pour les participants	18
1.27	Supprimer les en-pieds	18
1.28	Personnalisation	19
2	Diagramme de cas d'utilisation	21
2.1	Cas d'utilisation	21
2.2	Acteurs	21
2.3	Description des cas d'utilisation	21
2.4	Exemples très simples	22
2.5	Héritage	23
2.6	Notes	23
2.7	Stéréotypes	24
2.8	Changer les directions des flèches	24
2.9	Titrer le diagramme	25
2.10	Découper les diagrammes	26
2.11	De droite à gauche	26
2.12	La commande Skinparam	27
2.13	Exemple complet	28



3	Diagramme de classes	29
3.1	Relations entre classes	29
3.2	Libellés sur les relations	29
3.3	Définir les méthodes	31
3.4	Définir les visibilitées	32
3.5	Abstrait et statique	33
3.6	Corps de classe avancé	34
3.7	Notes et stéréotypes	35
3.8	Encore des notes	36
3.9	Note sur les liens	37
3.10	Classe abstraite et Interface	38
3.11	Caractères non alphabétiques	39
3.12	Masquer les attributs et les méthodes	40
3.13	Cacher des classes	41
3.14	Utilisation de la généricité	41
3.15	Caractère spécial	41
3.16	Packages	42
3.17	Modèle de paquet	42
3.18	Les espaces de noms	43
3.19	Creation automatique d'espace de nommage	44
3.20	Interface boucle	45
3.21	Changer la direction	45
3.22	Titre de diagramme	46
3.23	Diagramme de légende	46
3.24	Classes d'association	47
3.25	Personnalisation	48
3.26	Stéréotypes Personnalisés	48
3.27	Dégradé de couleur	49
3.28	Découper les grands diagrammes	50
4	Diagrammes d'activité	52
4.1	Exemple de base	52
4.2	Texte sur les flèches.	52
4.3	Changer la direction des flèches	52
4.4	Branches	53
4.5	Encore des branches	54
4.6	Synchronisation	55
4.7	Description détaillée	56
4.8	Notes	56
4.9	Partition	57
4.10	Mettre un titre	58
4.11	Paramètre de thème	59
4.12	Octogone	59
4.13	Exemple complet	60

5 Diagrammes d'activité (bêta)	63
5.1 Activité simple	63
5.2 Départ/Arrêt	63
5.3 Conditionnel	64
5.4 Boucle de répétition	65
5.5 Boucle While	65
5.6 Processus parallèle	66
5.7 Notes	66
5.8 Titre et légende	67
5.9 Couleurs	68
5.10 flèches	68
5.11 Groupement	69
5.12 Couloirs	70
5.13 Détacher	70
5.14 SDL	71
5.15 Exemple complet	72
6 Diagrammes de composants	74
6.1 Composants	74
6.2 Interfaces	74
6.3 Exemple simple	74
6.4 Mettre des notes	75
6.5 Regrouper des composants	75
6.6 Changer la direction des flèches	77
6.7 Ajouter un titre	78
6.8 Utiliser la notation UML2	78
6.9 Couleurs individuelles	78
6.10 Skinparam	79
7 Diagrammes d'état	81
7.1 Exemple simple	81
7.2 Etat composite	81
7.3 Nom long	82
7.4 Etat concurrent	83
7.5 Direction des flèches	84
7.6 Note	85
7.7 Plus de notes	86
7.8 Skinparam	86
8 Diagrammes d'objets	88
8.1 Définition des objets	88
8.2 Relations entre les objets	88
8.3 Ajout de champs	88
8.4 Caractéristiques communes avec les diagrammes de classes	89

9 Commandes communes	90
9.1 Entête et pied de page	90
9.2 Zoom	90
10 Modifier les polices et couleurs de caractères	91
10.1 Usage	91
10.2 Imbrication	91
10.3 Couleur	92
10.4 Couleur de police, nom et taille	93
10.5 Noir et blanc	96
11 Préprocesseur	97
11.1 Inclusion de fichier	97
11.2 Inclusion d'URL	97
11.3 Définition de constantes	97
11.4 Macro définition	98
11.5 Macro sur plusieurs lignes	98
11.6 Conditions	99
11.7 Chemin de recherche	100
11.8 Caractéristiques avancées	100
12 Internationalisation	102
12.1 Jeux de caractères	102
13 Nom des couleurs	104